1 Introduction

The paper chosen from this project is *Simulation of Ruin Probabilities for Subexponential Claims (S. Asmussen and K. Binswanger)* It explores simulation of probability of ruin given a heavy-tailed, in particular, subexponential, claim size distribution. In particular, 3 Monte Carlo methods are compared.
The goals of my project are to explain the objective and algorithm(s) in the paper, replicate the simulation(s) in the paper and analyze results, and repeat experiment with different parameters to see if we draw the same conclusions.

2 Definitions

2.1 Heavy-tailed Distribution

A distribution of a random variable Y is **heavy-tailed** if $P(Y > y)$ "goes to 0 slower than an exponential distribution". Mathematically, Y is heavy-tailed if $\lim_{y\to\infty} e^{\lambda y} P(Y > y) = \infty \; \forall \; \lambda > 0$

2.2 Subexponential Distribution

A non-negative random variable Y with distribution function F is subexponential if for $Y_1, \ldots, Y_n$ i.i.d., then for all $n \geq 2$,

$$lim_{y\to\infty} \frac{P(Y_1 + \cdots + Y_n > y)}{max(Y_1, \ldots, Y_n) > y} = 1$$

Subexponential distributions are a subset of heavy-tailed distributions.
Examples of subexponential distributions include Pareto, Lognormal, Lévy, Weibull $(0 < shape < 1)$, Burr, etc.

3 Preliminaries

Suppose an insurance company experiences incoming premiums and outgoing claims.

- Let $U(t)$ denote aggregate assets at time $t > 0$
- Y_i be size of claims (i.i.d. from a subexponential distribution B)
- $N(t)$ represent the number of claims (assume a Poisson Process with rate $\lambda > 0$, independent of Y_i's)
- $R(t) = \sum_{i=1}^{N(t)} Y_i$ representing the total size of claims at $t > 0$. This is a Compound Poisson Process
- $U(t) = U(0) + ct - R(t)$ denotes the balance at time t where U(0) is initial surplus, ct denotes incoming premiums and $R(t)$ denotes outgoing claims.

$U(t) = u + ct - R(t)$ denotes the balance at time t

- $u = U(0)$ is the initial fund amount (at time 0)
- c is rate at which premiums are received per unit time
- We assume $c > E[R(1)]$ (the expected payout amount per unit time), then define $\theta > 0$ as $c = (1+\theta)E[R(1)] = (1+\theta)\lambda E[Y]$

The probability of ruin is defined as

$$\Psi(u) = P(U(t) < 0 \text{ for some } t > 0)$$

We will try to estimate this value through Monte Carlo simulations, given a claim size distribution B and $\theta > 0$

4 Algorithms

Note that the goal for Monte Carlo is to simulate $\Psi(u) = E[Z]$ for some random variable Z.

4.1 Algorithm I

It can be shown that $1 - \Psi(u)$ is a compound geometric distribution function; that is,

$$\Psi(u) = P(S_K > u) = E[I(S_K > u)] = E[Z]$$

- where $S_K = \sum_{i=1}^{K} X_i$
- $K \sim Geometric(p), p = \frac{1}{1+\theta}$, independent of X_i's
- $X_1, X_2 \ldots$ are non-negative i.i.d. random variables with common density b_0 and cdf B_0 where $b_0 = \frac{1}{E[B]}(1 - B(s))$ and $B(s)$ is the distribution of the claims.

Note that $\Psi(u)$ depends on the parameter θ.

4.2 Algorithm II: Conditional Monte Carlo

Conditioning on $X_1, \ldots, X_{K-1}$, we get:

$$\begin{aligned}
\Psi(u) &= E[Z] && (1)\\
&= E[E[Z|X_1, \ldots, X_{K-1}]] && (2)\\
&= E[E[I(S_K > u)|X_1, \ldots, X_{K-1}]] && (3)\\
&= E[P(X_1 + \cdots + X_K > u|X_1, \ldots, X_{K-1})] && (4)\\
&= E[P(X_K > u - X_1 - \cdots - X_{K-1}] && (5)\\
&= E[\bar{B_0}(u - X_1 - \cdots - X_{K-1}] && (6)\\
& && (7)
\end{aligned}$$

where $\bar{B}_0(s)$ is defined as $1 - B_0(s)$

4.3 Algorithm III: Another Conditional Monte Carlo

Conditioning on order statistics and with some additional calculations, the authors got a new CMC estimator:

$$\frac{\bar{B}_0(max(X_{(n-1)}, u))}{\bar{B}_0(X_{(n-1)})}$$

The authors gave a proof that this estimator is asymptotically (logarithmically) efficient: i.e.

$$\liminf_{n\to\infty} \frac{\log \sigma_Z}{\log \Psi(u)} \geq 1$$

Therefore, we expect this estimator to perform the best.

