Analysis of a Threshold Strategy in a Discrete-time Sparre Andersen Model

by

Ana Maria Mera

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirements for the degree of
Master of Mathematics
in
Actuarial Science

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis, it is shown that the application of a threshold on the surplus level of a particular discrete-time delayed Sparre Andersen insurance risk model results in a process that can be analyzed as a doubly infinite Markov chain with finite blocks. Two fundamental cases, encompassing all possible values of the surplus level at the time of the first claim, are explored in detail. Matrix analytic methods are employed to establish a computational algorithm for each case. The resulting procedures are then used to calculate the probability distributions associated with fundamental ruin-related quantities of interest, such as the time of ruin, the surplus immediately prior to ruin, and the deficit at ruin. The ordinary Sparre Andersen model, an important special case of the general model, with varying threshold levels is considered in a numerical illustration.

Acknowledgements

I would like to thank my supervisor, Steve Drekic, for his valuable input. Also, I want to thank Mary Lou Dufton for her assistance and Mary Hardy for her support. This work has been assisted by the Natural Sciences and Engineering Research Council of Canada.

I would like to thank my friends, Margareta Ackerman and Matei Zaharia, for their advice along the way. Also, thank you Mom, Dad, Andrei, Lelia, Radun, and Kevin - I hope to give back to you all that you have given to me.

Mother Mary, I owe you everything. Bubu, thanks for watching over me.

Contents

1 Notation and Preliminaries

In this thesis, a particular threshold strategy on the *delayed* Sparre Andersen (i.e. renewal risk) insurance risk model in discrete time is considered. The following definitions are implications of this model. First of all, the number of claims process $\{N_t : t = 0, 1, 2, \ldots\}$ is assumed to be a modified discrete-time renewal process with independent positive interclaim times $\{W_1, W_2, W_3, \ldots\}$, where W_1 is the duration from time 0 until the first claim occurs and W_i, $i = 2, 3, 4, \ldots$, is the time between the $(i-1)$-th and i-th claims. Secondly, it is assumed that $\{W_2, W_3, W_4, \ldots\}$ is an independent and identically distributed (iid) sequence of positive random variables with common probability mass funtion (pmf) $a_j = Pr\{W_i = j\}$, $j = 1, 2, 3, \ldots, n_a$, and corresponding survival function $A_j = Pr\{W_i > j\} = 1 - \sum_{k=1}^{j} a_k$. In this thesis, it is assumed that $n_a < \infty$ (i.e. the interclaim time distribution of W_i, $i = 2, 3, 4, \ldots$, has finite support).

In the *ordinary* Sparre Andersen model (a special case of the *delayed* Sparre Andersen model), it is assumed that a claim occurs at time 0, so that W_1 has the same distribution as the ordinary interclaim times $\{W_2, W_3, W_4, \ldots\}$. But if W_1 is not a "full" interclaim time, asymptotically in time, the limiting distribution of this forward recurrence time is defined by the pmf $\tilde{a}_j = A_{j-1} / \sum_{k=1}^{n_a} A_{k-1}$, $j = 1, 2, 3, \ldots, n_a$ (e.g. see Karlin and Taylor [1975, pp. 192-193]). This leads to another important special case of the *delayed* Sparre Andersen model; namely the *stationary* Sparre Andersen model, in which W_1 has pmf $\tilde{a}_j$ rather than a_j. However, to accommodate all possible specifications of the Sparre Andersen model, it is assumed in this thesis that W_1 has a more general pmf $r_j = Pr\{W_1 = j\}$, $j = 1, 2, 3, \ldots, n_r$, where $n_r < \infty$. By appropriate choice of r_j, it is obvious that both the ordinary and stationary Sparre Andersen models are special cases of this more general (*delayed* Sparre Andersen) risk model.

In this analysis, U_t represents the amount of surplus at the end of time interval $[t-1, t)$, at which point the premiums and claims corresponding to this time interval have been paid (out). This thesis analyzes the application of a threshold level $Z \in \mathbb{Z}^+$ on the amount of surplus, affecting the amount of premium being received at any given point in time. It is assumed that the delayed Sparre Andersen insurance risk model of interest in this thesis has premiums that are collected at the rate of $p_t \in \mathbb{Z}^+$ at time t, $t = 0, 1, 2, \ldots$, where

$$p_t = \begin{cases} c_1 & \text{if } U_t < Z, \\ c_2 & \text{if } U_t \geq Z, \end{cases} \tag{1.1}$$

with $c_1 > c_2$. Beginning with an initial reserve $u \in \{0, 1, 2, \ldots\}$, the insurer's surplus at time t is given by

$$U_t = u + \sum_{i=0}^{t-1} p_i - \sum_{i=1}^{N_t} Y_i, \ t = 0, 1, 2, \ldots \ , \tag{1.2}$$

where p_i is determined using (1.1). Individual claim amounts $\{Y_1, Y_2, Y_3, \ldots\}$ are assumed to form an iid sequence of positive random variables with common pmf α_j, $j = 1, 2, 3, \ldots, m_\alpha$, and corresponding survival function $\Lambda_j = 1 - \sum_{k=1}^{j} \alpha_k$. However, unlike the interclaim time distributions defined above, the claim amount distribution can be either of finite or infinite support (i.e. $m_\alpha \leq \infty$).

The premium c_1 below the threshold is chosen to be greater than c_2 for practical reasons, as Z is assumed to be determined by the insurer as the level at which there is a sufficiently ample amount of surplus allowing for the payout of dividends to shareholders. Subsequently, during this period of time, premiums are being received as dividends are simultaneously being paid out. This renders the overall intake of the insurer to be less than the amount of pure premium. Thus, when the insurer's funds reach a surplus level of Z, there is a decrease in the"premium" being received by the insurer.

At any given time point, the usual convention that premiums are collected first (i.e. at the beginning of the time interval) before any claims are paid (i.e. at the end of the time interval) is adopted. That is, premiums for the time interval $[t-1, t)$ are received at $(t-1)^+$ at rate p_{t-1}, and any claims are paid out at t^-. The time of ruin, T, is defined as $T = \min\{t \in \mathbb{Z}^+ | U_t < 0\}$ with $T = \infty$ if $U_t \geq 0 \ \forall \ t \in \mathbb{Z}^+$. If ruin does occur, $|U_T|$ is defined as the deficit at ruin and $U_{T^-} = U_{T-1} + p_{T-1}$ as the surplus immediately prior to ruin. Clearly, $T = \infty$ if $m_\alpha \leq \min\{c_1, Z + c_2\}$. However, if $m_\alpha > \min\{c_1, Z + c_2\}$, then $|U_T| \in \{1, 2, 3, \ldots, m_\alpha - \min\{c_1, Z + c_2\}\}$ and $U_{T^-} \in \{\min\{c_1, Z + c_2\}, \min\{c_1, Z + c_2\} + 1, \ldots, m_\alpha - 1\}$. Throughout the remainder of the thesis, it is assumed that $m_\alpha > \min\{c_1, Z + c_2\}$.

2 Formulation of the Model

Adopting the same notation used in the original formulation of a computational algorithm for the delayed Sparre Andersen model without a threshold (see Alfa and Drekic [2007] for details), the interclaim time distribution defined by the pmf a_j must first be considered. This pertains to an arbitrary W_i, $i = 2, 3, 4, \ldots$, denoted by W. Letting $\tau_j = Pr\{W > j | W > j-1\} = A_j/A_{j-1}, j = 1, 2, 3, \ldots, n_a$, it can be immediately seen that $\tau_1 = A_1$ and $\tau_{n_a} = 0$. The $n_a \times n_a$ probability transition matrix for the surviving waiting times to the next claim occurrence is then defined as

$$S = \begin{pmatrix} 0 & \tau_1 & 0 & \cdots & 0 \\ 0 & 0 & \tau_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \tau_{n_a-1} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}. \tag{2.1}$$

Also, defining the $1 \times n_a$ row vector $\boldsymbol{e}_1 = (1, 0, 0, \ldots, 0)$, and letting the $n_a \times 1$ column vector of absorption probabilities (to claim occurrence) be

$$\boldsymbol{s} = \begin{pmatrix} 1-\tau_1 \\ 1-\tau_2 \\ \vdots \\ 1-\tau_{n_a-1} \\ 1 \end{pmatrix},$$

it can be shown that (e.g. see Alfa [2004])

$$a_j = \boldsymbol{e}_1 S^{j-1} \boldsymbol{s}, \; j = 1, 2, 3, \ldots, n_a.$$

The above formula also holds true for $j > n_a$ since it can be easily verified that $S^{j-1} = O_{n_a}$ (i.e. an $n_a \times n_a$ matrix of zeros) for $j > n_a$. Alfa [2004] refers to this as the "elapsed time" representation of the pmf a_j.

In what follows, the delayed Sparre Andersen model (described in the introduction) will be set up as a two-dimensional Markov chain conditional on certain assumptions. The purpose of this is to allow for the formulation of a model that represents the transition process of the amount of surplus, U_t, under discussion, in matrix form. In order to construct such a process, W_1 must be isolated in the sense that W_1 is fixed to be k, where $k \in \{1, 2, 3, \ldots, n_r\}$. Assuming that $W_1 = k$, L_t (i.e. the second dimension of this Markov chain) is defined as the "elapsed interclaim time" (at time t, $t = k, k+1, k+2, \ldots$) since the occurrence of the most recent claim. For this specified range of t, the bivariate stochastic process (U_t, L_t), which possesses the following Markovian relationship, is considered:

$$(U_{t+1}, L_{t+1}) = \begin{cases} (U_t + p_t, L_t + 1) & \text{if there is no claim at time } (t+1)^-, \\ (U_t + p_t - Y, 1) & \text{if there is a claim of amount } Y \text{ at time } (t+1)^-. \end{cases} \tag{2.2}$$

Since it has been assumed that a claim occurred at time k, $L_k = 1$. The state space for this Markov chain, denoted by Δ, is then given by $\Delta = \{(U_t, L_t) : U_t \in \mathbb{Z}; L_t = 1, 2, 3, \ldots, n_a\}$. The U_t component is referred to as the *level* of the process (corresponding to the amount of surplus) and the L_t component is referred to as the *phase* of the process. Upon occurrence of the first claim, the delayed process reverts to the ordinary process (having interclaim time distribution defined by the pmf a_j). Hence, if $W_1 = k$, the probability transition matrix P associated with this two-dimensional Markov chain for $t = k, k+1, k+2, \ldots$ is given by

$$
P = \begin{array}{c c}
 & \begin{array}{ccccccccccc} \cdots & -1 & 0 & 1 & \cdots & Z-2 & Z-1 & Z & Z+1 & Z+2 & \cdots \end{array} \\
\begin{array}{c} \vdots \\ -1 \\ 0 \\ 1 \\ \vdots \\ Z-2 \\ Z-1 \\ Z \\ Z+1 \\ Z+2 \\ \vdots \end{array} &
\begin{pmatrix}
\ddots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \\
\ddots & B_{c_1} & B_{c_1-1} & B_{c_1-2} & \cdots & B_{c_1-Z+1} & B_{c_1-Z} & B_{c_1-Z-1} & B_{c_1-Z-2} & B_{c_1-Z-3} & \cdots \\
\ddots & B_{c_1+1} & B_{c_1} & B_{c_1-1} & \cdots & B_{c_1-Z+2} & B_{c_1-Z+1} & B_{c_1-Z} & B_{c_1-Z-1} & B_{c_1-Z-2} & \cdots \\
\ddots & B_{c_1+2} & B_{c_1+1} & B_{c_1} & \cdots & B_{c_1-Z+3} & B_{c_1-Z+2} & B_{c_1-Z+1} & B_{c_1-Z} & B_{c_1-Z-1} & \cdots \\
\ddots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \\
\ddots & B_{Z+c_1-1} & B_{Z+c_1-2} & B_{Z+c_1-3} & \cdots & B_{c_1} & B_{c_1-1} & B_{c_1-2} & B_{c_1-3} & B_{c_1-4} & \cdots \\
\ddots & B_{Z+c_1} & B_{Z+c_1-1} & B_{Z+c_1-2} & \cdots & B_{c_1+1} & B_{c_1} & B_{c_1-1} & B_{c_1-2} & B_{c_1-3} & \cdots \\
\ddots & B_{Z+c_2+1} & B_{Z+c_2} & B_{Z+c_2+1} & \cdots & B_{c_2+2} & B_{c_2+1} & B_{c_2} & B_{c_2-1} & B_{c_2-2} & \cdots \\
\ddots & B_{Z+c_2+2} & B_{Z+c_2+1} & B_{Z+c_2} & \cdots & B_{c_2+3} & B_{c_2+2} & B_{c_2+1} & B_{c_2} & B_{c_2-1} & \cdots \\
\ddots & B_{Z+c_2+3} & B_{Z+c_2+2} & B_{Z+c_2+1} & \cdots & B_{c_2+4} & B_{c_2+3} & B_{c_2+2} & B_{c_2+1} & B_{c_2} & \cdots \\
\ddots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots
\end{pmatrix}
\end{array}, \qquad (2.3)
$$

where

$$B_i = \begin{cases} O_{n_a} & \text{if } i \in \mathbb{Z}^-, \\ S & \text{if } i = 0, \\ (\boldsymbol{s}\boldsymbol{e}_1)\alpha_i & \text{if } i \in \mathbb{Z}^+. \end{cases} \tag{2.4}$$

So, for example, the transition "$U_t = -1 \rightarrow U_{t+1} = -1$" has block element B_{c_1}, which contains the transition probabilities corresponding to this mapping. The reasoning for block element B_{c_1} is as follows. At a surplus level of "-1" at time t^-, a premium of $p_t = c_1$ is certain to be received. Hence, to arrive at the same surplus level of "-1" at the next time unit $(t+1)^-$, a claim of size c_1 must occur to neutralize the premium just received. Thus, the size of the claim required for this transition to occur determines the value of i in B_i. Hence, B_{c_1} is obtained. Furthermore, the components of the matrix B_{c_1} govern the "$L_t \rightarrow L_{t+1}$" process. Note that this is a doubly infinite Markov chain with finite blocks of size n_a (e.g. see Grassmann [2000]). Also, since $\alpha_i = 0 \; \forall \; i > m_\alpha$, $B_i = O_{n_a}$ if $i > m_\alpha$, and this will aid in simplifying the formulation of an algorithm based on this model.

