Univariate Simulation

Computational Statistics Assignment 2

Anders Gantzhorn Kristensen

10-11-2022

UNIVERSITY OF COPENHAGEN

Monte Carlo integration

Let $X_1, X_2, \dots \sim U(-1.9, 2)$ and define

$$S_n = 30 + \sum_{k=1}^{n} x_k$$

We'll be considering ruin probabilities of the form

$$p(n) = P(\exists k \leq n : S_k \leq 0) \overset{(\dagger_1)}{=} \mathbb{E}\left[I_{(\exists k \leq : S_k \leq 0)}\right]$$

Where $\dagger_1$ is the key that let's us use monte carlo integration i.e. SLLN. In particular, we'll be estimating $p(100)$ by

$$p(100) = \frac{1}{N} \sum_{i=1}^{N} I_{(\exists k \leq : S_k \leq 0), i}$$

With N realizations of the experiment

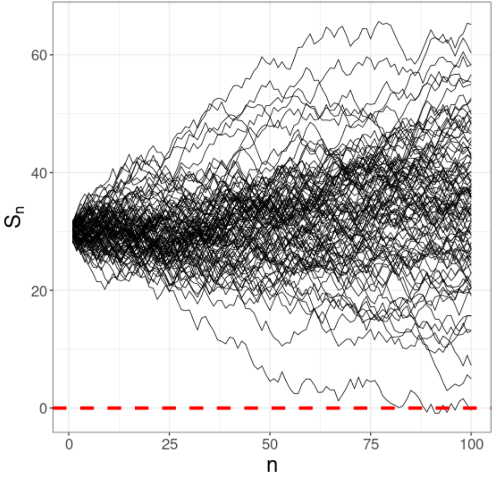
Used R-libraries

```
library(tidyverse)
library(microbenchmark)
library(Rcpp)
library(profvis)
library(riskRegression) #For colCumSum
ggplot2::theme_set(theme_bw())
```

Initial (naive) implementation

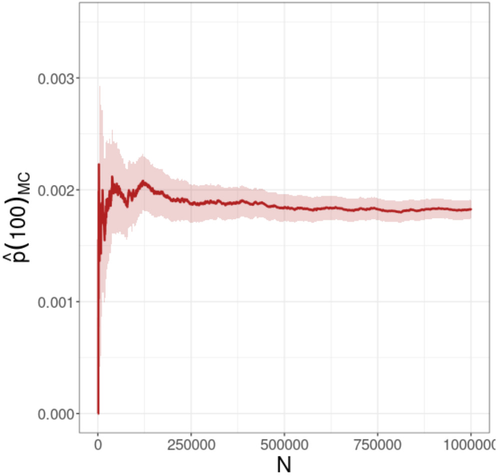
```
capital_flow_calculator <- function(n){
  X <- runif(n, -1.9, 2)
  S <- cumsum(X) + 30
  list(x = seq_along(S), y = S, h = any(S<0))
}
ruin_probability <- function(n = 100, N = 10000){
  ruin_vec <- numeric(N)
  for (i in 1:N){
    ruin_vec[i] <- capital_flow_calculator(n)$h
  }
  list(ruin_prob = mean(ruin_vec), number_of_ruins = sum(ruin_vec),
       number_of_sim = N, outcomes = ruin_vec)
}
}
```

Visualizing 100 samples



Evaluation of naive implementation

By the central limit theorem $\hat{p}(100)_{MC} \overset{\text{approx.}}{\sim} \mathcal{N}\left(p(100), \frac{\sigma^2_{MC}}{N}\right)$