5 Implementation

We assume that claim sizes follow a Pareto(1,2) distribution, $\theta = 0.1$, $n = 100000$ and write the following code:

5.1 B0_hat

First, we create a function B0_hat(x) where x is any real number.
Recall that $B_0(x)$, the cdf mentioned in the slide *Algorithm I* and recall $\bar{B_0}(x) = 1 - B_0(x)$. The function B0_hat(x) returns $\bar{B_0}(x)$ for any x.

```
B0_hat = function(x){
  if (x >=1){
    z = (2*x)^-1
  } else if (1 > x & x>=0){
    z = 1 - 0.5*x
  } else{
    z = 1
  }
  return(z)
}
```

5.2 alg1

This is the algorithm for the 1st Monte Carlo algorithm which is straightforward as we are taking $\hat{\Psi(u)} = \frac{1}{n}\sum_{i=1}^{n}[I(S_{K,i} > u)]$

My implementation is as follows:

```
alg1 = function(u){
  p = 1 - (1+0.1)^-1
  z = rep(NA,100000)
  k = rgeom(100000,p)
  for (i in 1:100000){
    if (k[i] >=1){
      v = runif(k[i])
      x = ifelse(v >= 0.5, (2*(1-v))^-1, 2*v)
      z[i] = ifelse(sum(x)>u, 1, 0)
    }
    else{
      z[i] = 0
    }
  }
  return(c(mean(z),1.96*sd(z)/sqrt(100000)))
}
```

5.3 alg2

This is the algorithm for the 2nd Monte Carlo algorithm.
Here, we are taking an estimator

$$\hat{\Psi(u)} = \frac{1}{n}\sum_{i=1}^{n}(\bar{B_0}(u - X_{1,i} - \cdots - X_{K-1,i}))$$

It is important to note that the method the authors use to generate random variables from B0_hat is through inversion. We are using the following fact:
For $Pareto(a, b)$ claim-size distributions, then

$$B_0^{-1}(x) = \frac{ab}{b-1}xI(x < \frac{b-1}{b}) + \frac{a}{(b(1-x))^{\frac{1}{b-1}}}I(x \geq \frac{b-1}{b})$$

My implementation is as follows:

```
alg2 = function(u){
  p = 1 - (1+0.05)^-1
  z = rep(NA,100000)
  k = rgeom(100000,p)
  for (i in 1:100000){
    if (k[i]-1 >=1){
      v = runif(k[i]-1)
      x = ifelse(v >= 0.5, (2*(1-v))^-1, 2*v)
      y = u - sum(x)
      z[i] = B0_hat(y)
    }
    else{
      z[i] = B0_hat(u)
    }
```

```
  }
  return(c(mean(z),1.96*sd(z)/sqrt(100000)))
}
```

5.4 alg3

This is the algorithm for the 3rd Monte Carlo algorithm.
Here, we are taking an estimator

$$\frac{1}{n}\sum_{i=1}^{n}\frac{\bar{B}_0(max(X_{(n-1),i},u))}{\bar{B}_0(X_{(n-1),i})}$$

As in Algorithm II, we generate random variables from B0_hat through inversion. My implementation is as follows:

```
alg3 = function(u){
  p = 1 - (1+0.05)^-1
  z = rep(NA,100000)
  k = rgeom(100000,p)
  for (i in 1:100000){
    if (k[i] >=2){
      v = runif(k[i])
      x = ifelse(v >= 0.5, (2*(1-v))^-1, 2*v)
      x = sort(x)[-length(x)]
      y = u - sum(x)
      m = max(x)
      z[i] = B0_hat(max(y,m))/B0_hat(m)
    }
    else{
      z[i] = B0_hat(u)/B0_hat(0)
    }
  }
  return(c(mean(z),1.96*sd(z)/sqrt(100000)))
}
```

6 Results

The results for each Algorithm given $u = 10, 50, 100, 500, 1000$ follows compared with the real values follows:

6.1 Algorithm I

Table 1: Algorithm I Simulation Results

u	actual value (only 2 sig digits)	our generated confidence interval
10	0.55	0.563 ± 0.0031
50	0.19	0.192 ± 0.0024
100	0.085	0.087 ± 0.0017
500	0.011	0.011 ± 0.00065
1000	0.0054	0.0052 ± 0.00044

The predicted values are close to the true values. It is impossible to tell definitively whether or not the real values fit inside the confidence intervals because the real values are only listed in the paper to 2 significant digits. We can extrapolate that perhaps *some* are within our confidence intervals.

6.2 Algorithm II

Table 2: Algorithm II Simulation Results

u	actual value (only 2 sig digits)	our generated confidence interval
10	0.55	0.565 ± 0.0028
50	0.19	0.196 ± 0.0023
100	0.085	0.086 ± 0.0017
500	0.011	0.012 ± 0.00063
1000	0.0054	0.0054 ± 0.00043

The predicted values are also close to the true values. In fact, for large u, the predicted values appear much closer to the real value than in Algorithm I.
Furthermore, the width of the confidence intervals are lower than that of Algorithm I, although not by a large degree.