For the computation of ruin-related quantities of interest, it is useful to partition the state space Δ into two state spaces, namely

$$\Delta_1 = \{(i, j) : i = 0, 1, 2, \ldots \; ; j = 1, 2, \ldots, n_a\}$$

and

$$\Delta_2 = \{(i, j) : i = -1, -2, -3, \ldots \; ; j = 1, 2, \ldots, n_a\}.$$

Two matrices, defined as C and D where $C : \Delta_1 \rightarrow \Delta_1$ and $D : \Delta_1 \rightarrow \Delta_2$, correspond to mapping "non-ruined" states of the system to "non-ruined" states and "ruined" states, respectively. As a result, the two matrices C and D look as follows:

$$
C = \begin{array}{cc}
 & \begin{array}{cccccccccc} 0 & 1 & 2 & \cdots & Z-2 & Z-1 & Z & Z+1 & Z+2 & \cdots \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ \vdots \\ Z-3 \\ Z-2 \\ Z-1 \\ Z \\ Z+1 \\ Z+2 \\ Z+3 \\ \vdots \end{array} &
\begin{pmatrix}
B_{c_1} & B_{c_1-1} & B_{c_1-2} & \cdots & B_{c_1-Z+2} & B_{c_1-Z+1} & B_{c_1-Z} & B_{c_1-Z-1} & B_{c_1-Z-2} & \cdots \\
B_{c_1+1} & B_{c_1} & B_{c_1-1} & \cdots & B_{c_1-Z+3} & B_{c_1-Z+2} & B_{c_1-Z+1} & B_{c_1-Z} & B_{c_1-Z-1} & \cdots \\
B_{c_1+2} & B_{c_1+1} & B_{c_1} & \cdots & B_{c_1-Z+4} & B_{c_1-Z+3} & B_{c_1-Z+2} & B_{c_1-Z+1} & B_{c_1-Z} & \cdots \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \\
B_{Z+c_1-3} & B_{Z+c_1-4} & B_{Z+c_1-5} & \cdots & B_{c_1-1} & B_{c_1-2} & B_{c_1-3} & B_{c_1-4} & B_{c_1-5} & \cdots \\
B_{Z+c_1-2} & B_{Z+c_1-3} & B_{Z+c_1-4} & \cdots & B_{c_1} & B_{c_1-1} & B_{c_1-2} & B_{c_1-3} & B_{c_1-4} & \cdots \\
B_{Z+c_1-1} & B_{Z+c_1-2} & B_{Z+c_1-3} & \cdots & B_{c_1+1} & B_{c_1} & B_{c_1-1} & B_{c_1-2} & B_{c_1-3} & \cdots \\
B_{Z+c_2} & B_{Z+c_2-1} & B_{Z+c_2-2} & \cdots & B_{c_2+2} & B_{c_2+1} & B_{c_2} & B_{c_2-1} & B_{c_2-2} & \cdots \\
B_{Z+c_2+1} & B_{Z+c_2} & B_{Z+c_2-1} & \cdots & B_{c_2+3} & B_{c_2+2} & B_{c_2+1} & B_{c_2} & B_{c_2-1} & \cdots \\
B_{Z+c_2+2} & B_{Z+c_2+1} & B_{Z+c_2} & \cdots & B_{c_2+4} & B_{c_2+3} & B_{c_2+2} & B_{c_2+1} & B_{c_2} & \cdots \\
B_{Z+c_2+3} & B_{Z+c_2+2} & B_{Z+c_2+1} & \cdots & B_{c_2+5} & B_{c_2+4} & B_{c_2+3} & B_{c_2+2} & B_{c_2+1} & \cdots \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots
\end{pmatrix}
\end{array} \tag{2.5}
$$

and

$$D = \begin{array}{cc} & \begin{array}{cccc} -1 & -2 & -3 & \cdots \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ \vdots \\ Z-3 \\ Z-2 \\ Z-1 \\ Z \\ Z+1 \\ Z+2 \\ Z+3 \\ \vdots \end{array} & \begin{pmatrix} B_{c_1+1} & B_{c_1+2} & B_{c_1+3} & \cdots \\ B_{c_1+2} & B_{c_1+3} & B_{c_1+4} & \cdots \\ B_{c_1+3} & B_{c_1+4} & B_{c_1+5} & \cdots \\ \vdots & \vdots & \vdots & \cdots \\ B_{Z+c_1-2} & B_{Z+c_1-1} & B_{Z+c_1} & \cdots \\ B_{Z+c_1-1} & B_{Z+c_1} & B_{Z+c_1+1} & \cdots \\ B_{Z+c_1} & B_{Z+c_1+1} & B_{Z+c_1+2} & \cdots \\ B_{Z+c_2+1} & B_{Z+c_2+2} & B_{Z+c_2+3} & \cdots \\ B_{Z+c_2+2} & B_{Z+c_2+3} & B_{Z+c_2+4} & \cdots \\ B_{Z+c_2+3} & B_{Z+c_2+4} & B_{Z+c_2+5} & \cdots \\ B_{Z+c_2+4} & B_{Z+c_2+5} & B_{Z+c_2+6} & \cdots \\ \vdots & \vdots & \vdots & \cdots \end{pmatrix} \end{array}. \tag{2.6}$$

In this thesis, two fundamental cases are considered, encompassing all possible values of the surplus level at the time of the first claim. These cases essentially correspond to the differing values of the *initial* (i.e. at time k) probability vector of the states in Δ_1, $\boldsymbol{b}^{(k)}$, which will be considered separately for each situation in the next section. Matrix analytic methods will then be used to establish a computational algorithm for each scenario. The algorithms that will be derived serve the ultimate purpose of calculating the probability distributions associated with fundamental ruin-related quantities of interest.

The general notation necessary for all further computations will now be specified. First of all, let

$$x_t = \begin{cases} 0 & \text{if } u \geq Z, \\ \max\{i \in \{1, 2, \ldots, t\} | u + c_1(i-1) < Z\} & \text{if } u < Z. \end{cases} \tag{2.7}$$

Secondly, define the function $f_n(t) = u + c_1 t + c_2(n-t)$. Then, the aforementioned $\boldsymbol{b}^{(k)}$ is generalized to be

$$\boldsymbol{b}^{(k)} = (\alpha_{f_k(x_k)}\boldsymbol{e}_1, \alpha_{f_k(x_k)-1}\boldsymbol{e}_1, \alpha_{f_k(x_k)-2}\boldsymbol{e}_1, \ldots, \alpha_2\boldsymbol{e}_1, \alpha_1\boldsymbol{e}_1, \boldsymbol{0}, \boldsymbol{0}, \ldots), \tag{2.8}$$

where $\boldsymbol{0}$ denotes the $1 \times n_a$ row vector of zeros. The i-th level of $\boldsymbol{b}^{(k)}$ is given by the $1 \times n_a$ row vector $\alpha_{f_k(x_k)-i}\boldsymbol{e}_1$ for each $i \in \Omega_k = \{0, 1, 2, \ldots, f_k(x_k) - 1\}$. Furthermore, two additional row vectors are needed, namely

$$\boldsymbol{g}_n^{(k)} = (\boldsymbol{g}_{n,0}^{(k)}, \boldsymbol{g}_{n,1}^{(k)}, \boldsymbol{g}_{n,2}^{(k)}, \ldots) = \boldsymbol{b}^{(k)} C^n, \ n = 0, 1, 2, \ldots \tag{2.9}$$

and

$$\boldsymbol{h}_n^{(k)} = (\boldsymbol{h}_{n,-1}^{(k)}, \boldsymbol{h}_{n,-2}^{(k)}, \boldsymbol{h}_{n,-3}^{(k)}, \ldots) = \boldsymbol{g}_{n-1}^{(k)} D = \boldsymbol{b}^{(k)} C^{n-1} D, \ n = 1, 2, 3, \ldots \ . \tag{2.10}$$

Note that $\boldsymbol{g}_n^{(k)}$ contains the probabilities of being in the various "non-ruined" states at time $k+n$ (i.e. after claims have been paid out for the time period $[k+n-1, k+n)$) without having visited a "ruined" state during the previous $n-1$ transitions, given that $U_k \in \Omega_k$ according to the probability vector $\boldsymbol{b}^{(k)}$. In a similar fashion, $\boldsymbol{h}_n^{(k)}$ contains the probabilities of being in the various "ruined" states *for the first time* at time $k+n$, given that $U_k \in \Omega_k$ according to the probability vector $\boldsymbol{b}^{(k)}$. Note that the probability of being in ruined state "$-j$" is governed by the vector $\boldsymbol{h}_{n,-j}^{(k)} = (h_{n,-j,1}^{(k)}, h_{n,-j,2}^{(k)}, \ldots, h_{n,-j,n_a}^{(k)})$ with the third subscript component representing the value of $L_{k+n} \in \{1, 2, \ldots, n_a\}$. However, upon further reflection, it must be the case that $h_{n,-j,i}^{(k)} = 0$ for $i \neq 1$, as ruin can only occur at claim instants, which, in this case, implies $L_{k+n} = 1$ with probability 1. Thus, it immediately follows that the structure of the $1 \times n_a$ row vector $\boldsymbol{h}_{n,-j}^{(k)}$ is simply given by $\boldsymbol{h}_{n,-j}^{(k)} = (\phi_{n,j}^{(k)}(u), 0, 0, \ldots, 0)$, where $\phi_{n,j}^{(k)}(u) = Pr\,\{T = k+n, |U_T| = j \mid U_k \in \Omega_k\}$. Consequently, as in the analysis with no threshold on the surplus level, the following result is obtained:

$$\phi_{n,j}^{(k)}(u) = \boldsymbol{h}_{n,-j}^{(k)} \boldsymbol{e}_1', \tag{2.11}$$

where $\boldsymbol{e}_1'$ denotes the transpose of $\boldsymbol{e}_1$.

Similar probabilistic reasoning can be applied to obtain a representation for $\psi_{n,i,j}^{(k)}(u) = Pr\,\{T = k+n, U_{T-} = i, |U_T| = j \mid U_k \in \Omega_k\}$. In order for ruin to occur at time $k+n$ with a surplus prior to ruin equal to i, (i) none of the previous $n-1$ transitions must have included a visit to any state in Δ_2, and (ii) the surplus level at time $k+n-1$ must be equal to $i - p_{k+n-1}$. Note that p_{k+n-1} represents the corresponding premium for the time interval $[k+n-1, k+n)$, which is received at $(k+n-1)^+$. The quantity corresponding to points (i) and (ii) is the $1 \times n_a$ row vector $\mathbf{g}_{n-1,i-p_{k+n-1}}^{(k)}$. At the next time unit (i.e. time $(k+n)^-$), a claim must necessarily occur but not before a premium of p_{k+n-1} is first collected, thereby raising the surplus level to i. Since $\boldsymbol{s}$ contains the absorption probabilities (to claim occurrence) from the n_a possible phase states, and the claim causing ruin must be of size $i+j$ in order to ensure that the deficit at ruin is equal to j, it follows that

$$\psi_{n,i,j}^{(k)}(u) = \boldsymbol{g}_{n-1,i-p_{k+n-1}}^{(k)} \ \boldsymbol{s} \ \alpha_{i+j}. \tag{2.12}$$

If $\min\{c_1, Z + c_2\} = c_1$, equation (2.12) simplifies to give

$$\psi_{n,i,j}^{(k)}(u) = \begin{cases} \mathbf{g}_{n-1,i-c_1}^{(k)} \ \boldsymbol{s} \ \alpha_{i+j} & \text{if } i = c_1, c_1 + 1, \ldots, Z + c_2 - 1, \\ (\mathbf{g}_{n-1,i-c_1}^{(k)} \ + \ \mathbf{g}_{n-1,i-c_2}^{(k)}) \ \boldsymbol{s} \ \alpha_{i+j} & \text{if } i = Z + c_2, Z + c_2 + 1, \ldots, Z + c_1 - 1, \\ \mathbf{g}_{n-1,i-c_2}^{(k)} \ \boldsymbol{s} \ \alpha_{i+j} & \text{if } i = Z + c_1, Z + c_1 + 1, \ldots, m_\alpha - 1. \end{cases} \tag{2.13}$$

Conversely, if $\min\{c_1, Z + c_2\} = Z + c_2$, equation (2.12) simplifies to give

$$\psi_{n,i,j}^{(k)}(u) = \begin{cases} \mathbf{g}_{n-1,i-c_2}^{(k)} \ \boldsymbol{s} \ \alpha_{i+j} & \text{if } i = Z + c_2, Z + c_2 + 1, \ldots, c_1 - 1, \\ (\mathbf{g}_{n-1,i-c_1}^{(k)} \ + \ \mathbf{g}_{n-1,i-c_2}^{(k)}) \ \boldsymbol{s} \ \alpha_{i+j} & \text{if } i = c_1, c_1 + 1, \ldots, Z + c_1 - 1, \\ \mathbf{g}_{n-1,i-c_2}^{(k)} \ \boldsymbol{s} \ \alpha_{i+j} & \text{if } i = Z + c_1, Z + c_1 + 1, \ldots, m_\alpha - 1. \end{cases} \tag{2.14}$$