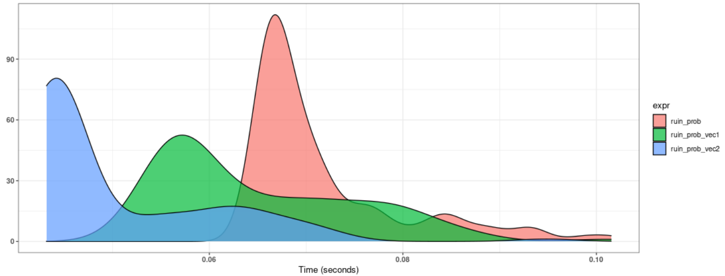
Discussion of vectorization & profiling

```
ruin_probability_vec2 <- function(n = 100, N){
  X <- runif(n*N, min = -1.9, max = 2)
  dim(X) <- c(n, N)
  S <- riskRegression::colCumSum(X) + 30
  I <- ifelse(S < 0, TRUE, FALSE)
  ruin_vec <- colSums(I)>0
  list(ruin_prob = mean(ruin_vec), number_of_ruins = sum(ruin_vec),
  running_ruin_prob = cumsum(ruin_vec)/(1:N),
  number_of_sim = N, outcomes = ruin_vec, path = S)
}
```

Initial profiling and benchmarking

With $N = 10000, \; n = 100.$

	expr	min	lq	mean	median	uq	max	neval
1	ruin_probability	64.38	66.43	75.26	68.43	74.35	414.51	100
2	ruin_probability_vec1	53.90	56.71	65.12	60.71	72.19	101.14	100
3	ruin_probability_vec2	43.17	43.81	50.27	44.57	55.91	94.82	100



Importance sampling

Define

$$g_{\theta,n}(x) = \frac{1}{\varphi(\theta)^n} \exp\left(\theta \sum_{k=1}^{n} x_k\right)$$

for $x \in (-1.9, 2)$ With

$$\varphi(\theta) = \int_{-1.9}^{2} \exp(\theta z \, dz) = \frac{\exp(2\theta) - \exp(-1.9\theta)}{\theta}$$

We simulate with the quantile transformation method. Note

$$Q_{\theta,1}(u) = \frac{\log\left(u\left(\exp(2\theta) - \exp(-1.9\theta)\right) + \exp(-1.9\theta)\right)}{\theta}$$

Importance sampling (continued)

We cummulatively sum realizations like before and check if any entries are negative. Now the estimator is

$$\hat{p}(100)_{IS} = \sum_{i=1}^{N} w(X_i) \cdot I_{(\exists k \leq n : S_{k,IS} \leq 0)}$$

With

$$w(X_i) = \frac{w^*(X_i)}{\sum_{i=1}^{n} w^*(X_i)}, \quad w^*(X_i) = \exp\left(-\theta X_i\right)$$

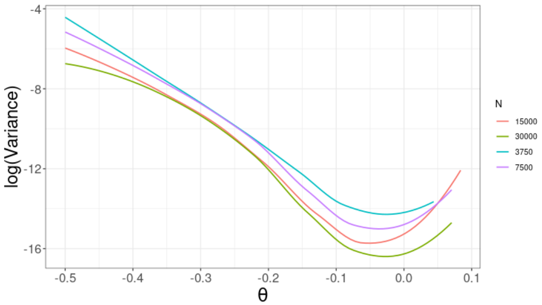
By the Δ-method

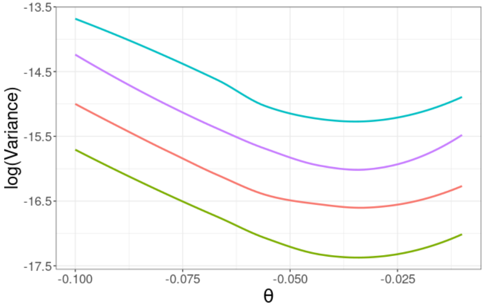
$$\hat{p}(100)_{IS} \overset{\text{approx.}}{\sim} \mathcal{N}\left(p(100), c^{-2}\left(\frac{\sigma_{IS}^2 + p(100)^2\sigma_w^2 - 2p(100)\gamma}{N}\right)\right)$$

