

DANMARKS TEKNISKE UNIVERSITET

---

# Simulation of Lévy Processes

---

JUNE 2020



***Authors:***

Nicolaj Hans Nielsen  
Anton Ruby Larsen  
Andreas Heidelberg Engly  
Karl Emil Takeuchi-Storm

***Study No:***

s184335  
s174356  
s170303  
s130377

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Methodology</b>	<b>1</b>
2.1	Theory . . . . .	1
2.1.1	Lévy Processes . . . . .	1
2.1.2	Compound Poisson process . . . . .	2
2.1.3	Brownian motion . . . . .	2
2.2	Explanation of the algorithm . . . . .	3
2.2.1	A closer look on P, A and M . . . . .	4
2.3	Explanation of constants . . . . .	6
2.3.1	Effects of variation of $\mu$ . . . . .	6
2.3.2	Effects of variation of $\sigma$ . . . . .	7
2.3.3	Effects of variation of $\lambda$ . . . . .	9
2.3.4	Combined values . . . . .	10
2.4	Jump Size Distributions . . . . .	11
2.4.1	Plots of Different Distributions . . . . .	11
2.4.2	Investigation of Variance . . . . .	12
2.4.3	Resulting Runs from Various Jump Distributions . . . . .	16
2.5	First passage probabilities . . . . .	18
<b>3</b>	<b>European call option</b>	<b>22</b>
3.1	Monte Carlo Simulation of Call Option . . . . .	22
3.1.1	Option simulation by control variate . . . . .	24
3.2	Black-Scholes Model . . . . .	24
3.3	European call option experiments . . . . .	26
<b>4</b>	<b>Discussion</b>	<b>26</b>
<b>5</b>	<b>Conclusion</b>	<b>27</b>
<b>6</b>	<b>References</b>	<b>28</b>
<b>A</b>	<b>Derivation of Statistical Moments</b>	<b>29</b>
A.1	Basic Properties of Expectations . . . . .	29
A.1.1	Linearity of Expectation . . . . .	29
A.1.2	Variance . . . . .	29
A.2	Derivation of Statistical Moments . . . . .	29
A.2.1	Pareto Distribution . . . . .	29
A.2.2	Erlang Distribution . . . . .	31
A.2.3	Exponential Distribution . . . . .	32
A.2.4	Hyperexponential Distribution . . . . .	33
<b>B</b>	<b>Code</b>	<b>34</b>
B.0.1	Explanation of the algorithm . . . . .	34
B.0.2	Explanation of the constants . . . . .	35
B.0.3	Jump size distribution . . . . .	43
B.0.4	First passage probabilities . . . . .	51
B.0.5	European Call option . . . . .	56

# 1 Introduction

In this project, we will investigate the properties of a Lévy process and examine how each component of the model influences the overall behavior. We will in detail test the parameters of the diffusion part, the parameters and distribution of the jump term. These types of models are used within the domain of financial engineering and therefore we try to estimate the price of European options. We use a simpler version with only the geometric Brownian motion to model the behavior of the underlying stock because it makes us able to compare with the Black-Scholes formula. With use of Monte-Carlo simulation and control variate as variance reduction, we obtained an approximation very close to the value calculated with the theoretical Black-Scholes formula. We then discussed how the more complicated Lévy process could be a better model to mimic the behavior of stock prices using the insight from the analysis of the model.

## 2 Methodology

### 2.1 Theory

In this project, we work with a Lévy process of type:

$$X_t = \mu t + \sigma B_t + \sum_{i=1}^{N_t} Y_i \quad (1)$$

This is also classified as a jump-diffusion model. The term,  $\sum_{i=1}^{N_t} Y_i$ , is the jump part which resembles a compound Poisson process while the diffusion part is given by,  $\mu t + \sigma B_t$ . The diffusion is modeled as a Brownian motion,  $\sigma B_t$ , with drift,  $\mu t$ .

In the following, we will cover relevant theory for the two parts and give formal definitions. This process is a subclass of the general class of Lévy process. We, therefore, first formally define a Lévy process:

#### 2.1.1 Lévy Processes

The formal definitions of the Lévy Process and the components are given below.

**Definition 2.1 Lévy Process** *Let  $L$  be a stochastic process. Then  $L_t$  is a Lévy process if the following conditions are satisfied:*

1.  $L_0 = 0$
2.  $L$  has independent increments:  $L_t - L_s$  is independent of  $\mathcal{F}_s, 0 \leq s < t < \infty$
3.  $L$  has stationary increments:  $\mathbb{P}(L_t - L_s \leq x) = \mathbb{P}(L_{t-s} \leq x), 0 \leq s < t < \infty$
4.  $L_t$  is continuous in probability:  $\lim_{t \rightarrow s} L_t = L_s$

[8]

Leading to the general formulation:

$$X_t = \mu t + \sigma B_t + Z_t \quad (2)$$

Where  $Z_t$  is the jump process, which defines the different Levy processes.

### 2.1.2 Compound Poisson process

Let the sequence  $\{T_1, T_2, \dots, T_n\}$  be the interarrival times of a number of events.  $\{T_1, T_2, \dots, T_n\}$  are independent random variables identically distributed with an exponential distribution with the parameter,  $\lambda > 0$ . The total number of events,  $N(t)$ , up to time  $t \geq 0$  then follows a Poisson process with intensity  $\lambda$ . This process is very useful because  $\{N(t)\}_{t \geq 0}$  has stationary and independent increments which means that for some  $t > s$ , the increment  $N(t) - N(s)$  is independent of all before time  $s$ . For the process to be a Lévy process this is central because if this was not the case it would not be stationary in its increments.

Now,  $\{N(t)\}_{t \geq 0}$  is a counting process as the step-wise increments - the size of the jumps - are of unit length, 1. In our case, we want the size of the jumps to follow an arbitrary distribution hence we let  $\{Y_i\}_{i \geq 1}$  be a sequence of independent random variables with some distribution  $F$ . We can then define the term from our compound Poisson process as:

$$Z_t = \sum_{i=1}^{N_t} Y_i$$

### 2.1.3 Brownian motion

The Brownian motion is given by the following definition:

**Theorem 2.1 Brownian Motion** *There exists a probability distribution over the set of continuous functions  $B : \mathbb{R} \rightarrow \mathbb{R}$  satisfying the following conditions taken from [5]:*

1.  $B(0)=0$
2. **Stationary** the random variables  $B(t) - B(s)$  is the normal distribution with mean 0 and variance  $t - s$
3. **Independent increment** The random variables  $B(t_i) - B(s_i)$  are mutually independent if the intervals  $[s_i, t_i]$  are non-overlapping

An intuitive way of understanding a Brownian motion is as the "limit" of a random walk. No matter how much you zoom in on the graph you will always get a jagged behavior. This also means that is non-differentiable everywhere.

We are able to simulate the Brownian motion with predetermined time intervals but when the jumps are happening at random exponentially distributed times this makes it a lot harder. We, therefore, introduce the Wiener-Hopf factorization which gives the ability to describe the Brownian motion at random exponentially distributed times.

**Theorem 2.2 Wiener-Hopf factorization** *Let  $T \sim \text{Exp}(\lambda)$*

$$V = \max_{0 \leq t \leq T} \mu t + \sigma B_t, \quad \text{and} \quad W = \left( \max_{0 \leq t \leq T} \mu t + \sigma B_t \right) - (\mu T + \sigma B_T)$$

*Then  $V$  and  $W$  are independent with  $V \sim \text{Exp}(\phi_1)$ ,  $W \sim \text{Exp}(\phi_2)$ , where*

$$\phi_1 = -\frac{\mu}{\sigma^2} + \sqrt{\frac{\mu^2}{\sigma^4} + \frac{2\lambda}{\sigma^2}} \quad \text{and} \quad \phi_2 = \frac{\mu}{\sigma^2} + \sqrt{\frac{\mu^2}{\sigma^4} + \frac{2\lambda}{\sigma^2}}$$

*Notice that in particular*

$$\mu T + \sigma B_T = V - W$$

Taken from [10].

We see that  $V_i$ , in the above theorem 2.2, will denote the maximum value of the Brownian motion in the last period.  $W$  is the maximum in the last period minus the value of the Brownian motion at time  $T$ . Therefore, as stated  $V-W$  will be the value of the Brownian motion at time  $T$ .

In the following, we will use the Wiener-Hopf factorization to simulate the process  $X_t$ . We use a discrete “skeleton” of 3 discrete stochastic processes,  $\{(P_i, A_i, M_i)\}_{i \in \mathbb{N}}$ . We have a set of coordinates for each jump hence for the  $i$ 'th jump, the first coordinate,  $P_i$ , is equal to the process prior to the  $i$ 'th jump.  $A_i$  is equal to the process after the  $i$ 'th jump, and  $M_i$  is equal to the maximum of the process attain up to the time of the  $i$ 'th jump. We define these as:

1.  $P_i = A_{i-1} + (V_i - W_i)$
2.  $A_i = P_i + Y_i$
3.  $M_i = \max\{M_{i-1}, A_{i-1} + V_i, A_i\}$

## 2.2 Explanation of the algorithm

We then look at one simulation of  $P$ ,  $A$ , and  $M$  of 1000 arrivals where we set  $Y_t$  as the exponential distribution with  $\lambda_Y = 2$ ,  $\sim \text{Exp}(\lambda_Y = 2)$ . Further, we set the drift  $\mu = 0.1$ , the Gaussian intensity  $\sigma = 0.5$ , and use intensity of  $\lambda_{\text{jump}} = 1$  for the Poisson process.