The justification of (2.13) and (2.14) is as follows. If c_1 is less than $Z + c_2$, the only way of reaching a surplus level of $i = c_1, c_1+1, \ldots, Z+c_2-1$ (prior to ruin) is by receiving a premium of c_1 since $i - c_2$ (for the specified i) would result in a surplus level below Z, where c_2 is not applicable. Next, for $i = Z + c_2, Z + c_2 + 1, \ldots, Z + c_1 - 1$, there are two ways of reaching this surplus level. Note that subtracting c_2 from i in this range results in a surplus level greater than or equal to Z, and that reducing each of these i's by c_1 yields a value less than Z. Hence, both of these premiums are plausible here. Lastly, for $i = Z + c_1, Z + c_1, \ldots, m_\alpha - 1$, decreasing this surplus level by c_2 results in a value greater than Z (as desired), but reducing it by c_1 also has the same impact, thus eliminating this premium option. The justification when c_1 is greater than $Z + c_2$ mirrors the analysis above.

expressions

Recall that in the analysis up to this point, it has been assumed that $W_1 = k$, where $k \in \{1, 2, 3, \ldots, n_r\}$. Conditioning on the value of W_1 yields, by the Law of Total Probability, the following two expressions:

$$\phi_{n,j}(u) = \sum_{k=1}^{n_r} r_k \phi_{n-k,j}^{(k)}(u), \ \ n = n_r + 1, n_r + 2, n_r + 3, \ldots \tag{2.15}$$

and

$$\psi_{n,i,j}(u) = \sum_{k=1}^{n_r} r_k \psi_{n-k,i,j}^{(k)}(u), \ \ n = n_r + 1, n_r + 2, n_r + 3, \ldots \ . \tag{2.16}$$

In order for $T = n$ when $n = 1, 2, \ldots, n_r$, the only possible ways of ruin occurring are: (i) the first claim, which causes ruin, occurs at time n, or (ii) the first claim, which does not cause ruin, occurs at some time $k \in \{1, 2, 3, \ldots, n - 1\}$, and ruin subsequently occurs $n - k$ time units later. Now, combining the outcomes for $n = 1, 2, \ldots, n_r$ with those for

$n = n_r + 1, n_r + 2, n_r + 3, \ldots$ (and recalling that $r_n = 0 \;\forall\; n > n_r$), the expression for $\phi_{n,j}(u)$ becomes

$$\phi_{n,j}(u) = \sum_{k=1}^{\min\{n-1,n_r\}} r_k \phi_{n-k,j}^{(k)}(u) + r_n \alpha_{f_n(x_n)+j}, \; n \in \mathbb{Z}^+. \tag{2.17}$$

Additionally, for $\psi_{n,i,j}(u)$, when $n \leq n_r$ and component (i) above is considered, i must equal $f_n(x_n)$ for ruin to occur in this manner. Hence,

$$\psi_{n,i,j}(u) = \sum_{k=1}^{\min\{n-1,n_r\}} r_k \psi_{n-k,i,j}^{(k)}(u) + \delta_{i,f_n(x_n)} r_n \alpha_{f_n(x_n)+j}, \; n \in \mathbb{Z}^+, \tag{2.18}$$

where $\delta_{i,f_n(x_n)}$ denotes the Kronecker delta function of i and $f_n(x_n)$ (i.e. $\delta_{i,f_n(x_n)} = 1$ if $i = f_n(x_n)$ and 0 otherwise). To obtain $\omega_{n,i}(u)$ for fixed $i \in \{\min\{c_1, Z + c_2\}, \min\{c_1, Z + c_2\} + 1, \ldots, m_\alpha - 1\}$, summing (2.18) from $j = 1$ to $m_\alpha - i$ yields

$$\begin{aligned}\omega_{n,i}(u) &= \sum_{k=1}^{\min\{n-1,n_r\}} r_k \sum_{j=1}^{m_\alpha - i} \psi_{n-k,i,j}^{(k)}(u) + \delta_{i,f_n(x_n)} r_n \sum_{j=1}^{m_\alpha - i} \alpha_{f_n(x_n)+j} \\ &= \Lambda_i \sum_{k=1}^{\min\{n-1,n_r\}} r_k (\mathbf{g}_{n-k-1,i-p_{n-1}}^{(k)} \boldsymbol{s}) + \delta_{i,f_n(x_n)} r_n \left(\Lambda_{f_n(x_n)} - \Lambda_{f_n(x_n)+m_\alpha-i}\right), n \in \mathbb{Z}^+, \end{aligned} \tag{2.19}$$

where (2.12) was used to establish the last equality. Clearly, $\Lambda_{f_n(x_n)+m_\alpha-i} = 0$ in the above formula for $\omega_{n,i}(u)$ if $m_\alpha = \infty$.

3 Computational Procedure

As in the determination of the minimum and maximum values for the deficit at ruin and surplus prior to ruin random variables in Section 1, the quantities $\min\{c_1, Z+c_2\}$ and $\max\{c_1, Z+c_2\}$ play a significant role in the derivation of the computational procedure, in combination with the form of the probability transition matrix P defined by (2.3). To obtain a general algorithm for computing $\boldsymbol{g}_{n,i}^{(k)}$ for all i, and consequently, for $\boldsymbol{g}_n^{(k)}$, the crucial identity $\boldsymbol{g}_n^{(k)} = \boldsymbol{g}_{n-1}^{(k)} C$ for $n \in \mathbb{Z}^+$, which can be inferred from (2.9), is used. Since $\boldsymbol{g}_0^{(k)} = \boldsymbol{b}^{(k)}$ contains zeros from a certain level onwards (i.e. level $f_k(x_k)$), along with the fact that the block elements, B_i, of the C matrix eventually become zero matrices, it can be concluded that $\boldsymbol{g}_n^{(k)}$ will contain zeros from a certain level onwards. This particular level, which is later determined on a case-by-case basis, will generally be denoted by $l(n)$. That is, $l(n-1)$ represents the level from which $\boldsymbol{g}_{n-1}^{(k)}$ contains zeros onwards. Applying these useful facts, the following basic equation is obtained for any $i \in \{0, 1, 2, \ldots, Z-1, Z, Z+1, \ldots, l(n)-1\}$:

$$\boldsymbol{g}_{n,i}^{(k)} = \sum_{j=0}^{Z-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_1} + \sum_{j=Z}^{l(n-1)-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_2}. \tag{3.1}$$

However, since $B_i = O_{n_a}$ for $i \in \mathbb{Z}^-$ as per (2.4), (3.1) becomes (for the same range of i)

$$\boldsymbol{g}_{n,i}^{(k)} = \sum_{j=\max\{0, i-c_1\}}^{Z-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_1} + \sum_{j=\max\{Z, i-c_2\}}^{l(n-1)-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_2}. \tag{3.2}$$

Next breaking (3.2) into cases determined by the value of i, the following computational procedure is constructed:

For $i \in \{0, 1, 2, \ldots, \min\{c_1, Z+c_2\}-1\}$,

$$\boldsymbol{g}_{n,i}^{(k)} = \sum_{j=0}^{Z-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_1} + \sum_{j=Z}^{l(n-1)-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_2}.$$

For $i \in \{\min\{c_1, Z+c_2\}, \min\{c_1, Z+c_2\}+1, \ldots, \max\{c_1, Z+c_2\}-1\}$,

$$\boldsymbol{g}_{n,i}^{(k)} = \begin{cases} \sum_{j=i-c_1}^{Z-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_1} + \sum_{j=Z}^{l(n-1)-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_2} \\ \quad \text{if } \max\{c_1, Z+c_2\} = Z+c_2, \\ \\ \sum_{j=0}^{Z-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_1} + \sum_{j=i-c_2}^{l(n-1)-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_2} \\ \quad \text{if } \max\{c_1, Z+c_2\} = c_1. \end{cases}$$

For $i \in \{\max\{c_1, Z+c_2\}, \max\{c_1, Z+c_2\}+1, \ldots, Z+c_1-1\}$,

$$\boldsymbol{g}_{n,i}^{(k)} = \sum_{j=i-c_1}^{Z-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_1} + \sum_{j=i-c_2}^{l(n-1)-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_2}.$$

For $i \in \{Z+c_1, Z+c_1+1, Z+c_1+2, \ldots, l(n)-1\}$,

$$\boldsymbol{g}_{n,i}^{(k)} = \sum_{j=i-c_2}^{l(n-1)-1} \boldsymbol{g}_{n-1,j}^{(k)} B_{j-i+c_2}.$$

the section where the specific case requiring these restrictions arises.

Substituting (2.4) into the above equations, the following general recursive procedure for computing the $1 \times n_a$ row vector $\boldsymbol{g}_n^{(k)}$, $n \in \mathbb{Z}^+$, can then be constructed:
Let $i \in \{0, 1, 2, \ldots, \min\{c_1, Z+c_2\}-1\}$. Then:

$$\boldsymbol{g}_{n,i}^{(k)} = \left(\sum_{j=0}^{Z-1} \alpha_{j-i+c_1} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s} + \sum_{j=Z}^{l(n-1)-1} \alpha_{j-i+c_2} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s}, 0, 0, \ldots, 0 \right). \tag{3.3}$$

Let $i \in \{\min\{c_1, Z+c_2\}, \min\{c_1, Z+c_2\}+1, \ldots, \max\{c_1, Z+c_2\}-1\}$. Then:

$$\boldsymbol{g}_{n,i}^{(k)} = \begin{cases} \boldsymbol{g}_{n-1,i-c_1}^{(k)} S + \left(\sum_{j=i-c_1+1}^{Z-1} \alpha_{j-i+c_1} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s} + \sum_{j=Z}^{l(n-1)-1} \alpha_{j-i+c_2} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s}, 0, 0, \ldots, 0 \right) \\ \qquad \text{if } \max\{c_1, Z+c_2\} = Z+c_2, \\ \\ \boldsymbol{g}_{n-1,i-c_2}^{(k)} S + \left(\sum_{j=0}^{Z-1} \alpha_{j-i+c_1} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s} + \sum_{j=i-c_2+1}^{l(n-1)-1} \alpha_{j-i+c_2} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s}, 0, 0, \ldots, 0 \right) \\ \qquad \text{if } \max\{c_1, Z+c_2\} = c_1. \end{cases} \tag{3.4}$$

Let $i \in \{\max\{c_1, Z+c_2\}, \max\{c_1, Z+c_2\}+1, \ldots, Z+c_1-1\}$. Then:

$$\boldsymbol{g}_{n,i}^{(k)} = \boldsymbol{g}_{n-1,i-c_1}^{(k)} S + \boldsymbol{g}_{n-1,i-c_2}^{(k)} S + \left(\sum_{j=i-c_1+1}^{Z-1} \alpha_{j-i+c_1} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s} + \sum_{j=i-c_2+1}^{l(n-1)-1} \alpha_{j-i+c_2} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s}, 0, 0, \ldots, 0 \right). \tag{3.5}$$

Let $i \in \{Z+c_1, Z+c_1+1, Z+c_1+2, \ldots, l(n)-1\}$. Then:

$$\boldsymbol{g}_{n,i}^{(k)} = \boldsymbol{g}_{n-1,i-c_2}^{(k)} S + \left(\sum_{j=i-c_2+1}^{l(n-1)-1} \alpha_{j-i+c_2} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s}, 0, 0, \ldots, 0 \right). \tag{3.6}$$

The details pertaining to the values of $l(n-1)$ and $l(n)$ in the above-deduced algorithm are specified in what follows.

3.1 Case 1: $u + c_1 x_k + c_2(k - x_k) - 1 \geq Z$

For $W_1 = k$ and $u + c_1 x_k + c_2(k - x_k) - 1 \geq Z$, it is possible that x_k can take on any value in the set $\{0, 1, \ldots, k\}$. The *initial* (i.e. starting at time k) probability row vector corresponding to the states in Δ_1 is given by $\boldsymbol{g}_0^{(k)} = \boldsymbol{b}^{(k)}$ as defined generally in (2.8). Note that the i-th level of $\boldsymbol{g}_0^{(k)}$ is given by the $1 \times n_a$ row vector $\alpha_{f_k(x_k)-i}\boldsymbol{e}_1$ for each $i \in \Omega_k = \{0, 1, 2, \ldots, f_k(x_k) - 1\}$. Moreover, $\boldsymbol{g}_0^{(k)}$ contains zeros from level $f_k(x_k)$ onwards. Hence, $l(0) = f_k(x_k)$.

In the computation of $\boldsymbol{g}_1^{(k)}$ given by (2.9), C is pre-multiplied by the row vector $\boldsymbol{g}_0^{(k)}$. Since $\boldsymbol{g}_0^{(k)}$ contains zeros from level $f_k(x_k)$ onwards, and it might be possible to obtain a higher value than $f_{k+1}(x_k) - 1$ (i.e. the maximum level with no drop below the threshold) if the surplus had fallen below Z at time k, then gained c_1, and so reached a bounded level of $Z + c_1 - 1$ at time $k + 1$, it is deduced that $\boldsymbol{g}_1^{(k)}$ has the form

$$\boldsymbol{g}_1^{(k)} = (\boldsymbol{g}_{1,0}^{(k)}, \boldsymbol{g}_{1,1}^{(k)}, \boldsymbol{g}_{1,2}^{(k)}, \ldots, \boldsymbol{g}_{1,\max\{Z+c_1, f_{k+1}(x_k)\}-2}^{(k)}, \boldsymbol{g}_{1,\max\{Z+c_1, f_{k+1}(x_k)\}-1}^{(k)}, \boldsymbol{0}, \boldsymbol{0}, \ldots),$$

Hence, $l(1) = \max\{Z + c_1, f_{k+1}(x_k)\}$. From $n \geq 1$ onwards, since at any future point the surplus could possibly fall below Z, the above argument can be carried out inductively to obtain

$$\boldsymbol{g}_n^{(k)} = (\boldsymbol{g}_{n,0}^{(k)}, \boldsymbol{g}_{n,1}^{(k)}, \boldsymbol{g}_{n,2}^{(k)}, \ldots, \boldsymbol{g}_{n,\max\{f_{k+n}(x_k), Z+c_1+c_2(n-1)\}-2}^{(k)}, \boldsymbol{g}_{n,\max\{f_{k+n}(x_k), Z+c_1+c_2(n-1)\}-1}^{(k)}, \boldsymbol{0}, \boldsymbol{0}, \ldots).$$

This implies $l(n) = \max\{f_{k+n}(x_k), Z + c_1 + c_2(n - 1)\}$.