Which gives us a way of picking $\theta \in \mathbb{R}$

Tuning of θ

Calculate variance for $N \in \{3750, 7500, 15000, 30000\}$ and along the sequence $\theta \in [-0.5, 0.15]$ in 50 points. After which we do a finer search

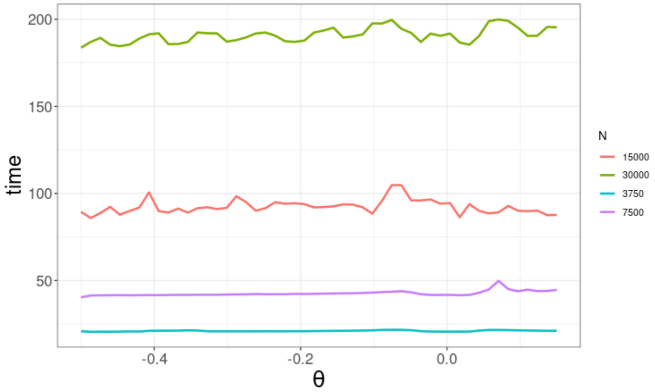




By finding the lowest value of the variance variance for each N and letting θ_{opt} be the mean of the corresponding θ.

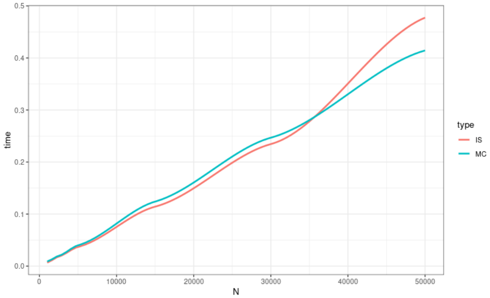
Tuning of θ (continued)

For the same values of N and θ we use microbenchmark and extract the median over 50 computations for each to evaluate the performance on a computation time basis.



Comparing importance sampling and direct implementation

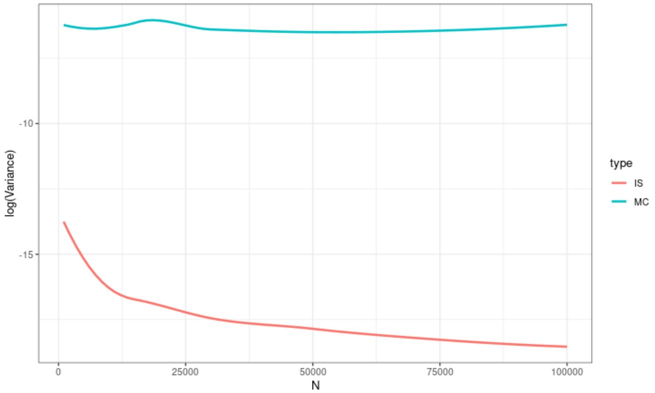
For $N \in \{1000, 2500, 5000, 7500, 10000, 12500, 15000, 30000, 50000, 75000, 100000\}$ we benchmark the naive MC vs our naive IS.



Seem very similar, altough MC might scale a bit better in the end. However...

Comparison of confidence intervals for MC and IS

We need far fewer computations to have a good estimate



Rcpp

We try to implement the inital implementation in C++ via the Rcpp package

```
double ruin_probability_cpp(int n, int N) {
  Rcpp::NumericVector x = Rcpp::runif(n*N, -1.9, 2.0);
  Rcpp::NumericMatrix m(n, N);
  std::copy(x.begin(), x.end(), m.begin());
  Rcpp::NumericMatrix S(n, N);
  Rcpp::LogicalVector I(N);
   for(int i = 0; i < N; ++i){
   Rcpp::NumericVector y = Rcpp::cumsum(m(Rcpp::_, i));
   S(Rcpp::_, i) = y+30;
   I(i) = any(S(Rcpp::_, i)<0).is_true();
   }
  return Rcpp::mean(I);
  }
```

Benchmarking of R implementations vs Rcpp implementation

We compare the 3 implementations made in R to or Rcpp implementation for different choices of N. We plot the median of the computation time