6.3 Algorithm III

Table 3: Algorithm III Simulation Results

u	actual value (only 2 sig digits)	our generated confidence interval
10	0.55	0.564 ± 0.0027
50	0.19	0.193 ± 0.0019
100	0.085	0.087 ± 0.0012
500	0.011	0.012 ± 0.00023
1000	0.0054	0.0054 ± 0.00011

The predicted results are also close to the true values. Similar as in Algorithm II, for large u, the predicted values appear much closer to the real value than in Algorithm I.

The width of the confidence interval are lower than that of Algorithm II for all u. But note that as u increased, this reduction became very large.
We conclude that due to this massive decrease in variance that Alg III probably performed the best for the $Pareto(1, 2)$ distribution.

7 Extension

7.1 Change to $\theta = 0.2$

For the extension portion, I first looked at changing $\theta = 0.1$ to $\theta = 0.2$
Note here that we do not have the real values anymore so the sample values cannot be compared against the true values.
Running simulations for $u = 10, 50, 100, 500, 1000$ for $Pareto(1, 2)$, $\theta = 0.2$, $n = 100000$ yields

Table 4: Simulation Results $\theta = 0.2$

u	Alg I	Alg II	Alg III
10	0.737 ± 0.0027	0.738 ± 0.0025	0.739 ± 0.0024
50	0.385 ± 0.0030	0.386 ± 0.0029	0.389 ± 0.0026
100	0.217 ± 0.0026	0.218 ± 0.0025	0.217 ± 0.0021
500	0.0276 ± 0.0010	0.0278 ± 0.0010	0.0275 ± 0.00051
1000	0.0120 ± 0.00067	0.01150 ± 0.00064	0.0117 ± 0.00023

7.2 Change to $\theta = 0.05$

I first looked at changing θ to $\theta = 0.05$
Note here that we do not have the real values anymore so the sample values cannot be compared against the true values.
Running simulations for $u = 10, 50, 100, 500, 1000$ for $Pareto(1, 2)$, $\theta = 0.05$, $n = 100000$ yields

Table 5: Simulation Results $\theta = 0.05$

u	Alg I	Alg II	Alg III
10	0.737 ± 0.0027	0.738 ± 0.0025	0.739 ± 0.0024
50	0.385 ± 0.0030	0.386 ± 0.0029	0.389 ± 0.0026
100	0.217 ± 0.0026	0.218 ± 0.0025	0.217 ± 0.0021
500	0.0276 ± 0.0010	0.0278 ± 0.0010	0.0275 ± 0.00051
1000	0.0120 ± 0.00067	0.01150 ± 0.00064	0.0117 ± 0.00023

As before, the width of the confidence interval clearly decreased from Alg I to Alg II and from Alg II to Alg III. Especially for large u, the variance reduction of Alg III is dramatic so Alg III performed the best when $\theta = 0.2$.

8 Comparison of Results

From their simulations, the authors concluded that

1. Alg I works great for small u but underestimates Ψ for large u
2. Alg II overestimates Ψ for small u and underestimates Ψ for large u
3. Alg III doesn't overestimate or underestimate

From our simulation results above:

1. In Alg I, I did not notice any underestimation
2. I did notice that Alg II does *appear* to overestimate $\Psi(u)$ (but not that it underestimates Ψ for large u although cannot know for sure)
3. For Alg III, I did not notice any over/under estimation

9 Conclusions

Algorithm III was the main focal point of the papers and the authors set out to prove that it performed the best.
In my replications, I assumed a $Pareto(1, 2)$-distributed claim sizes for $u = 10, 50, 100, 500, 1000$. For $\theta = 0.1$, the sample values were all close to to the actual values. In addition, the width of the confidence interval decreased from Alg I to Alg II to Alg III for all u. While the difference from Alg I to Alg II was minor, that from Alg II to Alg III was massive and reinforces that Alg III performed the best.
For the extension simulations, I looked at the cases where $\theta = 0.2$ and $\theta = 0.05$. Here, we did not have any real values; however, the sample values were all close to each other. As with $\theta = 0.1$, the width of the confidence interval decreased from Alg I to Alg II to Alg III for all u. We again saw that while the difference from Alg I to Alg II was minor, that from Alg II to Alg III was massive.
From our replications, we conclude that Algorithm III improves heavily upon naive Monte Carlo (Algorithm I) and is the best out of the 3 algorithms.

10 References

[1] Asmussen, S., &; Binswanger, K. (2014, August 29). *Simulation of ruin probabilities for subexponential claims: Astin Bulletin: The Journal of the IAA*. Cambridge Core.