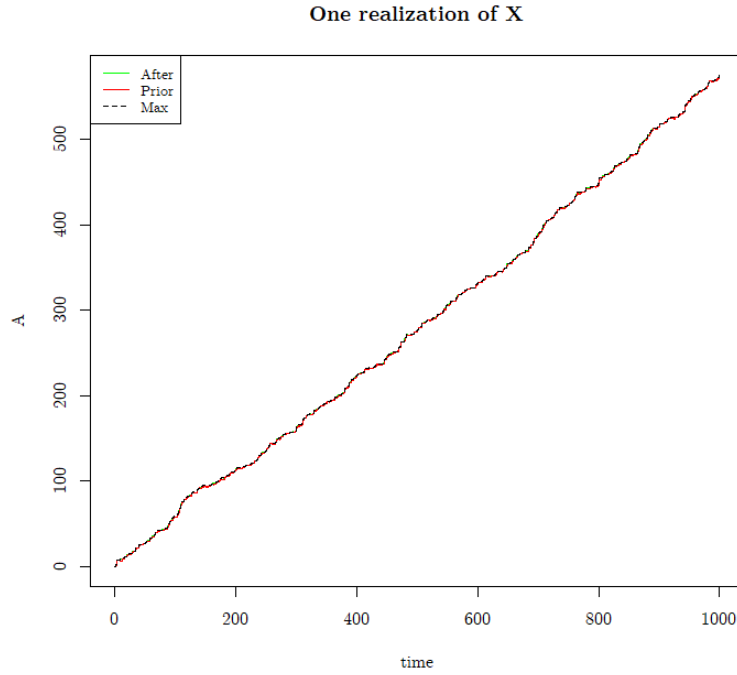


Figure 1: One simulation of 1000 arrivals with  $\lambda = 1$ ,  $\sigma = 0.5$ ,  $\mu = 0.1$ ,  $\lambda_Y = 2$

For that the values of  $A$ ,  $P$ , and  $M$  follows each other closely and increases steadily to around 600. A more local analysis is performed in section 2.2.1. We now question what would happen with the value at arrival 1000, if we repeated the simulation 100 times. The distributions of  $A$ ,  $P$ , and  $M$  at the 1000 arrival is plotted as histograms in Figure 2.

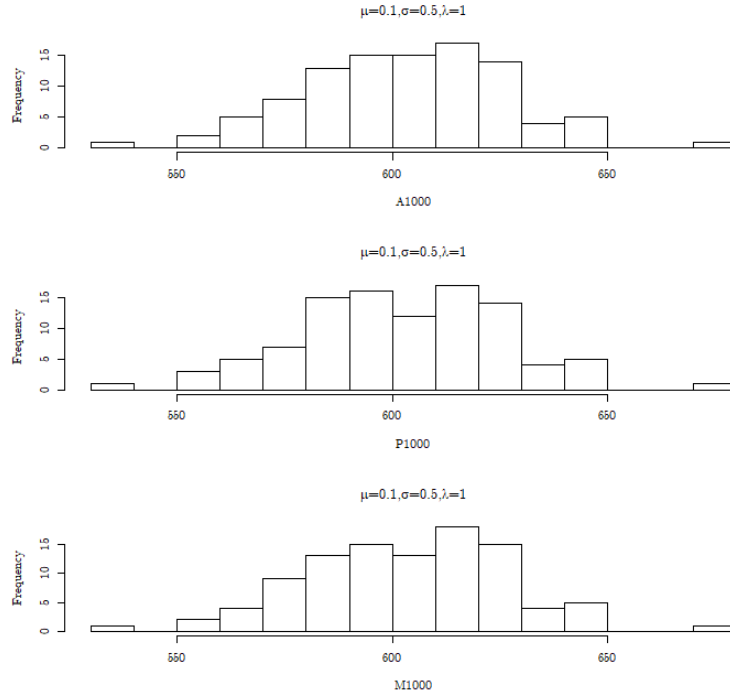


Figure 2: Histograms of 100 realizations of the 1000th arrival of the processes P, A and M

Initially, we see that all the values are almost identically distributed and most values are in the interval from 550 to 650. Therefore, we see that the single realization in figure 1 does not seem to be a special realization.

### 2.2.1 A closer look on P, A and M

In this section, we will take a closer look at what each process P, A, and M actually do. To support the explanation 5 arrivals have been plotted in Figure 3.

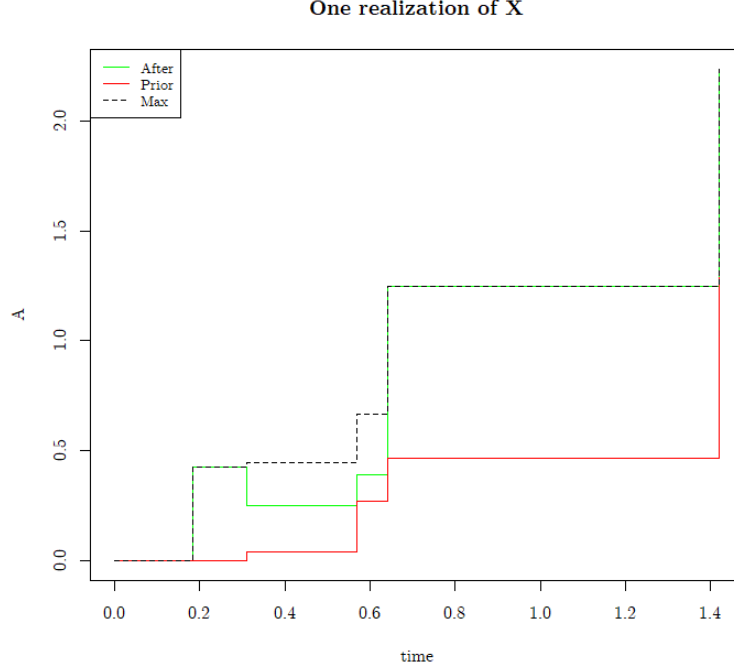


Figure 3: One simulation of 5 arrivals with  $\lambda = 1$ ,  $\sigma = 0.6$ ,  $\mu = 0.1$ ,  $\lambda_Y = 2$

First of all, we see that the time between the arrivals is not equidistant because the jumps happen with inter arrival times,  $T \sim \text{Exp}(\lambda = 1)$ .

**P** We start with  $P$  which is given as  $P_i = A_{i-1} + (V_i - W_i)$ . The first terms is the level of the process after the  $(i - 1)$  jump,  $A_{i-1}$ . To this we add the contribution from the Brownian motion from jump  $(i - 1)$  to  $i$ . We know from the Wiener-Hopf factorization in section 2.1.3 that  $\mu T_i + \sigma B_{T_i} = V_i - W_i$  hence we directly see the contribution as the second term in  $P_i = A_{i-1} + (V_i - W_i)$ . Graphically we can see this in Figure 3 where the red line follows the behavior of the green line but is always below.

**A** The green line in Figure 3 is  $A$  which is always a jump higher than the red line.  $A$  is calculated as  $A_i = P_i + Y_i$  hence it simply is the process prior to the jump,  $P_i$ , and then we add the size of the jump,  $Y_i$  which is a random variable that in this case follows an exponential distribution,  $Y_i \sim \text{Exp}(\lambda_Y = 2)$ .

**M** The last process is  $M$ . It is calculated as  $M_i = \max\{M_{i-1}, A_{i-1} + V_i, A_i\}$ . Here the max is either the same as at the  $i - 1$ 'th jump,  $M_{i-1}$ , otherwise it was attained between the jumps. Since  $V_i$  is the maximal value that the Brownian motion attain between the  $i - 1$  and  $i$  jump, another candidate for the max is  $M_i = A_{i-1} + V_i$ . We can see this max graphically in Figure 3 around time 0.3 and before 0.6 where the max is above  $A_i$ . Lastly, we could also have reached a new max with the jump,  $M_i = A_{i-1} + (V_i + W_i) + Y_i$ . We see this multiple times where the dotted and the green lines lie upon each other e.g. at time 0.2.

## 2.3 Explanation of constants

We continue by describing the constants  $\mu$ ,  $\sigma$ , and  $\lambda$ . As can be seen in equation 1,  $\mu t$  simply adds a trend to the overall function, where a high  $\mu$  will correspond to a large slope in an affine function. The gain of the Brownian motion is  $\sigma$ , which increases the amplitude of the Brownian motion. Lastly the constant  $\lambda$  determines how often jumps occur, as if we increase  $\lambda$  we would decrease the time between jumps. The expectation of an exponential process is  $\frac{1}{\lambda}$  which corresponds to the average time between occurrences. Therefore, a  $\lambda$  under 1 will on average correspond to less than 1 jump per time unit whereas  $\lambda$  above 1 will give more than one jump per time unit. In all simulations in this subsection  $Y_t \sim N(0, 1)$  is used, as we are not interested in the actual jump distribution.

### 2.3.1 Effects of variation of $\mu$

$\mu$  is the drift term in the Brownian motion which will add  $\mu$  to the level of  $X$  every time tick, thus changing the size and sign of the  $\mu$  thus changes the trend in a given model.

We use two plots to describe the behavior of  $\mu$ . A low  $\mu$  on the left has little influence on the behavior, however, it becomes very dominating in the process on the right with a high  $\mu$ .