Thus, in this first case considered, $\boldsymbol{g}_{n,i}^{(k)}$ for all i (and, consequently, $\boldsymbol{g}_n^{(k)}$) can be derived by applying the final general recursion (i.e. equations (3.3) through (3.6)) where

$$l(n) = \begin{cases} u + c_1 x_k + c_2(k - x_k) & \text{if } n = 0, \\ \max\{u + c_1 x_k + c_2(k + n - x_k), Z + c_1 + c_2(n - 1)\} & \text{if } n \in \mathbb{Z}^+, \end{cases} \tag{3.1.1}$$

starting with $\boldsymbol{g}_{0,j}^{(k)} = \alpha_{u+c_1x_k+c_2(k-x_k)-j}\boldsymbol{e}_1$, $j = 0, 1, 2, \ldots, u + c_1 x_k + c_2(k - x_k) - 1$.

In addition, since, $\boldsymbol{g}_{n,\max\{f_k(x_k), Z+c_1-c_2\}+\ell}^{(k)} = \boldsymbol{0}\ \forall\ \ell = c_2 n, c_2 n + 1, c_2 n + 2, \ldots$, it can be seen from (2.4), (2.6), and (2.10) that for $n \in \mathbb{Z}^+$

$$\begin{aligned} \boldsymbol{h}_{n,-j}^{(k)} &= \sum_{\ell=c_1}^{Z+c_1-1} \boldsymbol{g}_{n-1,\ell-c_1}^{(k)} B_{j+\ell} + \sum_{\ell=Z+c_2}^{\max\{f_{k+n}(x_k), Z+c_1+c_2(n-1)\}-1} \boldsymbol{g}_{n-1,\ell-c_2}^{(k)} B_{j+\ell} \\ &= \left(\sum_{\ell=c_1}^{Z+c_1-1} \alpha_{j+\ell}\boldsymbol{g}_{n-1,\ell-c_1}^{(k)}\boldsymbol{s} + \sum_{\ell=Z+c_2}^{\max\{f_{k+n}(x_k), Z+c_1+c_2(n-1)\}-1} \alpha_{j+\ell}\boldsymbol{g}_{n-1,\ell-c_2}^{(k)}\boldsymbol{s}, 0, 0, \ldots, 0 \right). \end{aligned} \tag{3.1.2}$$

Hence, substituting (3.1.2) into (2.11) immediately yields

$$\phi_{n,j}^{(k)}(u) = \sum_{\ell=c_1}^{Z+c_1-1} \alpha_{j+\ell} \boldsymbol{g}_{n-1,\ell-c_1}^{(k)} \boldsymbol{s} + \sum_{\ell=Z+c_2}^{\max\{f_{k+n}(x_k),Z+c_1+c_2(n-1)\}-1} \alpha_{j+\ell} \boldsymbol{g}_{n-1,\ell-c_2}^{(k)} \boldsymbol{s}. \qquad (3.1.3)$$

Finally, (3.1.3) can then be substituted into (2.17) to obtain a form for $\phi_{n,j}(u)$.

3.2 Case 2: $u + c_1k - 1 < Z$

For this case, suppose that $x_k = k$ such that $u+c_1x_k+c_2(k-x_k)-1 = u+c_1k-1 < Z$, or equivalently, $u+c_1k \leq Z$. Define $[x]$ to be the smallest nonnegative integer greater than or equal to x. Because a premium of c_2 cannot be received until level Z is at least achieved, it is essential to find the point at which the threshold level Z is reached (i.e. the point at which $u + c_1k - 1 + c_1 i = Z$). Solving this equation for i, one obtains $i = \frac{Z-(u+c_1k)+1}{c_1}$. Hence, assuming that $W_1 = k$, the elapsed time after k at which a premium of c_2 would begin to be received, t_k^*, is thus $t_k^* = [\frac{Z-(u+c_1k)+1}{c_1}]$. Note that $t_k^* \geq 1$.

With $x_k = k$, it readily follows that (2.8) simplifies to give

$$\boldsymbol{g}_0^{(k)} = (\alpha_{u+c_1k}\boldsymbol{e}_1, \alpha_{u+c_1k-1}\boldsymbol{e}_1, \alpha_{u+c_1k-2}\boldsymbol{e}_1, \ldots, \alpha_2\boldsymbol{e}_1, \alpha_1\boldsymbol{e}_1, \boldsymbol{0}, \boldsymbol{0}, \ldots). \qquad (3.2.1)$$

Note that the i-th level of $\boldsymbol{g}_0^{(k)}$ is given by the $1 \times n_a$ row vector $\alpha_{u+c_1k-i}\boldsymbol{e}_1$ for each $i \in \Omega_k = \{0, 1, 2, \ldots, u + c_1k - 1\}$. Moreover, $\boldsymbol{g}_0^{(k)}$ contains zeros from level $u + c_1k$ onwards. Hence, $l(0) = u + c_1k$.

For the first time following k, a premium of c_1 is sure to be received at time k^+ as the threshold level has not yet been reached (recalling that $t_k^* \geq 1$). Even in the minimal case that $t_k^* = 1$, the effect of the new premium will first have an impact at time $(k+1)^+$. As a result, $\boldsymbol{g}_1^{(k)}$ contains zeros from level $u + c_1(k+1)$ onwards, which is c_1 levels further than that in $\boldsymbol{g}_0^{(k)} = \boldsymbol{b}^{(k)}$. Thus, $\boldsymbol{g}_1^{(k)}$ can be written in the form

$$\boldsymbol{g}_1^{(k)} = (\boldsymbol{g}_{1,0}^{(k)}, \boldsymbol{g}_{1,1}^{(k)}, \boldsymbol{g}_{1,2}^{(k)}, \ldots, \boldsymbol{g}_{1,u+c_1(k+1)-2}^{(k)}, \boldsymbol{g}_{n,u+c_1(k+1)-1}^{(k)}, \boldsymbol{0}, \boldsymbol{0}, \ldots).$$

Continuing this process inductively, it can be established that $\boldsymbol{g}_n^{(k)}$ contains zeros from level $u + c_1(k + n)$ onwards for $n = 1, 2, \ldots, t_k^*$, so that

$$\boldsymbol{g}_n^{(k)} = (\boldsymbol{g}_{n,0}^{(k)}, \boldsymbol{g}_{n,1}^{(k)}, \boldsymbol{g}_{n,2}^{(k)}, \ldots, \boldsymbol{g}_{n,u+c_1(k+n)-2}^{(k)}, \boldsymbol{g}_{n,u+c_1(k+n)-1}^{(k)}, \boldsymbol{0}, \boldsymbol{0}, \ldots).$$

Thus, $l(n) = u+c_1(k+n)$ for $n = 0, 1, 2, \ldots, t_k^*$. This particular case of $n \leq t_k^*$ necessitates additional restrictions on the upper limits of both i and some of the summation terms present in the general recursive procedure for $\boldsymbol{g}_{n,i}^{(k)}$ given by equations (3.3) through (3.6). Specifically, in the general recursive formula, restrictions on the bounds for i in $\boldsymbol{g}_{n,i}^{(k)}$ and on the upper bound of the sum term that would otherwise be $Z-1$ in $\boldsymbol{g}_{n,i}^{(k)}$ are induced by this case. Since t_k^* is defined such that $u+c_1k-1+c_1t_k^* \geq Z$ and $u+c_1k-1+c_1n < Z$ for $n < t_k^*$, then $l(n-1)-1 = u+c_1(k+n-1)-1$ (i.e. the ultimate upper limit on the sum terms in the general recursion) is certain to be less than Z (or less than or equal to $Z-1$) for $n = 1, 2, \ldots, t_k^*$. However, for $n > t_k^*$, this is not the case as $l(n-1)-1 \geq Z$. This implies that all second summation terms in equations (3.3) through (3.6) are empty. The expressions for $\boldsymbol{g}_{n,i}^{(k)}$ for all cases of i are now examined to determine where the restrictions specified above are further applicable. For the first case of $i \in \{0, 1, 2, \ldots, \min\{\min\{c_1, Z+c_2\}, l(n)\}-1\}$, the first term that would for all other cases have an upper bound of $Z-1$ becomes $l(n-1)-1$. For the second range of $i \in \{\min\{c_1, Z+c_2\}, \min\{c_1, Z+c_2\}+1, \ldots, \min\{\max\{c_1, Z+c_2\}, l(n)\}-1\}$, there are two cases to consider. If $\max\{c_1, Z+c_2\} = Z+c_2$, then the above restriction must again be incorporated in the first sum, so its upper bound becomes $l(n-1)-1$. However, when $\max\{c_1, Z+c_2\} = c_1$, the threshold level on surplus will be surpassed at or before the first claim at any time $k \in \{1, 2, 3, \ldots, n_r\}$ with probability 1, and this is a contradiction to the assumption underlying Case 2. Hence, it is impossible for $\max\{c_1, Z+c_2\} = c_1$. In the third case of $i \in \{\max\{c_1, Z+c_2\}, \max\{c_1, Z+c_2\}+1, \ldots, \min\{Z+c_1, l(n)\}-1\}$, the adjusted upper bound of $l(n-1)-1$ must again be added to the first summation term. The last case of $i \in \{Z+c_1, Z+c_1+1, \ldots, l(n)-1\}$ is dropped since $\max\{Z+c_1-1, l(n)-1\} = Z+c_1-1$ for $n = 1, 2, \ldots, t_k^*$.

Hence, the general recursive formula which incorporates the above simplifying considerations becomes the following:

Let $i \in \{0, 1, 2, \ldots, c_1-1\}$. Then:

$$\boldsymbol{g}_{n,i}^{(k)} = \left(\sum_{j=0}^{u+c_1(k+n-1)-1} \alpha_{j-i+c_1} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s}, 0, 0, \ldots, 0 \right).$$

Let $i \in \{c_1, c_1+1, c_1+2, \ldots, u+c_1(k+n)-1\}$. Then:

$$\boldsymbol{g}_{n,i}^{(k)} = \boldsymbol{g}_{n-1,i-c_1}^{(k)} S + \left(\sum_{j=i-c_1+1}^{u+c_1(k+n-1)-1} \alpha_{j-i+c_1} \boldsymbol{g}_{n-1,j}^{(k)} \boldsymbol{s}, 0, 0, \ldots, 0 \right).$$

In the analysis of $n = t_k^*+1, t_k^*+2, t_k^*+3, \ldots$, $\boldsymbol{g}_{t_k^*+1}^{(k)}$ is first looked at in isolation. At time $k+t_k^*+1$, the threshold level has been crossed, giving rise to a possible maximum surplus value of $Z-1+c_1$. As a result, $\boldsymbol{g}^{(k)}_{t^*_k+1}$ contains zeros from level $\max\{u+c_1(k+t^*_k)+c_2, Z+c_1\}$ onwards. Thus, $\boldsymbol{g}^{(k)}_{t^*_k+1}$ can be written succinctly in the form

$$\boldsymbol{g}^{(k)}_{t^*_k+1} = (\boldsymbol{g}^{(k)}_{t^*_k+1,0}, \boldsymbol{g}^{(k)}_{t^*_k+1,1}, \boldsymbol{g}^{(k)}_{t^*_k+1,2}, \ldots, \boldsymbol{g}^{(k)}_{t^*_k+1,\max\{f_{k+t^*_k+1}(k+t^*_k),Z+c_1\}-2}, \boldsymbol{g}^{(k)}_{t^*_k+1,\max\{f_{k+t^*_k+1}(k+t^*_k),Z+c_1\}-1}, \mathbf{0}, \mathbf{0}, \ldots).$$

Hence, $l(t^*_k+1) = \max\{f_{k+t^*_k+1}(k+t^*_k), Z+c_1\}$. Now, since at any point after $n = t^*_k$ the surplus could fall below level Z, it is readily established that $\boldsymbol{g}^{(k)}_n$ contains zeros from level $\max\{f_{k+n}(k+t^*_k), Z+c_1+c_2(n-t^*_k-1)\}$ onwards for $n = t^*_k+1, t^*_k+2, t^*_k+3, \ldots$, so that

$$\boldsymbol{g}^{(k)}_n = (\boldsymbol{g}^{(k)}_{n,0}, \boldsymbol{g}^{(k)}_{n,1}, \boldsymbol{g}^{(k)}_{n,2}, \ldots, \boldsymbol{g}^{(k)}_{n,\max\{f_{k+n}(k+t^*_k),Z+c_1+c_2(n-t^*_k-1)\}-2}, \boldsymbol{g}^{(k)}_{n,\max\{f_{k+n}(k+t^*_k),Z+c_1+c_2(n-t^*_k-1)\}-1}, \mathbf{0}, \mathbf{0}, \ldots).$$

Thus, for $n > t^*_k$, $l(n) = \max\{f_{k+n}(k+t^*_k), Z+c_1+c_2(n-t^*_k-1)\}$. Hence, in the second (and final) case considered, $\boldsymbol{g}^{(k)}_{n,i}$ for all i (and, consequently, $\boldsymbol{g}^{(k)}_n$) can be derived by applying the general recursion (i.e. equations (3.3) through (3.6)) where

$$l(n) = \begin{cases} u+c_1(k+n) & \text{if } n = 0,1,2,\ldots,t^*_k, \\ \max\{u+c_1(k+t^*_k)+c_2(n-t^*_k), Z+c_1+c_2(n-t^*_k-1)\} & \text{if } n = t^*_k+1, t^*_k+2, t^*_k+3, \ldots. \end{cases} \tag{3.2.2}$$

starting with $\boldsymbol{g}^{(k)}_{0,j} = \alpha_{u+c_1k-j}\boldsymbol{e}_1$, $j = 0,1,2,\ldots,u+c_1k-1$.