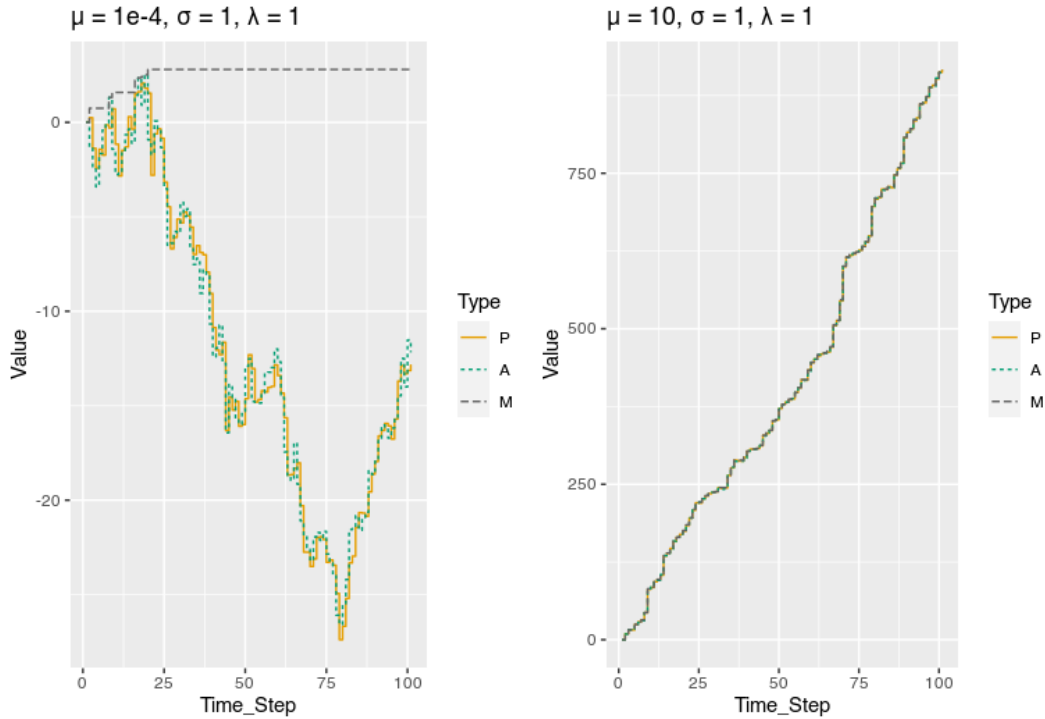


Figure 4: Two simulations of extreme  $\mu$  values

A simulation is carried out 1000 times for 1000 time steps, using  $\mu = [0.1, 0.5, 1, 2, 5, 10]$  and the value for  $\sigma$  and  $\lambda$  of 1. In figure 5 the mean value of processes after the jumps with changing  $\mu$  are plotted. The black line in the graph is used for reference to the Lévy process, with constants (1, 1, 1). As can be seen the trend changes with different  $\mu$  values.



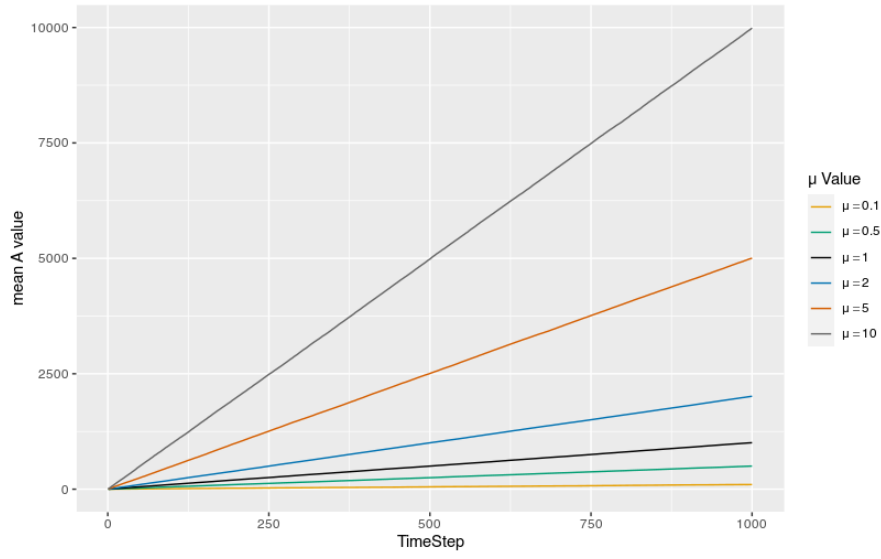


Figure 5: Average *after jump* values for 1000 simulations

### 2.3.2 Effects of variation of $\sigma$

Next  $\sigma$  is the diffusion term in the Brownian motion. It amplifies the standard normally distributed variation from the Brownian motion. Therefore, a high  $\sigma$  will correspond to a more jagged function while a  $\sigma = 0$  will correspond to a process without any random fluctuations.

Next we look at  $\sigma$  which has been plotted in Figure 6.

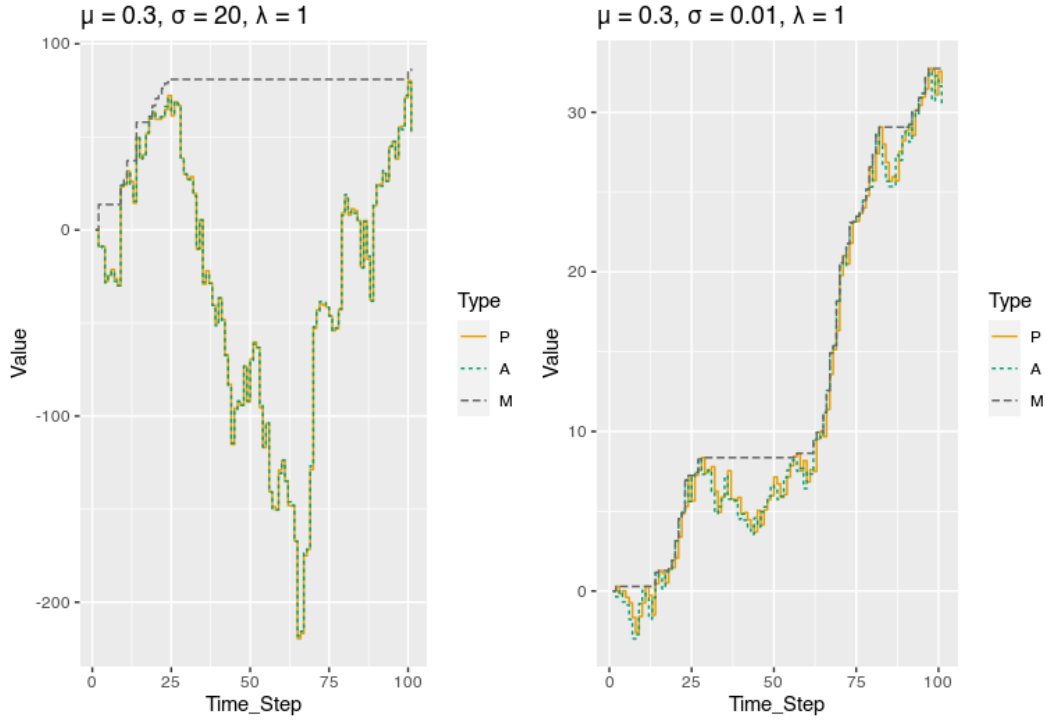


Figure 6: Two simulations of extreme  $\sigma$  values

We see in the left plot where  $\sigma = 20$  that we have huge fluctuation while in the right plot where  $\sigma = 0.01$  the function is increasing steadily without any significant fluctuations.

To further illustrate the effects of  $\sigma$ , similar simulations are carried out a thousand times for a thousand time steps. The simulation has a  $\mu$  of 1 and  $\sigma = [1, 10, 25, 50, 100, 500]$ . The plot in figure 5 shows the average maximum attained value of the Lévy process. The effect of the Brownian motion with  $\sigma$  below 25, is not very noticeable. The maximum attained value, can be seen as a measure of the spread of results at the 1000th time step.

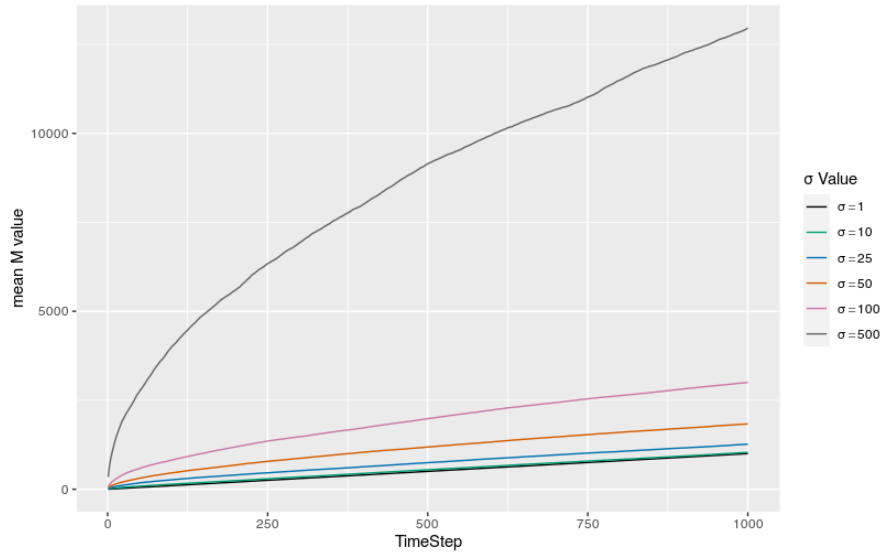


Figure 7: Average *max attained* values for 1000 simulations

### 2.3.3 Effects of variation of $\lambda$

Lastly, we look at  $\lambda$ . If we look at the time scale, we see that in the left plot where  $\lambda$  is small the time scale is large because in average only 1 arrival happens every 20 time units. The right plot is the opposite where in average 20 arrivals happens per time unit.

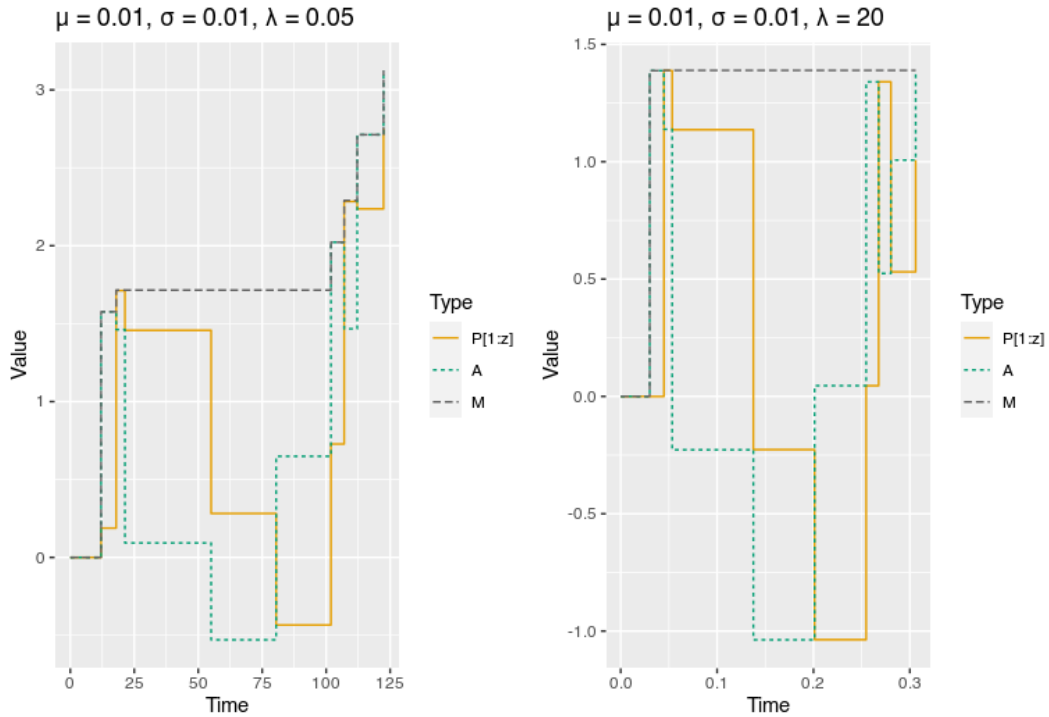


Figure 8: Two simulations of extreme  $\lambda$  values

The above plots show nearly the same motion of jumps, however, in the left plot, the Brownian motion has more time to change between the jumps.

A further look into the behavior of the changed average time steps is obtained by looking at the average of 1000 simulations, where the  $\mu$  has a greater impact when the time steps become longer. In figure 9 the time step size changes, resulting in more time for the function to develop.

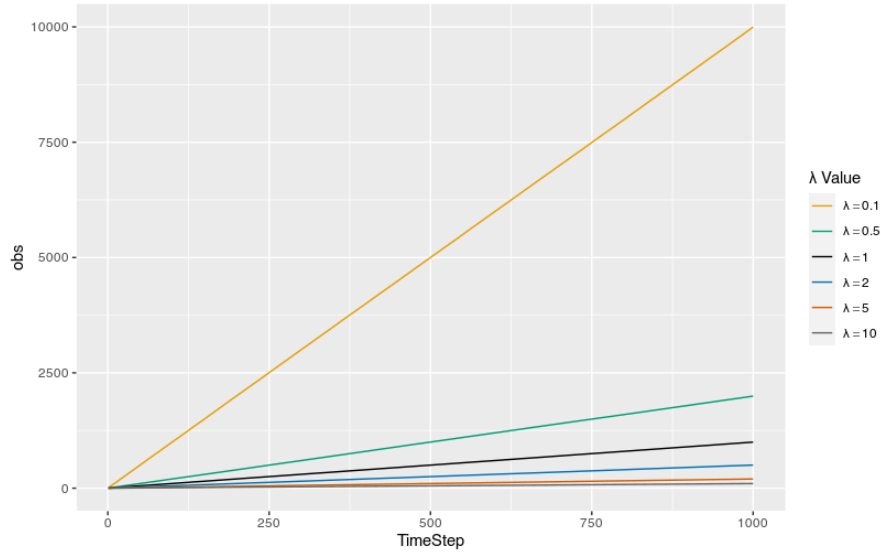


Figure 9: Average *after jump* values for 1000 simulations

#### 2.3.4 Combined values

Comparing the results for the averages in figure 5 for  $\mu = 10$  and figure 9 for  $\sigma = 0.1$ , it is seen that both arrive at the same result. This is explained by the change in time step size. If we inspect a function at 10 times the resolution, then the drift is expected to contribute  $1/10$  per time, which is the case for the two simulations.

## 2.4 Jump Size Distributions

In order to investigate the contribution of  $Y_i$  we will conduct an analysis of the distributions isolated and in the context of a Lévy Process.

### 2.4.1 Plots of Different Distributions

At first, we investigate the plots for the different probability distributions. All the probability density functions have been chosen such that the expectation is 3.

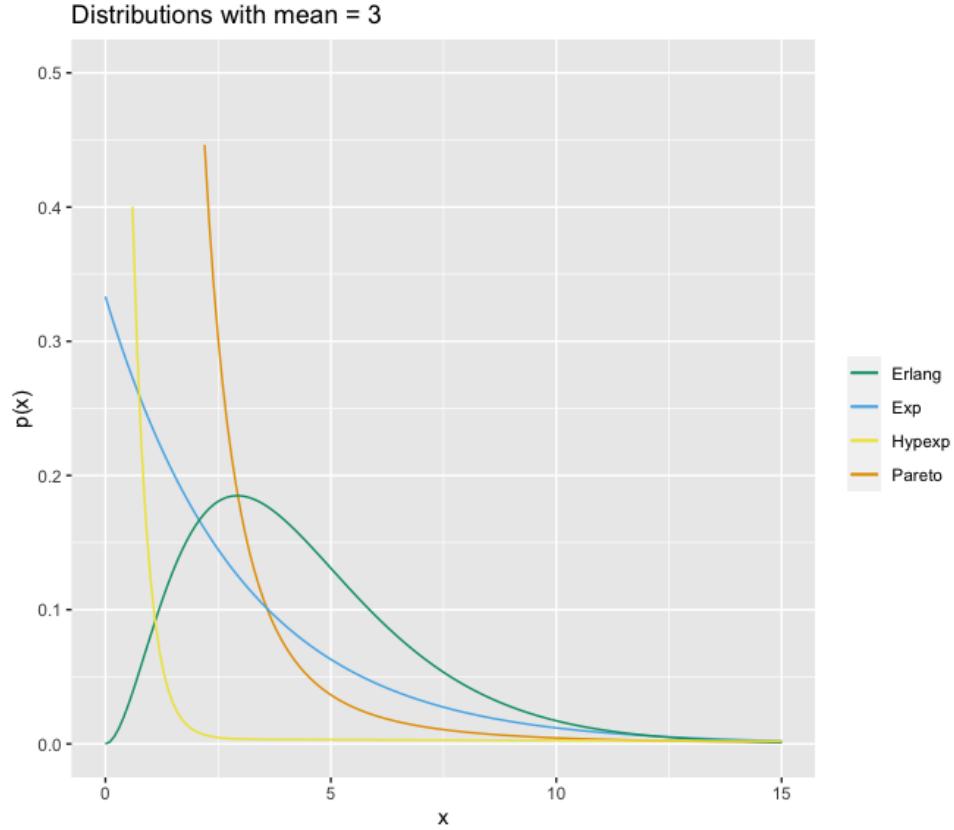


Figure 10: Probability Density Functions

A thing to notice from the probability density functions is that the tails differ in size. Since they are all right skewed, it is of interest to take a closer look at the tails.