For $n = 1,2,\ldots,t^*_k$, it is observed that since $\boldsymbol{g}^{(k)}_{n,u+c_1k+\ell} = \mathbf{0}\ \forall\ \ell = c_1n, c_1n+1, c_1n+2, \ldots$, it follows from (2.4), (2.6), and (2.10) that

$$\boldsymbol{h}^{(k)}_{n,-j} = \sum_{\ell=c_1}^{u+c_1(k+n)-1} \boldsymbol{g}^{(k)}_{n-1,\ell-c_1} B_{j+\ell} = \left(\sum_{\ell=c_1}^{u+c_1(k+n)-1} \alpha_{j+\ell}\boldsymbol{g}^{(k)}_{n-1,\ell-c_1}\boldsymbol{s}, 0, 0, \ldots, 0 \right). \tag{3.2.3}$$

Hence, substituting (3.2.3) into (2.11) immediately yields

$$\phi^{(k)}_{n,j}(u) = \sum_{\ell=c_1}^{u+c_1(k+n)-1} \alpha_{j+\ell}\boldsymbol{g}^{(k)}_{n-1,\ell-c_1}\boldsymbol{s}. \tag{3.2.4}$$

For $n = t^*_k+1, t^*_k+2, t^*_k+3, \ldots$, $\boldsymbol{g}^{(k)}_{n,\max\{u+c_1(k+t^*_k)-c_2t^*_k, Z+c_1-c_2(1+t^*_k)\}+\ell} = \mathbf{0}\ \forall\ \ell = c_2n, c_2n+1, c_2n+2, \ldots$, and so it follows from (2.4), (2.6), and (2.10) that

$$\begin{aligned} \boldsymbol{h}^{(k)}_{n,-j} &= \sum_{\ell=c_1}^{Z+c_1-1} \boldsymbol{g}^{(k)}_{n-1,\ell-c_1} B_{j+\ell} + \sum_{\ell=Z+c_2}^{\max\{f_{k+n}(k+t^*_k),Z+c_1+c_2(n-1-t^*_k)\}-1} \boldsymbol{g}^{(k)}_{n-1,\ell-c_2} B_{j+\ell} \\ &= \left(\sum_{\ell=c_1}^{Z+c_1-1} \alpha_{j+\ell}\boldsymbol{g}^{(k)}_{n-1,\ell-c_1}\boldsymbol{s} + \sum_{\ell=Z+c_2}^{\max\{f_{k+n}(k+t^*_k),Z+c_1+c_2(n-1-t^*_k)\}-1} \alpha_{j+\ell}\boldsymbol{g}^{(k)}_{n-1,\ell-c_2}\boldsymbol{s}, 0, 0, \ldots, 0 \right). \end{aligned} \tag{3.2.5}$$

Hence, substituting (3.2.5) into (2.11) immediately yields

$$\phi_{n,j}^{(k)}(u) = \sum_{\ell=c_1}^{Z+c_1-1} \alpha_{j+\ell} \boldsymbol{g}_{n-1,\ell-c_1}^{(k)} \boldsymbol{s} + \sum_{\ell=Z+c_2}^{\max\{f_{k+n}(k+t_k^*), Z+c_1+c_2(n-1-t_k^*)\}-1} \alpha_{j+\ell} \boldsymbol{g}_{n-1,\ell-c_2}^{(k)} \boldsymbol{s}. \quad (3.2.6)$$

Finally, (3.2.4) and (3.2.6) can be substituted into (2.17) to obtain a form for $\phi_{n,j}(u)$.

4 Numerical Example

In this section, the application of the proposed algorithm is illustrated with a numerical example. The example chosen to apply the computational algorithm developed in this thesis is intended to demonstrate that for the ordinary model (i.e. $r_k = a_k$ for $k = 1, 2, 3, \ldots$), the higher the premium rate the lower the probability of ruin, and similarly, the higher the threshold level the lower the probability of ruin. Specifically, the ordinary interclaim times have pmf

$$a_j = \begin{cases} (0.075)(0.925)^{j-1} & \text{if } j = 1, 2, 3, \ldots, n_a - 1, \\ (0.925)^{n_a-1} & \text{if } j = n_a. \end{cases} \tag{4.1}$$

In other words, the pmf (4.1) is that of a truncated geometric distribution with all the probability mass on $\{n_a, n_a + 1, n_a + 2, \ldots\}$ assigned to the support value n_a. Clearly, $\sum_{j=1}^{n_a} a_j = 1$. Moreover, as n_a becomes larger, the closer $\{a_j\}_{j=1}^{n_a}$ approximates this particular geometric distribution having mean $40/3 \simeq 13.333$. For this example, $n_a = 100$ so that $a_{n_a} = 0.00044$. Let the individual claim amount distribution be given by the pmf

$$\alpha_j = G(j-1) - G(j), \; j \in \mathbb{Z}^+, \tag{4.2}$$

where $G(x) = (1 + x/30)^{-4}$, $x \geq 0$, is the survival function of a Pareto distribution with mean 10. Note that $m_\alpha = \infty$, which implies that $|U_T|$ is distributed on $\mathbb{Z}^+$ and $U_{T-} \in \{\min\{c_1, Z + c_2\}, \min\{c_1, Z + c_2\} + 1, \min\{c_1, Z + c_2\} + 2, \ldots\}$.

Tables 1 to 4 display the values (rounded to 5 significant figures) of

$$\Psi_{n,x,y}(u) = Pr\left\{T < n, U_{T-} \leq x, |U_T| \leq y \mid U_0 = u\right\} = \sum_{\ell=1}^{n-1} \sum_{i=\min\{c_1, Z+c_2\}}^{x} \sum_{j=1}^{y} \psi_{\ell,i,j}(u) \tag{4.3}$$

for a discrete-time risk process with $u = 50$, and varying combinations of c_1, c_2, and Z, having the above interclaim time and claim amount distributions. The values in Tables 1 to 4 were generated by first implementing the general recursive procedure drawn out in Section 3 using *Microsoft Visual C++* (Version 6.0), and then summing the trivariate probabilities computed via (2.13), (2.14), and (2.18). The four tables containing the values of $\Psi_{n,x,y}(50)$ correspond numerically to the following four different scenarios of premium rate combinations:

(1) $c_1 = 2$ and $c_2 = 1$;

(2) $c_1 = 3$ and $c_2 = 1$;

(3) $c_1 = 3$ and $c_2 = 2$;

(4) $c_1 = 4$ and $c_2 = 2$.

Within each table, the key is as follows:

(1) threshold level $Z = 20$;

(2) threshold level $Z = 40$;

(3) threshold level $Z = 60$;

(4) threshold level $Z = 80$.

The following observations are made concerning the results in Tables 1 to 4:

(a) When $c_1 = 2$ and $c_2 = 1$ (i.e. Scenario (1)), for every combination of x, y, n, and Z, the value of $\Psi_{n,x,y}(50)$ is greater than that resulting from Scenario (2) where $c_1 = 3$ and $c_2 = 1$. This is to be expected as the probability of ruin increases as the value of the premium below the threshold level decreases. This observation is also true when comparing ruin probabilities for Scenario (3) where $c_1 = 3$ and $c_2 = 2$ and Scenario (4) where $c_1 = 4$ and $c_2 = 2$.

(b) Under all scenarios, for set values of n, x, and y, the value of $\Psi_{n,x,y}(50)$ decreases as the size of the threshold level increases. The reason for this is that, in each scenario, c_2 is chosen to be less than c_1. Hence, the higher the threshold value, the longer that a higher premium is being received, and subsequently, the lower the probability of ruin.

(c) Note that the percentage decrease in the probability of ruin when going from, say threshold level $Z = 20$ to $Z = 40$, is always greater for set values of x, y, and n, in Scenario (2) than in Scenario (1) as the difference between c_1 and c_2 is greater. This is true of the percentage decrease in the ruin probability when comparing any threshold level to a higher one. The same holds true when comparing Scenario (3) to Scenario (4). Also, when analyzing Scenario (2) and Scenario (3), where the value of c_2 changes as opposed to c_1, the same changes are observed. That is, for identical values of n, x, and y, the greater the difference between c_1 and c_2, the greater the decrease in the probability of ruin as the threshold level is increased.

Table 1 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (1)

(a) $x = 10$

		$n = 50$	$n = 100$	$n = 250$
$y = 10$	(1)	0.0026025	0.0045832	0.0075309
	(2)	0.0021084	0.0034356	0.0053669
	(3)	0.001468	0.0022639	0.0034135
	(4)	0.0011555	0.0016296	0.0022992
$y = 25$	(1)	0.0036767	0.0064757	0.010641
	(2)	0.0029782	0.0048534	0.007582
	(3)	0.0020735	0.0031979	0.0048219
	(4)	0.0016322	0.0023019	0.0032476
$y = 50$	(1)	0.0040674	0.0071643	0.011773
	(2)	0.0032944	0.0053691	0.0083877
	(3)	0.0022936	0.0035375	0.005334
	(4)	0.0018054	0.002564	0.0035925

Table 1 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (1)

(b) $x = 25$

		$n = 50$	$n = 100$	$n = 250$
$y = 10$	(1)	0.0097188	0.017569	0.029132
	(2)	0.0063468	0.01045	0.016347
	(3)	0.0043531	0.0067598	0.010182
	(4)	0.0034271	0.0048448	0.0068063
$y = 25$	(1)	0.014632	0.026475	0.043915
	(2)	0.0094552	0.015574	0.024363
	(3)	0.006481	0.010067	0.015162
	(4)	0.0051023	0.0072136	0.010132
$y = 50$	(1)	0.016822	0.030453	0.050522
	(2)	0.010799	0.017791	0.027831
	(3)	0.0073991	0.011494	0.017312
	(4)	0.0058251	0.0082357	0.011567

Table 1 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (1)

(c) $x = 50$

		$n = 50$	$n = 100$	$n = 250$
$y = 10$	(1)	0.020066	0.036394	0.060197
	(2)	0.014524	0.023815	0.036866
	(3)	0.0082123	0.012639	0.018808
	(4)	0.006431	0.0089691	0.012393
$y = 25$	(1)	0.032016	0.058061	0.096001
	(2)	0.023354	0.038283	0.059217
	(3)	0.01299	0.019976	0.029695
	(4)	0.010168	0.014164	0.019542
$y = 50$	(1)	0.038396	0.069611	0.11506
	(2)	0.02821	0.046231	0.071472
	(3)	0.015503	0.023826	0.03539
	(4)	0.012131	0.016883	0.023268

Table 2 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (2)

(a) $x = 10$

		$n = 50$	$n = 100$	$n = 250$
$y = 10$	(1)	0.001427	0.0024736	0.0040444
	(2)	0.0010474	0.0016559	0.0025506
	(3)	0.00060651	0.00091592	0.0013801
	(4)	0.00040534	0.0005642	0.00080792
$y = 25$	(1)	0.0020215	0.0035044	0.0057298
	(2)	0.0014835	0.0023454	0.0036126
	(3)	0.00085893	0.0012971	0.0019545
	(4)	0.00057403	0.00079899	0.0011441
$y = 50$	(1)	0.0022394	0.0038822	0.0063477
	(2)	0.0016433	0.002598	0.0040017
	(3)	0.00095136	0.0014367	0.0021648
	(4)	0.00063579	0.00088495	0.0012672

Table 2 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (2)

(b) $x = 25$

		$n = 50$	$n = 100$	$n = 250$
$y = 10$	(1)	0.0072794	0.012946	0.021346
	(2)	0.0035266	0.0055993	0.0086089
	(3)	0.0019846	0.0029998	0.0045016
	(4)	0.001318	0.0018268	0.0025962
$y = 25$	(1)	0.011084	0.019725	0.032533
	(2)	0.0052756	0.0083773	0.012879
	(3)	0.0029658	0.0044829	0.0067262
	(4)	0.0019691	0.0027289	0.003877
$y = 50$	(1)	0.012829	0.022842	0.037679
	(2)	0.0060393	0.0095907	0.014744
	(3)	0.0033929	0.0051286	0.0076943
	(4)	0.0022524	0.0031212	0.0044336

Table 2 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (2)

(c) $x = 50$

		$n = 50$	$n = 100$	$n = 250$
$y = 10$	(1)	0.018038	0.032474	0.053591
	(2)	0.010708	0.016924	0.025652
	(3)	0.0040137	0.0059966	0.0088687
	(4)	0.002624	0.0035677	0.0049648
$y = 25$	(1)	0.029145	0.052476	0.086581
	(2)	0.017556	0.027742	0.042016
	(3)	0.0063912	0.0095397	0.014092
	(4)	0.0041731	0.005665	0.0078693
$y = 50$	(1)	0.035232	0.06343	0.10463
	(2)	0.021499	0.033967	0.051415
	(3)	0.0076607	0.011427	0.016865
	(4)	0.0049975	0.0067764	0.0094013

Table 3 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (3)

(a) $x = 10$

		$n = 50$	$n = 100$	$n = 150$
$y = 10$	(1)	0.00058146	0.00069206	0.00072053
	(2)	0.00052314	0.00061361	0.00063714
	(3)	0.00040973	0.00047638	0.00049425
	(4)	0.00034644	0.0003944	0.00040765
$y = 25$	(1)	0.00082356	0.00098023	0.0010205
	(2)	0.00074092	0.00086905	0.00090237
	(3)	0.00058027	0.00067465	0.00069996
	(4)	0.00049063	0.00055855	0.00057732
$y = 50$	(1)	0.00091224	0.0010858	0.0011304
	(2)	0.00082068	0.0009626	0.00099951
	(3)	0.00064271	0.00074726	0.00077529
	(4)	0.00054343	0.00061865	0.00063944

Table 3 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (3)