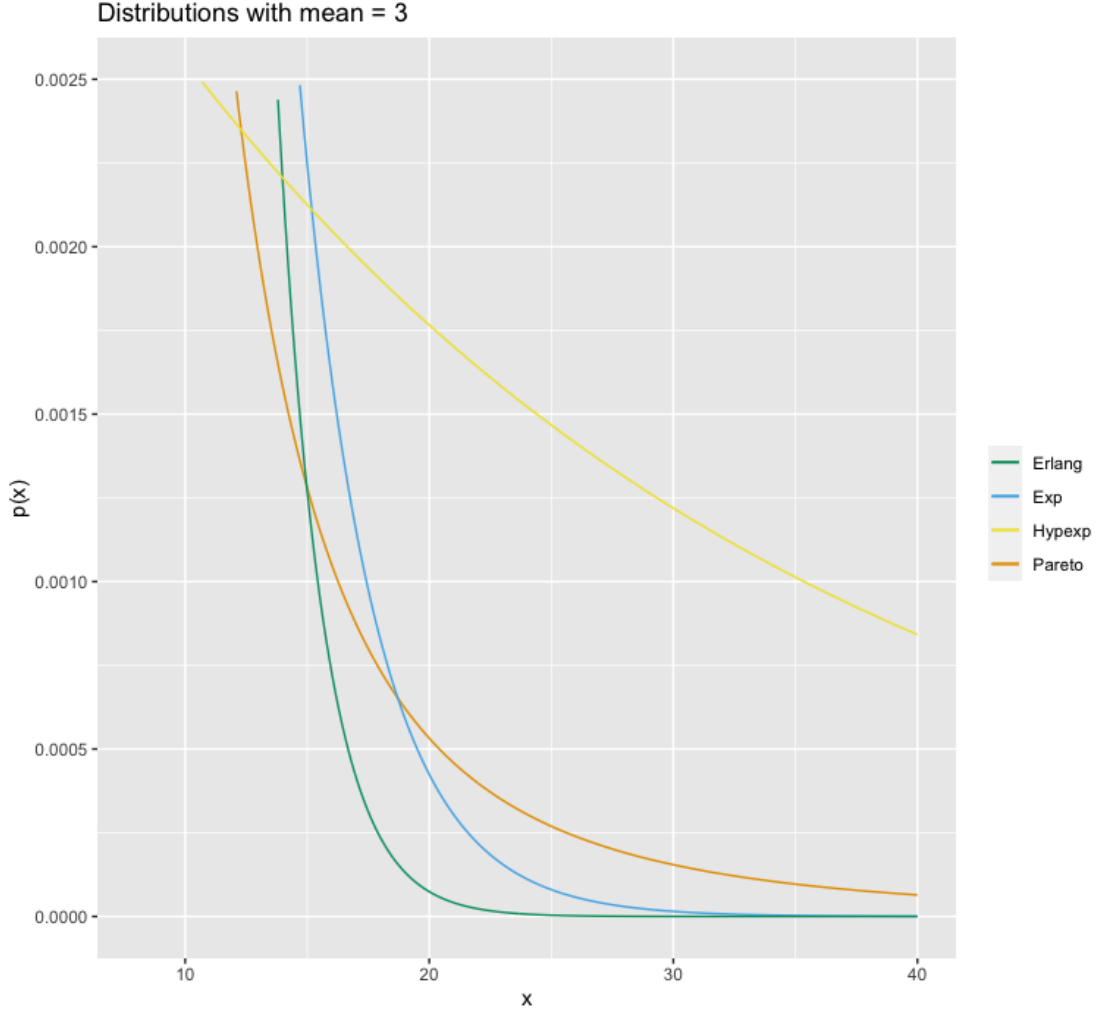


Figure 11: Tails of Probability Density Functions

It is seen that higher values are more probable (relatively) when  $Y_i \sim Hyperexp(p_1 = 0.8, p_2 = 0.2, \lambda_1 = 3, \lambda_2 = 0.037)$ . The increased likelihood of large values is caused by the second term of the distribution, which generate large values with probability  $p_2 = 0.2$ . Hence, we expect to experience a large jump once in a while. The other distributions approaches a slim tail at almost the same pace.

#### 2.4.2 Investigation of Variance

To support our understanding of the distributions with regards to the jumps  $Y_i$ , we will calculate the variance related to each distribution. We refer to the appendix A for exact derivations. Furthermore, we will plot a segment of the run for each case. In a previous subsection the meanings of the constants  $\sigma$ ,  $\mu$ , and  $\lambda$ . In these examples we use fixed  $\sigma = 0.2$ ,  $\mu = 1$ , and  $\lambda = 2$ .

Let  $Y_i \sim Erlang(k = 3, \lambda = 0.68)$ .

$$Var[Y_i] = \frac{k \cdot (k + 1)}{\lambda^2} \approx 6.4$$

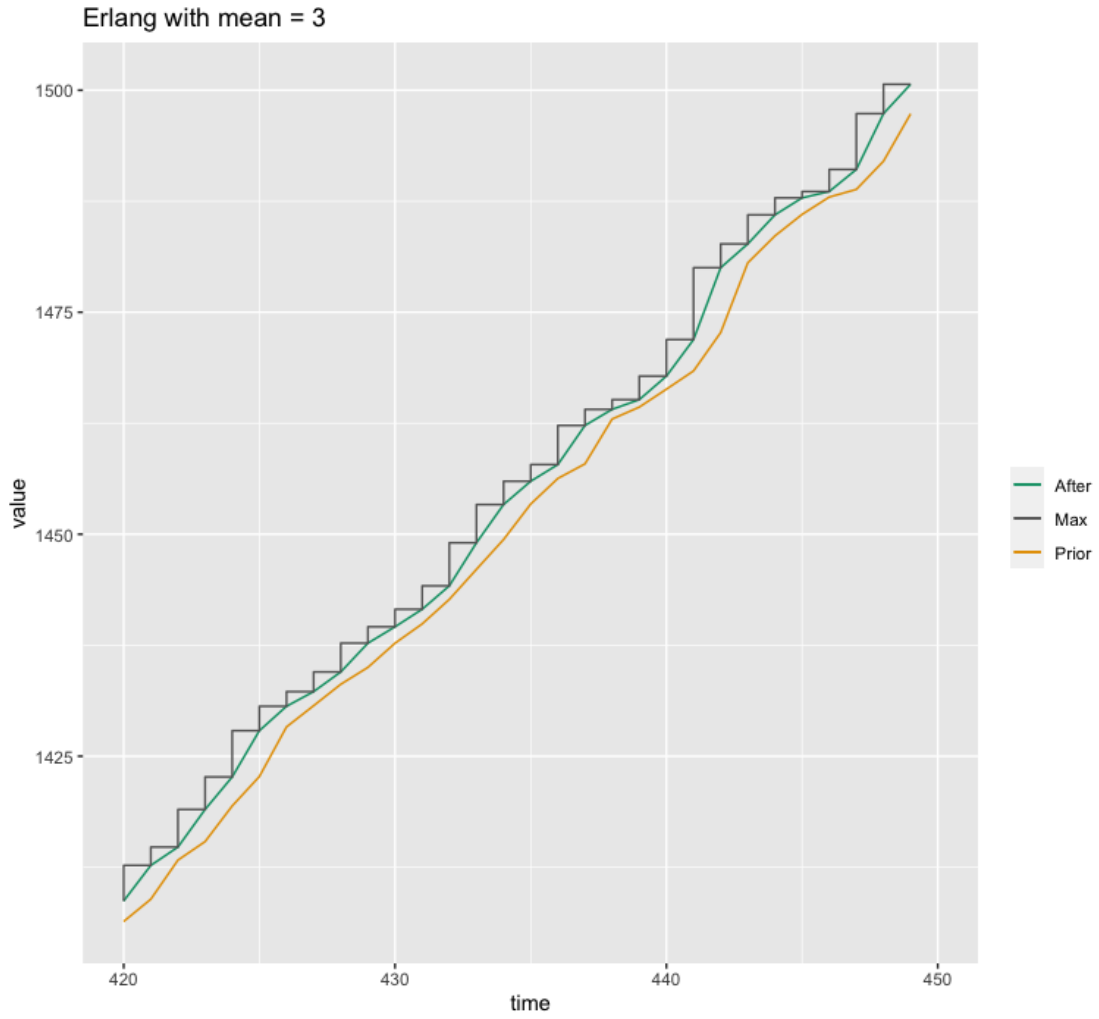


Figure 12: Realization with Erlang distributed jump sizes

Relative to the other distributions to be examined, the jumps sized do not differ hugely. This is due to the very low variance of the distribution.

Let  $Y_i \sim \text{Exp}(\lambda = \frac{1}{3})$ .