(b) $x = 25$

		$n = 50$	$n = 100$	$n = 150$
$y = 10$	(1)	0.0021687	0.0025867	0.0026911
	(2)	0.0017347	0.0020343	0.0021101
	(3)	0.001343	0.0015598	0.0016164
	(4)	0.0011342	0.0012884	0.0013298
$y = 25$	(1)	0.0032612	0.00389	0.0040468
	(2)	0.0025936	0.0030413	0.0031545
	(3)	0.002007	0.0023309	0.0024153
	(4)	0.0016949	0.0019252	0.001987
$y = 50$	(1)	0.0037455	0.0044679	0.0046479
	(2)	0.002968	0.0034803	0.0036098
	(3)	0.0022961	0.0026666	0.0027631
	(4)	0.001939	0.0022023	0.002273

Table 3 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (3)

(c) $x = 50$

		$n = 50$	$n = 100$	$n = 150$
$y = 10$	(1)	0.0051004	0.006061	0.00629
	(2)	0.0039945	0.0046369	0.0047917
	(3)	0.0027074	0.0031148	0.0032169
	(4)	0.002282	0.0025652	0.0026384
$y = 25$	(1)	0.0082025	0.009744	0.01011
	(2)	0.0064303	0.0074589	0.0077058
	(3)	0.0043101	0.0049548	0.0051159
	(4)	0.0036323	0.0040797	0.0041948
$y = 50$	(1)	0.0098941	0.01175	0.01219
	(2)	0.0077703	0.0090085	0.0093049
	(3)	0.0051653	0.0059346	0.0061263
	(4)	0.0043525	0.0048857	0.0050225

Table 4 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (4)

(a) $x = 10$

		$n = 50$	$n = 100$	$n = 150$
$y = 10$	(1)	0.00035685	0.00017674	0.00044001
	(2)	0.00030415	0.00035392	0.00036703
	(3)	0.00020934	0.00024223	0.00025132
	(4)	0.00015574	0.0004229	0.00018289
$y = 25$	(1)	0.00050675	0.00060054	0.00062483
	(2)	0.00043188	0.00050254	0.00052115
	(3)	0.00029724	0.00034392	0.00035683
	(4)	0.00022113	0.00025093	0.00025966
$y = 50$	(1)	0.00056205	0.00066608	0.00069302
	(2)	0.00047899	0.00055735	0.00057799
	(3)	0.00032964	0.00038142	0.00039573
	(4)	0.00024523	0.00027829	0.00028796

Table 4 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (4)

(b) $x = 25$

		$n = 50$	$n = 100$	$n = 150$
$y = 10$	(1)	0.0016479	0.0019536	0.0020301
	(2)	0.0011105	0.0012891	0.0013348
	(3)	0.00074915	0.00086406	0.00089501
	(4)	0.00055477	0.00062682	0.00064739
$y = 25$	(1)	0.0024981	0.0029615	0.0030775
	(2)	0.0016656	0.0019332	0.0020017
	(3)	0.0011228	0.0012949	0.0013412
	(4)	0.00083135	0.00093916	0.00096993
$y = 50$	(1)	0.0028833	0.0034182	0.003552
	(2)	0.0019094	0.002216	0.0022945
	(3)	0.0012866	0.0014837	0.0015367
	(4)	0.00095255	0.001076	0.0011112

Table 4 – Values of $\Psi_{n,x,y}(50)$ corresponding to Scenario (4)

(c) $x = 50$

		$n = 50$	$n = 100$	$n = 150$
$y = 10$	(1)	0.0046033	0.0054422	0.0056415
	(2)	0.0030205	0.0034586	0.0035646
	(3)	0.0015689	0.0017894	0.0018466
	(4)	0.0011531	0.0012866	0.0013233
$y = 25$	(1)	0.0074799	0.0088405	0.0091628
	(2)	0.0049229	0.0056322	0.0058031
	(3)	0.002507	0.0028569	0.0029472
	(4)	0.0018416	0.0020527	0.0021106
$y = 50$	(1)	0.0090825	0.010732	0.011122
	(2)	0.0060012	0.0068618	0.0070686
	(3)	0.0030114	0.0034297	0.0035374
	(4)	0.0022112	0.002463	0.002532

5 References

Alfa, A. S. and Drekic, S. (2007). "Algorithmic analysis of the Sparre Andersen model in discrete time." *ASTIN Bulletin* **37**, in press.

Alfa, A. S. (2004). "Markov chain representations of discrete distributions applied to queueing models." *Computers & Operations Research* **31**, 2365-2385.

Grassmann, W. K. (2000). *Computational Probability*, Kluwer Academic Publishers, Boston.

Karlin, S. and Taylor, H. M. (1975). *A First Course in Stochastic Processes*, 2nd edition, Academic Press, New York.

6 Appendix

The following is the code used to implement the computational algorithm. It outputs the results in the tables of the numerical example.

```
// ThesisResults.cpp : Defines the entry point for the console
 application.
//
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
 return 0;
}

#include <iostream>
#include <math.h>
#include <ctime>
#include <string>
using namespace std;

//instantiate global variables that are unchanging through the program
int u;
int w;
int Z;
int c1;
int c2;
int minBound;
int maxBound;
//define all vectors that are used later
double * e1;
double  * s;
double ** S;

double * gnMaxBound(int k, int n, int j);
double * gnZ(int k, int n, int j);
```

```
double * gFinal(int k, int n, int i);
//ALWAYS CHANGE PARAMETERS
//[n-2][50+c1(96+n)][w]
//set 3-d matrix parameters initially to save memory
double * storage[248][742][100];
double * last;
//file to write to as go through program
FILE* file;

//calculates probability of certain waiting time
double amtwt(int i)
{
 if(i<w)
 {
  return .075*pow(.925,(i-1));
 }
 return pow(.925,w-1);

}

//function that takes minimum of 2 values
double min (int a,int b)
{
 if (a<b)
  return a;
 return b;
}
//function that takes maximum of 2 values
double max (int a,int b)
{
 if (a > b)
  return a;
 return b;
}
//initializing all pointer arrays and counters
```

```
void initialize(int n)
{
 minBound = min(c1, Z+c2);
 maxBound = max(c1, Z+c2);
 S = new double * [w];    // dynamic allocation of pointer array for
 rows

 for(int y=0;y<w;y++)
  S[y]=new double [w];     // dynamic allocation of columns for each
 row

 //initializing matrix S
 for(int i=0;i<w;i++)
 {
  for(int q=0;q<w;q++)
  {
   if(q==i+1)
    S[i][q]=.925;
   else
    S[i][q]=0;
  }
 }

 //initializing default entry in storage matrix
 last = new double [w];
 for ( int a=0; a<w;a++)
 {
  last[a]=5;
 }

 //initializing e1
 e1 = new double [w];
 e1[0]=1;
 for ( int h=1; h<w; h++)
 {
```

```
  e1[h]=0;
 }
 //initializing s
 s = new double [w];
 for(int d=0; d<(w-1); d++)
 {
  s[d]=.075;
 }
 s[w-1]=1;


 //inputting last as default entry in storage matrix
 for(int l=0;l < (n-2);l++)
 {
  for(int p=0;p < u+c1*(w+n-4);p++)
  {
   for(int t=0;t < min(n-2,w); t++)
   {
    storage[l][p][t]=last;
   }
  }
 }
}

//function that writes (final) elements of storage matrix to a file
void writeToFile(int n)
{
 //FILE* f = fopen(filename.c_str(), "wb");

  int start=0;
  if (n > (w+3))
  start=n-(w+3);

  for(int l=start;l<(n-2);l++)
  {
```

```
   for(int m=0;m < u+c1*(w+n-4);m++)
    {
      for(int a=0;a<min(n-2,w); a++)
      {
    int wasEmpty;
    if(storage[l][m][a][0] == 5)
    {
     wasEmpty = 1;
     fwrite(&wasEmpty, 1, sizeof(int), file);
    }
    else
    {
     wasEmpty = 0;
     fwrite(&wasEmpty, 1, sizeof(int), file);
     fwrite(storage[l][m][a], 100, sizeof(double), file);
     //cout << "Wrote non-empty stuff at " << l << " " << m << " " << a
 << endl;
    }
      }
    }
  }
 //fclose(f);
}
//function reads and inputs elements into storage matrix from file
 written to
void readFromFile(string filename)
{
  FILE* f = fopen(filename.c_str(), "rb");

  int u,n,w;

  fread(&u, 1, sizeof(int), f);
  fread(&n, 1, sizeof(int), f);
  fread(&w, 1, sizeof(int), f);
```

```
  //cout << "Reading " << filename << " with u=" << u << ", n=" << n <<
 ", w=" << k << endl;
  int start=0;
  /*if (n >= (w+3))
  start=n-w-3;*/

  for(int l=start;l< (n-2);l++)
  {
   for(int m=0;m<u+c1*(w+n-4);m++)
    {
      for(int a=0;a<min(n-2,w); a++)
      {
    int wasEmpty;
    fread(&wasEmpty, 1, sizeof(int), f);
    if(wasEmpty == 0)
    {
     //cout << "Found non-empty stuff at " << l << " " << m << " " << a
 << endl;
     storage[l][m][a] = new double[100];
     fread(storage[l][m][a], 100, sizeof(double), f);
          }
     }
   }
 }
 fclose(f);
}
//function multiplies a vector by a scalar and returns a "new"
 resulting vector
double * vecScaMult(double vec[],double scalar)
{
 double * res=new double[w];
 for(int i=0;i<w;i++)
 {
  res[i]=vec[i]*scalar;
 }
```

```
 return res;
}
//multiplies a vector by a matrix (using shortcut) and returns a "new"
 vector
double * vecMatMult(double vec[],double * mat[])
{
  double * res= new double [w];
  res[0] = 0;
  for(int i=1; i<w;i++)
  {
  res[i]=vec[i-1]*mat[i-1][i];
  }
  return res;
}
//dot product of 2 vectors
double  vecMult(double vec1[],double vec2[])
{ double prob =0;
 for (int i=0;i<w;i++)
 {
  prob+=vec1[i]*vec2[i];
 }
 return prob;
}
//this function does operations on vectors (adding the first to the
 second,
//which is returned) instead of creating new one
void vecAdd(double vec1[],double vec2[])
{
 for (int i=0;i<w;i++)
 {
  vec2[i]=vec1[i]+vec2[i];
 }
}
//two vectors are added together, and returned as a "new" vector
double * vecAdd2(double vec1[],double vec2[])
```

```
{
 double * add = new double [w];
 for (int i=0; i< w; i++)
 {
  add[i] = vec1[i]+vec2[i];
 }
 return add;
}
//survival function of a Pareto distribution having mean 10
double pa(double j)
{
 return pow((double)(1.0+j/30.0),-4.0);
}
//claim amount pdf
double amtcl(double i)
{
 if(i>0)
 {
  return (pa(i-1)-pa(i));
 }
 return 0;
}
//ensure this is pos (another version of claim amount pdf)
/*double amtcl(double i)
{
 double check=0;
 if(i>0)
 {
  check = (pa(i-1)-pa(i));
 }
 if (check>=0 && check <=1)
  return check;
 return 0;
}*/
//Kronecker delta function
```

```
double delta(int a,int b)
{
 if(a==b)
 {
  return 1;
 }
 return 0;
}
//obtains x_t required for l(n)
int getX (int t)
{
 if (u >= Z) return 0;
 for (int i=t; i > 0;i--)
 {
  if (u+c1*(i-1) < Z)
   return i;
 }
 return -1;
}

//obtains highest "i" value in g(n,k,i) for which the latter is not
 zero
int limit (int k,int n)
{
 int xk = getX(k);
 int limitg1 = u+c1*xk+c2*(k-xk);
 if ( limitg1 > Z)
 {
  if (n > 0)
   return max(limitg1+c2*n,Z+c1+c2*(n-1)) ;
  return limitg1;
 }

 int tkStar = (int) ceil((double)((Z-(u+c1*k)+1)/c1));
 if (n <= tkStar)
```

```
 {
  return u+c1*(k+n);
 }
 return max(u+c1*(k+tkStar)+c2*(n-tkStar),Z+c1+c2*(n-1-tkStar));
}
//obtains g(n=0,k,i)
double * g0(int k,int j)
{
 //don't use "new" to free up storage space
 int xk = getX(k);
 double * prob;
 prob=vecScaMult(e1,amtcl(u+c1*xk+c2*(k-xk)-j));
 storage[0][j][k-1]=prob;
 return prob;
}
//function for i=0,1,....,min(minBound,l(n))-1, returns g(n>0,k,i)
double *gnMinBound(int k,int n,int i)
{
 //resulting g(n,k,i) vector that takes up memory space and is place in
 storage
 double * ana=new double [w];
 for(int b=0;b<w;b++)
 {
  ana[b]=0;
 }
 int limKNless1 = limit(k,n-1);
 //int limKN = limit(k,n);
 //points to g(n-1,k,j) term to be used in first sum, changes with each
 iteration
 double * bob;
 for(int j=0; j < min(Z,limKNless1);j++)
 {
  if(storage[n-1][j][k-1][0]!=5)
  {
   bob=storage[n-1][j][k-1];
```

```
 }
 else if(n==1)
 {
  bob=g0(k,j);
 }
 else if (j < min(minBound,limKNless1))
 {
  bob=gnMinBound(k,n-1,j);
 }
 else if (j< min(maxBound,limKNless1))
 {
  bob=gnMaxBound(k,n-1,j);
 }
 else if (j< min(Z+c1,limKNless1))
 {
  bob=gnZ(k,n-1,j);
 }
 else
 {
  bob=gFinal(k,n-1,j);
 }
 //vecMult does not create new storage space
 ana[0]+=amtcl(j-i+c1)*vecMult(bob,s);
}
//points to g(n-1,k,j) term to be used in second sum, changes with
each iteration
double * dad;
for(int j=Z; j< limKNless1;j++)
{
 if(storage[n-1][j][k-1][0]!=5)
 {
  dad=storage[n-1][j][k-1];
 }
 else if(n==1)
```

```
   {
    dad=g0(k,j);
   }
   else if(j < min(minBound,limKNless1))
   {
    dad=gnMinBound(k,n-1,j);
   }
   else if (j < min(maxBound,limKNless1))
   {
    dad=gnMaxBound(k,n-1,j);
   }
   else if (j < min(Z+c1,limKNless1))
   {
    dad=gnZ(k,n-1,j);
   }
   else
   {
    dad=gFinal(k,n-1,j);
   }
   //added to first entry in vector that holds this g(n,k,i)
   ana[0]+=amtcl(j-i+c2)*vecMult(dad,s);
  }
  //store corresp to i, which now varies
  storage[n][i][k-1]=ana;

  return ana;
 }
 //returns and stores value of g(k,n,i) for
  i=minBound,....min(maxBound,l(n))-1
 double * gnMaxBound(int k,int n,int i)
 {
  int limKNless1 = limit(k,n-1);
  //int limKNless1 = limit(k,n);

  //code for case with maximum bound = Z+c2
```

```
if (maxBound == (Z+c2))
{
 //points to component required for first term (that is,
g(n-1,k,i-c1))
 double * first;
 if(storage[n-1][i-c1][k-1][0]!=5)
 {
  first=storage[n-1][i-c1][k-1];
 }
 else if(n==1)
 {
  first=g0(k,i-c1);
 }
 else if (i < min(minBound,limKNless1)+c1)
 {
  first=gnMinBound(k,n-1,i-c1);
 }
 else if (i < min(maxBound,limKNless1)+c1)
 {
  first=gnMaxBound(k,n-1,i-c1);
 }
 else if (i < min(Z+c1,limKNless1)+c1)
 {
  first = gnZ(k,n-1,i-c1);
 }
 else
 {
  first = gFinal(k,n-1,i-c1);
 }

 //points to first term, that is, g(n-1,k,i-c1)*S
 double * sec;
 sec=vecMatMult(first,S);

 //creates "new" memory space for this g(n,k,i) in storage matrix,
```

```
which cannot be deleted
 double * ana = new double[w];
 for(int b=0;b<w;b++)
 {
  ana[b]=0;
 }

 //points to g(k,n-1,j) for each iteration in first sum
 double * bob;
 for(int j=(i-c1+1);j < min(Z,limKNless1);j++)
 {
  if(storage[n-1][j][k-1][0]!=5)
  {
   bob=storage[n-1][j][k-1];
  }
  else if(n==1)
  {
   bob=g0(k,j);
  }
  else if(j < min(minBound,limKNless1))
  {
   bob=gnMinBound(k,n-1,j);
  }
  else if (j < min(maxBound,limKNless1))
  {
   bob=gnMaxBound(k,n-1,j);
  }
  else if (j < min(Z+c1,limKNless1))
  {
   bob=gnZ(k,n-1,j);
  }
  else
  {
   bob=gFinal(k,n-1,j);
  }
```

```
  //obtains contribution of first sum to first entry in the resulting
vector
  ana[0]+=amtcl(j-i+c1)*vecMult(bob,s);
 }

 //.points to g(k,n-1,i) for each iteration in second sum
 double * dad;
 for(int j=Z; j< limKNless1;j++)
 {
  if(storage[n-1][j][k-1][0]!=5)
  {
   dad=storage[n-1][j][k-1];
  }
  else if(n==1)
  {
   dad=g0(k,j);
  }
  else if(j < min(minBound,limKNless1))
  {
   dad=gnMinBound(k,n-1,j);
  }
  else if (j < min(maxBound,limKNless1))
  {
   dad=gnMaxBound(k,n-1,j);
  }
  else if (j < min(Z+c1,limKNless1))
  {
   dad=gnZ(k,n-1,j);
  }
  else
  {
   dad=gFinal(k,n-1,j);
  }
```

```
  //obtains contribution of second sum to first entry in resulting
vector
  ana[0]+=amtcl(j-i+c2)*vecMult(dad,s);
 }

 vecAdd(ana,sec);  //modify sec which points to new space
 //"ana" can be deleted as "sec" is required for storage
 delete[] ana;
 storage[n][i][k-1]=sec;
 return sec;
}
 //code for case with maximum bound = c1

 //points to g(n-1,k,i-c2) in first term
 double * first;
 if(storage[n-1][i-c2][k-1][0]!=5)
 {
  first=storage[n-1][i-c2][k-1];
 }
 else if(n==1)
 {
  first=g0(k,i-c2);
 }
 else if (i < min(minBound,limKNless1)+c2)
 {
  first=gnMinBound(k,n-1,i-c2);
 }
 else if (i < min(maxBound,limKNless1)+c2)
 {
  first=gnMaxBound(k,n-1,i-c2);
 }
 else if (i < min(Z+c1,limKNless1)+c2)
 {
  first = gnZ(k,n-1,i-c2);
```

```
}
else
{
 first = gFinal(k,n-1,i-c2);
}

//points to first term, that is g(k,n-1,i-c2)*S
double * sec;
sec=vecMatMult(first,S);

//vector that holds values of second term in g(k,n,i)
double * ana = new double[w];
for(int b=0;b<w;b++)
{
 ana[b]=0;
}

//points to g(k,n-1,j) for each iteration in first sum
double * bob;
for(int j = 0;j < Z;j++)
{
 if(storage[n-1][j][k-1][0]!=5)
 {
  bob=storage[n-1][j][k-1];
 }
 else if(n==1)
 {
  bob=g0(k,j);
 }
 else if(j < min(minBound,limKNless1))
 {
  bob=gnMinBound(k,n-1,j);
 }
 else if (j < min(maxBound,limKNless1))
 {
```

```
   bob=gnMaxBound(k,n-1,j);
  }
  else if (j < min(Z+c1,limKNless1))
  {
   bob=gnZ(k,n-1,j);
  }
  else
  {
   bob=gFinal(k,n-1,j);
  }

  //adds contribution of each component in first sum to vector
corresponding to second term
  ana[0]+=amtcl(j-i+c1)*vecMult(bob,s);
 }

 //points to each g(k,n-1,j) for each iteration in second sum of
second term
 double * dad;
 for(int j = (i-c2+1); j< limKNless1;j++)
 {
  if(storage[n-1][j][k-1][0]!=5)
  {
   dad=storage[n-1][j][k-1];
  }
  else if(n==1)
  {
   dad=g0(k,j);
  }
  else if(j < min(minBound,limKNless1))
  {
   dad=gnMinBound(k,n-1,j);
  }
  else if (j < min(maxBound,limKNless1))
  {
```

```
    dad=gnMaxBound(k,n-1,j);
   }
   else if (j < min(Z+c1,limKNless1))
   {
    dad=gnZ(k,n-1,j);
   }
   else
   {
    dad=gFinal(k,n-1,j);
   }

   //each component of second sum is added to first entry of
   //vector holding second term values
   ana[0]+=amtcl(j-i+c2)*vecMult(dad,s);
  }
  vecAdd(ana,sec);  //modify sec which points to new space
  //"ana" can be deleted as it created "new" space and is no longer
 required
  delete[] ana;
  storage[n][i][k-1]=sec;
  return sec;
}
//returns and stores value of g(k,n,i) for
 i=maxBound,...min(Z+c1,l(n))-1
double * gnZ(int k,int n,int i)
{
 int limKNless1 = limit(k,n-1);
 //int limKNless1 = limit(k,n);
 //points to g(k,n-1,i-c1) in first term of g(k,n,i)
 double * first;
 if(storage[n-1][i-c1][k-1][0]!=5)
 {
  first=storage[n-1][i-c1][k-1];
 }
 else if(n==1)
```

```
{
 first=g0(k,i-c1);
}
else if (i < min(minBound,limKNless1)+c1)
{
 first=gnMinBound(k,n-1,i-c1);
}
else if (i < min(maxBound,limKNless1)+c1)
{
 first=gnMaxBound(k,n-1,i-c1);
}
else if (i < min(Z+c1,limKNless1)+c1)
{
 first = gnZ(k,n-1,i-c1);
}
else
{
 first = gFinal(k,n-1,i-c1);
}

//points to first term of g(k,n,i)
double * sec;
sec=vecMatMult(first,S);

//points to g(k,n-1,i-c1) in second term of g(k,n,i)
double * third;
if(storage[n-1][i-c2][k-1][0]!=5)
{
 third=storage[n-1][i-c2][k-1];
}
else if(n==1)
{
 third=g0(k,i-c2);
}
else if (i < min(minBound,limKNless1)+c2)
```

```
{
 third=gnMinBound(k,n-1,i-c2);
}
else if (i < min(maxBound,limKNless1)+c2)
{
 third=gnMaxBound(k,n-1,i-c2);
}
else if (i < min(Z+c1,limKNless1)+c2)
{
 third = gnZ(k,n-1,i-c2);
}
else
{
 third = gFinal(k,n-1,i-c2);
}
//points to second term of g(k,n,i)
double * fou;
fou=vecMatMult(third,S);
vecAdd(fou,sec);  //modify sec which points to new space
//'fou' obtained from 'vecMatMult' created new space, that is no
longer needed
delete[] fou;
//creates new space to hold vector that is third term of g(k,n,i)
double * ana = new double[w];
for(int b=0;b<w;b++)
{
 ana[b]=0;
}

//points to g(k,n-1,j) on each iteration of first sum in third term
double * bob;
for(int j = (i-c1+1);j < min(Z,limKNless1);j++)
{
 if(storage[n-1][j][k-1][0]!=5)
 {
```

```
  bob=storage[n-1][j][k-1];
 }
 else if(n==1)
 {
  bob=g0(k,j);
 }
 else if(j < min(minBound,limKNless1))
 {
  bob=gnMinBound(k,n-1,j);
 }
 else if (j < min(maxBound,limKNless1))
 {
  bob=gnMaxBound(k,n-1,j);
 }
 else if (j < min(Z+c1,limKNless1))