$$\text{Var}[Y_i] = \frac{1}{\lambda^2} \approx 9$$

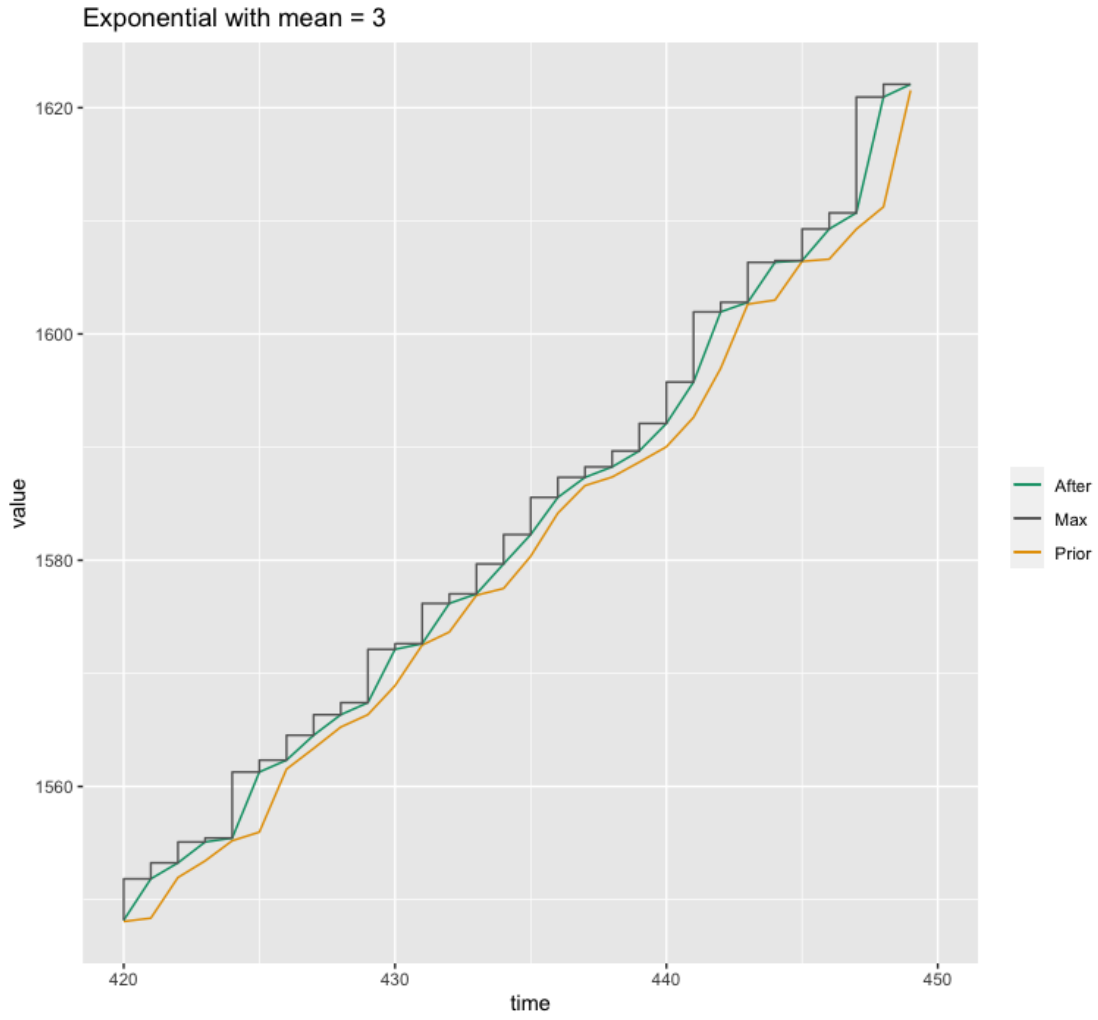


Figure 13: Jump Size distributed by Exponential

As for the Erlang distributed case, the jump sizes do not differ hugely. This comes as no surprise, since the variance is more or less the same.

Let  $Y_i \sim Hyperexp(p_1 = 0.8, p_2 = 0.2, \lambda_1 = 3, \lambda_2 = 0.037)$ .

$$Var[Y_i] = \sum_{i=1}^n p_i \frac{2}{\lambda_i^2} - \left( \sum_{i=1}^n \frac{p_i}{\lambda_i} \right)^2 \approx 137.2$$



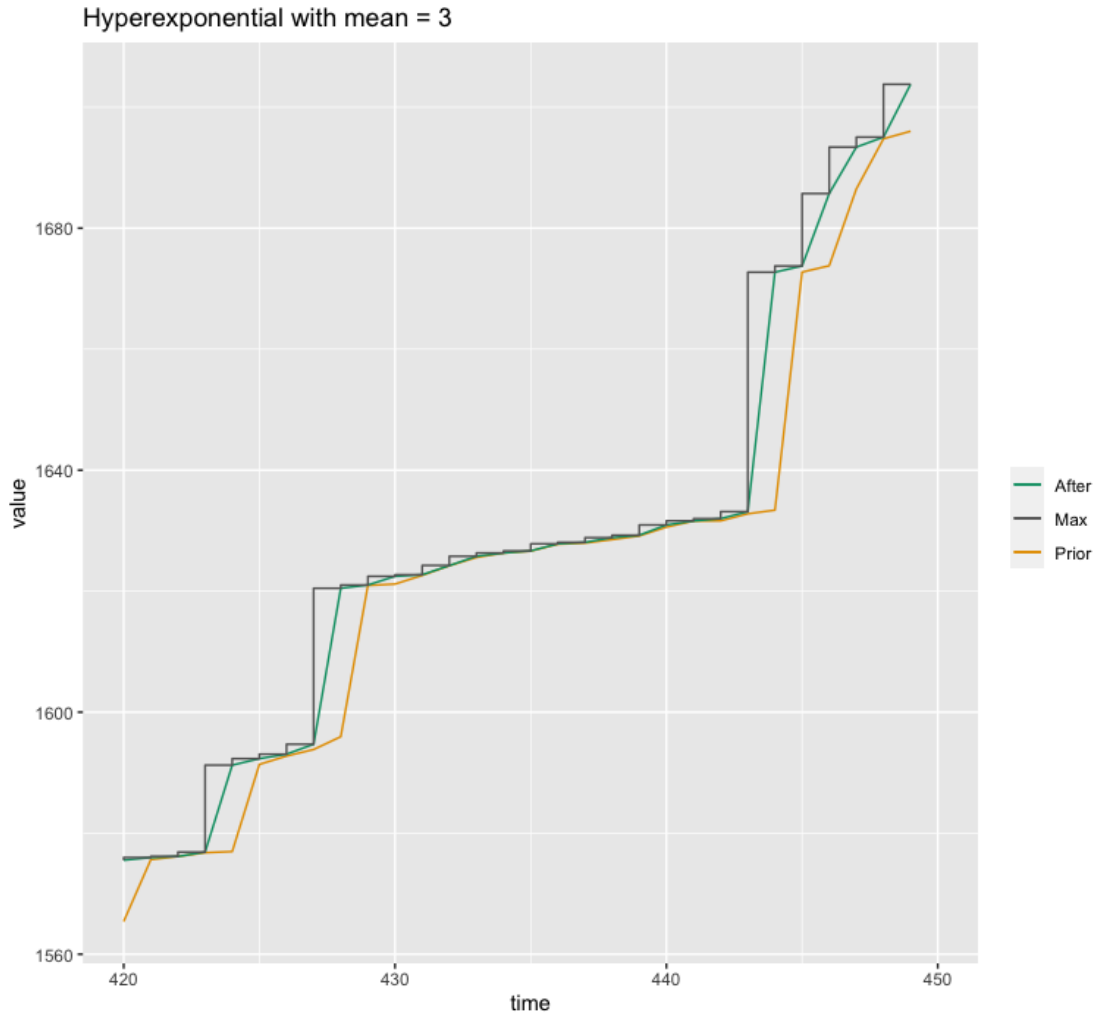


Figure 14: Jump Size distributed by Hyperexponential

Due to the high variance, the jump sizes will once in a while be huge.

Let  $Y_i \sim \text{Pareto}(\beta = 1.53, k = 2.05)$ .

$$\text{Var}[Y_i] = \frac{\beta^2 k}{(k-2)(k-1)^2} \approx 87.8$$

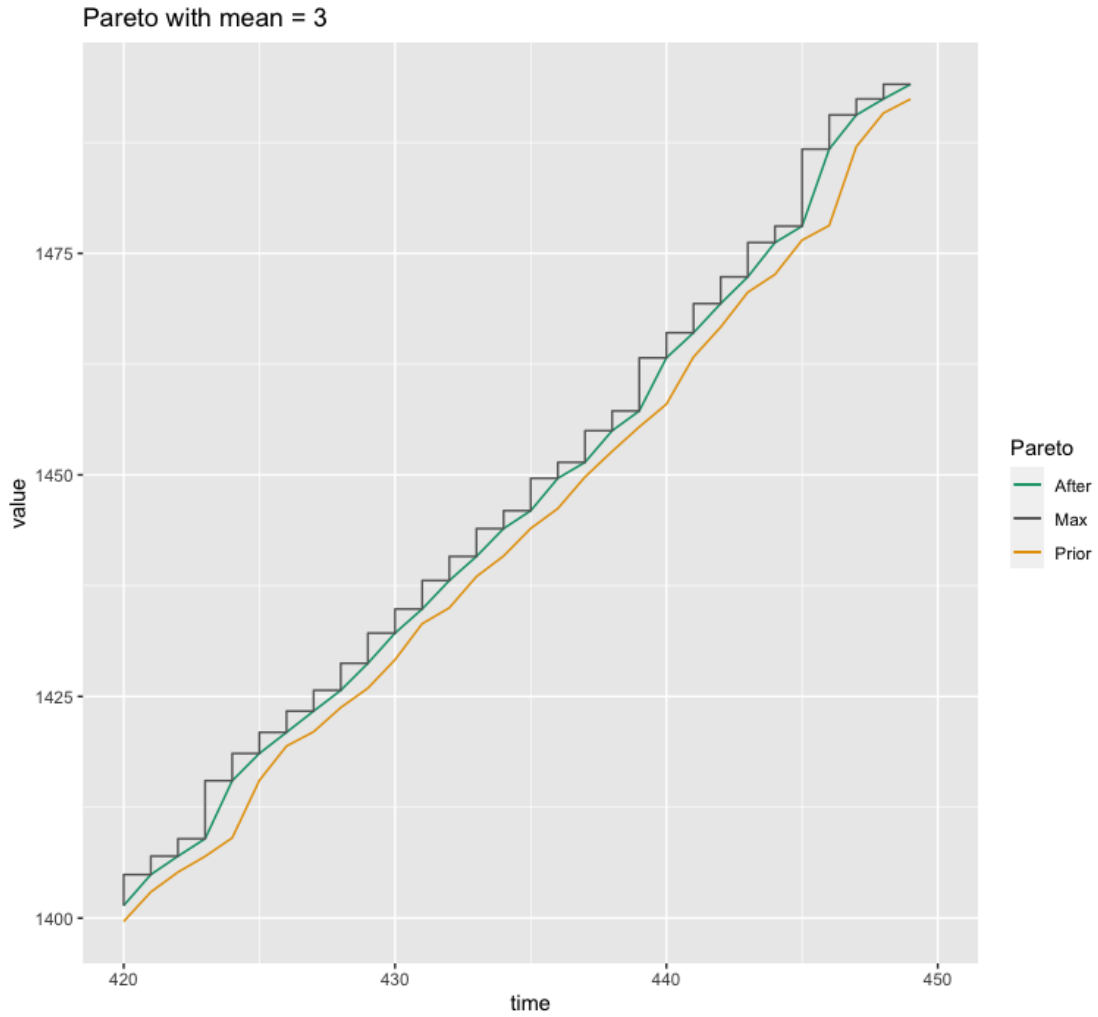


Figure 15: Jump Size distributed by Pareto

The Pareto distributed case will also experience some large jumps. However, as was seen from the distribution plots, it will have a lot of values clustered within a short range of the mean. This is due to the nature of the distribution. What is also seen is that we will observe extreme values with a higher probability than the Erlang and exponentially distributed case. It will, however, not be as extreme as for the hyperexponential case.