 {
  bob=gnZ(k,n-1,j);
 }
 else
 {
  bob=gFinal(k,n-1,j);
 }

 //add contribution of first sum to first entry of vector (of third
term) on each iteration
 ana[0]+=amtcl(j-i+c1)*vecMult(bob,s);
}

//points to g(k,n-1,j) on each iteration of second sum in third term
double * dad;
for(int j = (i-c2+1); j< limKNless1;j++)
{
 if(storage[n-1][j][k-1][0]!=5)
 {
  dad=storage[n-1][j][k-1];
```

```
  }
  else if(n==1)
  {
   dad=g0(k,j);
  }
  else if(j < min(minBound,limKNless1))
  {
   dad=gnMinBound(k,n-1,j);
  }
  else if (j < min(maxBound,limKNless1))
  {
   dad=gnMaxBound(k,n-1,j);
  }
  else if (j < min(Z+c1,limKNless1))
  {
   dad=gnZ(k,n-1,j);
  }
  else
  {
   dad=gFinal(k,n-1,j);
  }

  //adds contribution of each compnent in second sum of third term
  ana[0]+=amtcl(j-i+c2)*vecMult(dad,s);
 }

 vecAdd(ana,sec);  //modify sec which points to new space
 //"ana" which created "new" memory space can now be deleted
 delete[] ana;
 storage[n][i][k-1]=sec;
 return sec;

}
//returns and stores g(k,n,i) for i=Z+c1,...,l(n)-1
double * gFinal(int k, int n, int i)
```

```
{
 int limKNless1 = limit(k,n-1);
 //int limKNless1 = limit(k,n);
 //points to g(k,n-1,i-c2) of first term
 double * first;
 if(storage[n-1][i-c2][k-1][0]!=5)
 {
  first=storage[n-1][i-c2][k-1];
 }
 else if(n==1)
 {
  first=g0(k,i-c2);
 }
 else if (i < min(minBound,limKNless1)+c2)
 {
  first=gnMinBound(k,n-1,i-c2);
 }
 else if (i < min(maxBound,limKNless1)+c2)
 {
  first=gnMaxBound(k,n-1,i-c2);
 }
 else if (i < min(Z+c1,limKNless1)+c2)
 {
  first = gnZ(k,n-1,i-c2);
 }
 else
 {
  first = gFinal(k,n-1,i-c2);
 }

 //points to first term of g(k,n,i)
 double * sec;
 sec=vecMatMult(first,S);

 //creates "new" memory space for vector that is second term
```

```
double * ana = new double[w];
for(int b=0;b<w;b++)
{
 ana[b]=0;
}

//points to g(k,n,j) in each iteration of sum in second term
double * dad;
for(int j = (i-c2+1); j < limKNless1;j++)
{
 if(storage[n-1][j][k-1][0]!=5)
 {
  dad=storage[n-1][j][k-1];
 }
 else if(n==1)
 {
  dad=g0(k,j);
 }
 else if(j < min(minBound,limKNless1))
 {
  dad=gnMinBound(k,n-1,j);
 }
 else if (j < min(maxBound,limKNless1))
 {
  dad=gnMaxBound(k,n-1,j);
 }
 else if (j < min(Z+c1,limKNless1))
 {
  dad=gnZ(k,n-1,j);
 }
 else
 {
  dad=gFinal(k,n-1,j);
 }
 //adds each component of sum to first entry in vector for second term
```

```
  ana[0]+=amtcl(j-i+c2)*vecMult(dad,s);
 }
 vecAdd(ana,sec);  //modify sec which points to new space
 //"ana" must be deleted as it created "new" memory space that is no
 longer needed
 delete[] ana;
 storage[n][i][k-1]=sec;
 return sec;
}
//computes psi(k,n,k,i,j(u)) for which there are 3 possibilities
//depending on the minimum bound
double psi(int k,int n,int i,int j)
{
 int limKNless1 = limit(k,n-1);
 //int limKNless1 = limit(k,n);

 if (minBound == c1)
 {
  if (i < maxBound) //i=c1,....Z+c2-1
  {
   int l=i-c1;

   //checks storage matrix in case the g(k,n-1,l) required has already
 been computed
   if(n==1)
   {
    if(storage[0][l][k-1][0]!=5)
     return amtcl(i+j)*vecMult(storage[0][l][k-1],s);
    return amtcl(i+j)*vecMult(g0(k,l),s);
   }
   if(l < min(minBound,limKNless1))
   {
    if(storage[n-1][l][k-1][0]!=5)
     return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
    return amtcl(i+j)*vecMult(gnMinBound(k,n-1,l),s);
```

```
  }
  if (l < min(maxBound,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnMaxBound(k,n-1,l),s);
  }
  if (l < min(Z+c1,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnZ(k,n-1,l),s);
  }
  if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
  return amtcl(i+j)*vecMult(gFinal(k,n-1,l),s);
 }

 else if (i < Z+c1)  //i=Z+c2,...,Z+c1-1
 {
  int l=i-c1;
  int m=i-c2;
  double * term1;
  double * term2;

  //checks storage matrix in case the g(k,n-1,l) required has already
been computed
  if(n==1)
  {
   if(storage[n-1][l][k-1][0]!=5)
    term1 = storage[n-1][l][k-1];
   else
    term1 = g0(k,l);

   if(storage[n-1][m][k-1][0]!=5)
```

```
    term2 = storage[n-1][m][k-1];
   else
    term2 = g0(k,m);

   //'corr' term required as term1 & term2 cannot be overridden
   //since they point to element in storage matrix
   double * corr;
   corr=vecAdd2(term1,term2);
   double hold = amtcl(i+j)*vecMult(corr,s);
   //'CORR' no longer needed as 'hold' stores result;
                 delete[] corr;
   return hold;
  }
  if(l < min(minBound,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    term1 = storage[n-1][l][k-1];
   else
    term1 = gnMinBound(k,n-1,l);
  }
  else if (l < min(maxBound,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    term1 = storage[n-1][l][k-1];
   else
    term1 = gnMaxBound(k,n-1,l);
  }
  else if (l < min(Z+c1,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    term1 = storage[n-1][l][k-1];
   else
    term1 = gnZ(k,n-1,l);
   //term1 = gnZ(k,n-1,l); (previous way of obtaining term1 - less
efficient)
```

```
}
else
{
 if(storage[n-1][l][k-1][0]!=5)
  term1 = storage[n-1][l][k-1];
 else
  term1 = gFinal(k,n-1,l);
 //term1 = gFinal(k,n-1,l);
}

if(m < min(minBound,limKNless1))
{
 if(storage[n-1][m][k-1][0]!=5)
  term2 = storage[n-1][m][k-1];
 else
  term2 = gnMinBound(k,n-1,m);
}
else if (m < min(maxBound,limKNless1))
{
 if(storage[n-1][m][k-1][0]!=5)
  term2 = storage[n-1][m][k-1];
 else
  term2 = gnMaxBound(k,n-1,m);
}
else if (m < min(Z+c1,limKNless1))
{
 if(storage[n-1][m][k-1][0]!=5)
  term2 = storage[n-1][m][k-1];
 else
  term2 = gnZ(k,n-1,m);
}
else
{
 if(storage[n-1][m][k-1][0]!=5)
  term2 = storage[n-1][m][k-1];
```

```
    else
     term2 = gFinal(k,n-1,m);
   }
   double * corr;
   //'corr' required as neither term1 not term2 can be overridden
   corr=vecAdd2(term1,term2);
   //'hold' needed temporarily so 'corr', which pointed to new memory
 space can be deleted
   double hold = amtcl(i+j)*vecMult(corr,s);
   delete[] corr;
   return hold;
  }
  else   //i=Z+c1,...infinity
  {
   int l=i-c2;
   if(n==1)
   {
    if(storage[0][l][k-1][0]!=5)
     return amtcl(i+j)*vecMult(storage[0][l][k-1],s);
    return amtcl(i+j)*vecMult(g0(k,l),s);
   }
   if(l < min(minBound,limKNless1))
   {
    if(storage[n-1][l][k-1][0]!=5)
     return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
    return amtcl(i+j)*vecMult(gnMinBound(k,n-1,l),s);
   }
   if (l < min(maxBound,limKNless1))
   {
    if(storage[n-1][l][k-1][0]!=5)
     return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);

    return amtcl(i+j)*vecMult(gnMaxBound(k,n-1,l),s);
   }
   if (l < min(Z+c1,limKNless1))
```

```
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnZ(k,n-1,l),s);
  }
  if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
  return amtcl(i+j)*vecMult(gFinal(k,n-1,l),s);
 }
}
 //case of minimum bound = Z+c2
 if (i < maxBound) //i=Z+c2,...c1-1
 {

  int l=i-c2;
  //checks storage matrix in case the g(k,n-1,l) required has already
been computed
  if(n==1)
  {
   if(storage[0][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[0][l][k-1],s);
   return amtcl(i+j)*vecMult(g0(k,l),s);
  }
  if(l < min(minBound,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnMinBound(k,n-1,l),s);
  }
  if (l < min(maxBound,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnMaxBound(k,n-1,l),s);
  }
```

```
  if (l < min(Z+c1,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnZ(k,n-1,l),s);
  }
  if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
  return amtcl(i+j)*vecMult(gFinal(k,n-1,l),s);
 }

 else if (i < Z+c1)  //i=c1,...Z+c1-1
 {
  int l=i-c1;
  int m=i-c2;
  double * term1;
  double * term2;
  //checks storage matrix in case the g(k,n-1,l) required has already
been computed
  if(n==1)
  {
   if(storage[n-1][l][k-1][0]!=5)
    term1 = storage[n-1][l][k-1];
   else
    term1 = g0(k,l);

   if(storage[n-1][m][k-1][0]!=5)
    term2 = storage[n-1][m][k-1];
   else
    term2 = g0(k,m);

   double * corr;
   //'corr' points to new space created by vecAdd2
   corr=vecAdd2(term1,term2);
   double hold = amtcl(i+j)*vecMult(corr,s);
```

```
               //new space created by vecAdd2 is no longer needed
 delete[] corr;
 return hold;
}
if(l < min(minBound,limKNless1))
{
 if(storage[n-1][l][k-1][0]!=5)
  term1 = storage[n-1][l][k-1];
 else
  term1 = gnMinBound(k,n-1,l);
}
else if (l < min(maxBound,limKNless1))
{
 if(storage[n-1][l][k-1][0]!=5)
  term1 = storage[n-1][l][k-1];
 else
  term1 = gnMaxBound(k,n-1,l);

}
else if (l < min(Z+c1,limKNless1))
{
 if(storage[n-1][l][k-1][0]!=5)
  term1 = storage[n-1][l][k-1];
 else
  term1 = gnZ(k,n-1,l);
 //term1 = gnZ(k,n-1,l);
}
else
{
 if(storage[n-1][l][k-1][0]!=5)
  term1 = storage[n-1][l][k-1];
 else
  term1 = gFinal(k,n-1,l);
 //term1 = gFinal(k,n-1,l);
}
```

```
if(m < min(minBound,limKNless1))
{
 if(storage[n-1][m][k-1][0]!=5)
  term2 = storage[n-1][m][k-1];
 else
  term2 = gnMinBound(k,n-1,m);
}
else if (m < min(maxBound,limKNless1))
{
 if(storage[n-1][m][k-1][0]!=5)
  term2 = storage[n-1][m][k-1];
 else
  term2 = gnMaxBound(k,n-1,m);
}
else if (m < min(Z+c1,limKNless1))
{
 if(storage[n-1][m][k-1][0]!=5)
  term2 = storage[n-1][m][k-1];
 else
  term2 = gnZ(k,n-1,m);
}
else
{
 if(storage[n-1][m][k-1][0]!=5)
  term2 = storage[n-1][m][k-1];
 else
  term2 = gFinal(k,n-1,m);
}
double * corr;
//'new' vector must be created to hold sum of term1 & term2
corr=vecAdd2(term1,term2);
double hold = amtcl(i+j)*vecMult(corr,s);
//'new' vector, 'corr', no longer needed so deleted
delete[] corr;
```

```
  return hold;
 }
 else   //i=Z+c1,....infinity
 {

  int l=i-c2;
  //checks storage matrix in case the g(k,n-1,l) required has already
been computed
  if(n==1)
  {
   if(storage[0][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[0][l][k-1],s);
   return amtcl(i+j)*vecMult(g0(k,l),s);
  }

  if(l < min(minBound,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnMinBound(k,n-1,l),s);
  }

  if (l < min(maxBound,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnMaxBound(k,n-1,l),s);
  }

  if (l < min(Z+c1,limKNless1))
  {
   if(storage[n-1][l][k-1][0]!=5)
    return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gnZ(k,n-1,l),s);
  }
```

```
   if(storage[n-1][l][k-1][0]!=5)
     return amtcl(i+j)*vecMult(storage[n-1][l][k-1],s);
   return amtcl(i+j)*vecMult(gFinal(k,n-1,l),s);
  }
}
//computes psi(n,i,j(u))
double psi0(int n,int i,int j)
{
 double prob=0;
 if(n>1)
 {
  for(int a=min(n-1,w);a>0;a--)
  {
   prob+=amtwt(a)*psi(a,n-a,i,j);
  }
  //so g's calculated in increasing order- more efficient
 }

 if(n<=w) //since first claim must occur before the max. waiting time
 {
  int xn = getX(n);
  int secIndex = u+c1*xn+c2*(n-xn);
  return prob+amtwt(n)*amtcl(secIndex+j)*delta(i,secIndex);
 }

 return prob;
}
//used to write to file while deleting elements
void specialWriteAndDelete(int nChange)
{

 //done in 'psicdf' , hence 'file' is opened & closed outside of this
 method
  for(int m=0;m<(u+c1*(nChange+min(nChange-1,w)-1));m++)
```

```
  {
   for(int a=0;a<min(nChange-1,w); a++)
   {
    int wasEmpty;

    if(storage[nChange][m][a][0] == 5)
    {
    wasEmpty = 1;
    fwrite(&wasEmpty, 1, sizeof(int), file);
    }

    else
    {
    wasEmpty = 0;
    fwrite(&wasEmpty, 1, sizeof(int), file);
    fwrite(storage[nChange][m][a], 100, sizeof(double), file);
    delete[] storage[nChange][m][a];
    }
   }
  }
}
//used only to delete (while in 'psicdf')
void specialDel(int n)
{
 for(int m=0;m<(u+c1*(n+min(n-1,w)-1));m++)
  {
   for(int a=0;a<min(n-1,w); a++)
   {
    if(storage[n][m][a][0]!=5)
     delete[] storage[n][m][a];

     //test deletion
     //storage[n][m][a]=0;
   }
  }
```

```
}

//computes the sum of psi(l,i,j(u)) for l=1,...n-1, i=minBound,...x,
  j=1,...y
double psicdf(int n,int x,int y)
{
 //prob. T<n,i<=x,j<=y
 double prob=0;
 for (int a=1;a<n;a++)
 {
  for(int b = minBound;b<(x+1);b++)
  {
   for(int d=1;d<(y+1);d++)
   {
    prob += psi0(a,b,d);
   }
  }
  //deletes as goes along elements no longer needed
  if(a>=(w+3))
  {
   //of 2 functions below, use first when writing elements to file,
   //and second when only deleting
   //specialWriteAndDelete(a-w-3);
   specialDel(a-w-3);
  }
 }
 return prob;
}
//delete elements not yet deleted at end of program
void deletSto(int n)
{
 int start=0;

 //in case below, change starting point to account for elements already
 deleted throughout
```

```
 if (n > (w+3))
  start=n-w-3;
 for(int l=start;l < (n-2);l++)
 {
  for(int m=0;m<(u+c1*(l+min(l-1,w)-1));m++)
  {
   for  (int a=0;a<min(l-1,w); a++)
   {
    if(storage[l][m][a][0]!=5)
     delete[] storage[l][m][a];
   }
  }
 }

 //removes S
 for(int a=0;a<w;a++)
 {
  delete S[a];
 }
 delete[] S;
 //removes s
 delete[] s;
 //removes e1;
 delete[] e1;
 delete[] last;
}
//starts program and console window where parameters are set by user
void main()
{
 int n,x,y;
 char o;
 for (;;)
 //for(int a=1;a<5;a++)
 {
 cout << "Please enter the amount of surplus: ";
```

```
cin >> u;
cout << "\nPlease enter n:";
cin >> n;
cout << "\nPlease enter x:";
cin >> x;
cout << "\nPlease enter y:";
cin >> y;
cout << "\nPlease enter the max waiting time:";
cin >> w;
cout << "\nPlease enter value of the boundary on surplus:";
cin >> Z;
cout << "\nPlease enter amount of premium below the boundary:";
cin >> c1;
cout << "\nPlease enter amount of premium above the boundary:";
cin >> c2;
cout << ".....calculating....\n"<<endl;


cout.precision(5L);
//initialize storage matrix
initialize(n);
//write elements into storage matrix
//readFromFile("mem21z20n100.dat");   //need to fix file from
//which reading before run program
//cout << "finished reading in" << endl;
//cout << storage[0][1][1][0] << endl;
//cout << storage[40][5][30][0] << endl;
//cout << storage[10][4][3][0] << endl;
clock_t start = clock();
//open 'file' to which write during program, also needs to be set
before running program
/*file = fopen("mem21z20n250.dat", "wb");
fwrite(&u, 1, sizeof(int), file);
fwrite(&n, 1, sizeof(int), file);
fwrite(&w, 1, sizeof(int), file);*/
```

```
 cout << psicdf(n,x,y)<<endl<<endl;
 cout << "Took " << (clock()-start)/double(CLK_TCK)/60 <<"
 min"<<endl<<endl;

 //write to file remaining elements not written through
 'specialWriteAndDelete'
 //writeToFile(n);
 //global variable 'file' now closing
 //fclose(file);

 //delete elements in storage matrix not yet deleted
 //deletSto(n);

 cout << "\nTo continue enter y, or to quit enter q:";
 cin >> o;
 if (o == 'q' || o == 'Q')
 break;
   }
}
```