With these considerations in mind, we can have a glance at the distribution of the max values resulting from using the different jump sizes.

### 2.4.3 Resulting Runs from Various Jump Distributions

The distributions will be very similar for respectively the prior, the after, and the maximum case.

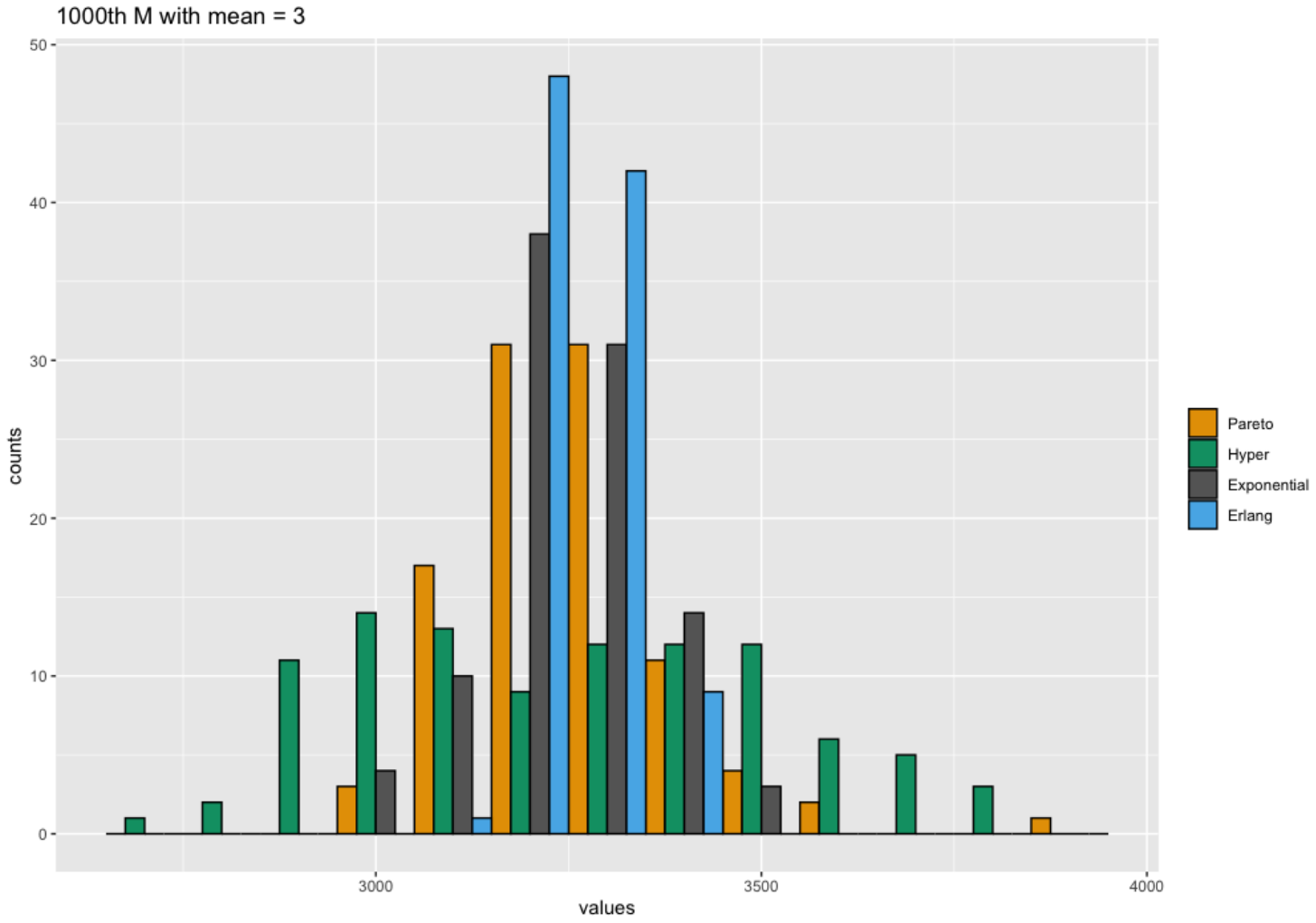


Figure 16: 1000th Step for Maximum Value

The binwidth is set to be 100. Hence, for every interval of length 100 we can observe 4 pillars each representing the number of values from the different distributions in that specific interval.

The distributions with the lowest variances are gathered around 3250, while the high variance distributions cause runs resulting in extreme values to both sides. This is expected as the distribution of the maximum values will be affected by the variance of the jump sizes.

## 2.5 First passage probabilities

A first passage probability is defined as  $\mathbb{P}(\sup_{t \leq 0} X_t > a)$  for some fixed  $a$ . We will start by looking at first passage probabilities for 10  $a$ 's equidistant from 1000 to 3000. The constants  $\mu$ ,  $\sigma$ , and  $\lambda$  are set to  $\mu = 0$ ,  $\sigma = 5$ , and  $\lambda = 1$ . Furthermore, we set the expectation of the jump sizes,  $E[Y_t]$ , to 5 which makes it dominate in the long run. We will simulate  $X$  where  $Y_t$  is following an Exponential, Erlang, Pareto, and hyper-exponential distribution.

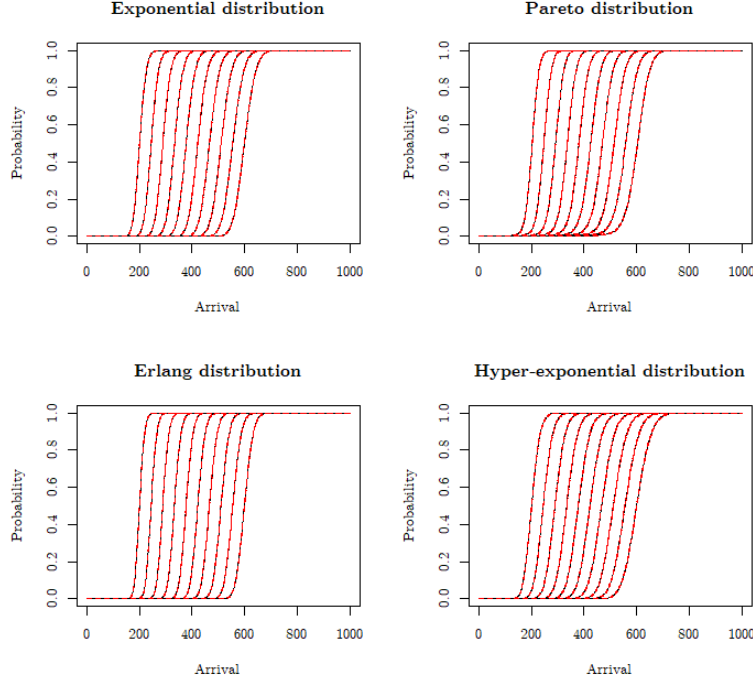


Figure 17: First passage probabilities with equidistant  $a$ 's from 1000 to 3000

For each plot in Figure 17 we have 10 lines where the line furthest to the left represents  $a = 1000$ . From here  $a$  increases with 222.2 for each line towards the right. The red dotted lines are the lower and upper confidence interval for the probability but the variance is so small that we do not see any width. To see the details on how the confidence has been calculated we refer to the code.

In 17 all plots are approximately equal and when keeping the expectation constant at 5, the exponential and Erlang will not be able to differ a lot from what we see in Figure 17. The pareto and hyper-exponential on the other hand can keep the expectation constant at 5 but exhibit very different variances.

In Figure 18 we see the Pareto distribution for different  $k$ 's but all with an expectation of 5. We see how the heavy tail of low  $k$  distributions flattens the slope of each line. This is because most values will be small but once in a while, very large values will be drawn. This can make the process jump above 3000 in one jump and therefore we do not see as neat separation as in low variance distributions.

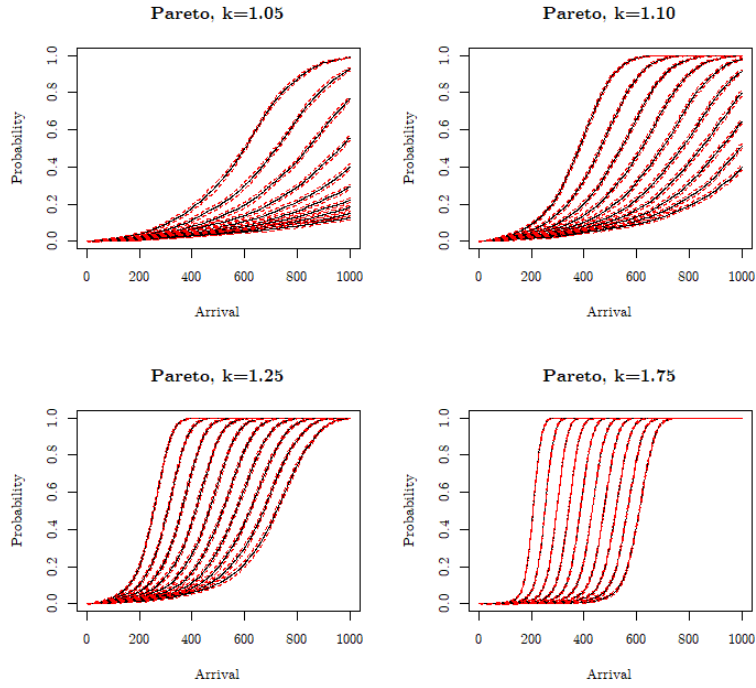


Figure 18: The parameters of the distributions were  $(k=1.05, \beta = 0.2381)$ ,  $(k=1.10, \beta = 0.4545)$ ,  $(k=1.25, \beta = 1)$ , and  $(k=1.75, \beta = 2.1429)$

Next, we will look at the first passage probability for when the hyper exponential exhibits a heavy tail. To make it comparable with the Pareto distribution we found the empirical variance of the Pareto distribution with an expectation of 5 and  $k=1.05$ . We then tuned the parameters of the hyper exponential such it had the same variance and an expectation of 5.

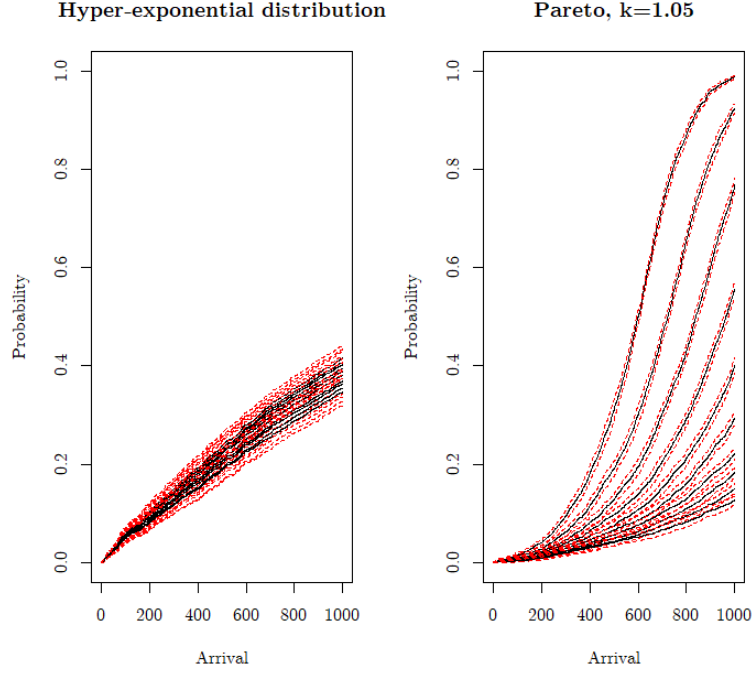


Figure 19: The hyper-exponential had  $p=(0.9994,0.0006)$  and  $\lambda=(20,0.0001177805843)$  and the pareto had  $k=1.05$ ,  $\beta=0.2381$

We see in Figure 19 that the two distributions are very different. The different lines for the values for  $a$  in the pareto plot are well separated indicating that when  $a = 1000$  we have a lot higher first passage probability than for 3000. This is not the case in the hyper exponential and to understand this we will look at the loglog plot over the empirical survival function.

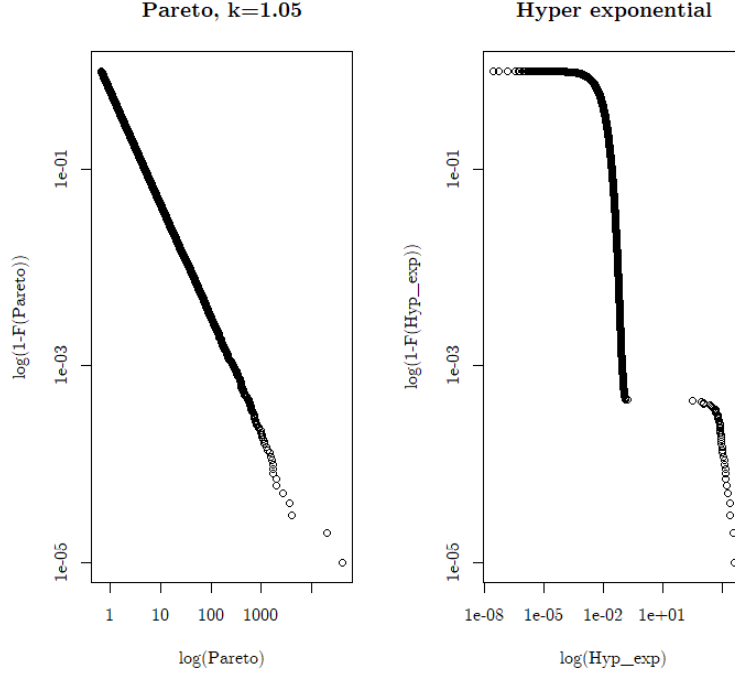


Figure 20: The hyper-exponential had  $p=(0.9994,0.0006)$  and  $\lambda =(20,0.0001177805843)$ , while the Pareto had  $k=1.05$ ,  $\beta =0.2381$

We see that the density decreases "smoothly" for the Pareto distribution meaning we are able to sample the whole range. On the other hand, there is a gap in the probability density for the hyper-exponential. This means that either we sample very small values or very large values. This is the reason we do not see a difference for  $a = 1000$  or  $a = 3000$  because either we only sample small values which will not get us above 1000 or we sample one very large value which will get us above 3000.

Lastly, we will look at the first passage probability for a pure Brownian motion.

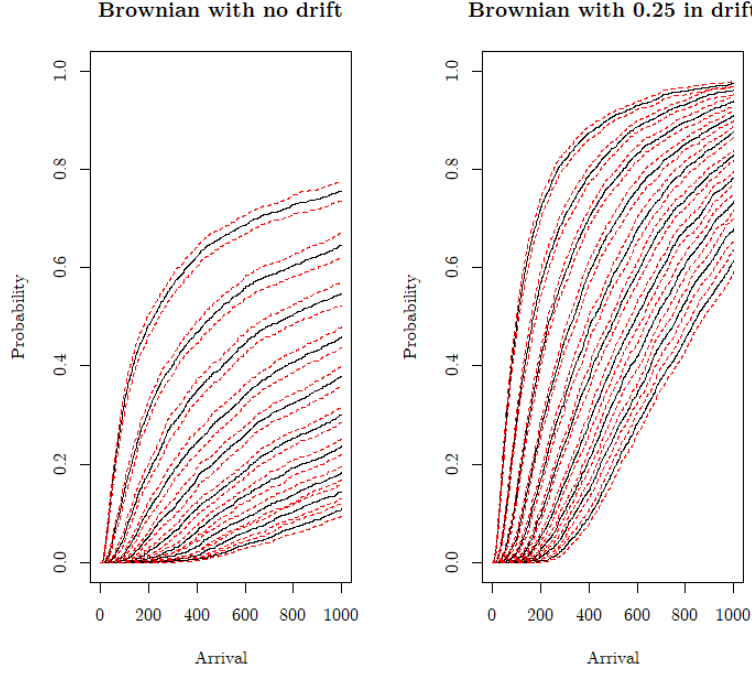


Figure 21: The a lines are from 50 to 250 and the Brownian motion has a  $\sigma = 5$  and the drift can be read from the titles.

In Figure 21 we see two plots where only the Brownian motion is simulated. The jump process was still in the simulation but  $\lambda$  of the underlying exponential distribution was set to 100000 meaning every jump is undetectable.

### 3 European call option

In the following we estimate the price of European call option using a Monte Carlo estimation and the Black-Scholes model. We assume that the price of the option,  $S_t$ , follows a geometric Brownian motion, i.e., we have the stochastic differential equation:

$$dS_t = \mu S_t dt + \sigma S_t dB_t \quad (3)$$

where  $dB_t$  is a standard Brownian motion described in 2.1.3.

All the assumptions associated with geometric Brownian motion still applies, see section 2.1.3.

#### 3.1 Monte Carlo Simulation of Call Option

The evolution of a stock price can be modeled by a geometric Brownian motion which can be seen in equation 3. The stock price can be modeled by the function  $\log(S_t)$  which we can derive by use of Itô's formula given in 4. It is described in greater detail in [2].

$$du = u' dx + \frac{1}{2} u'' dx^2 \quad (4)$$



We apply 4 on 3 with function  $\log S_t$

$$u(x) = \log(x) \quad u'(x) = \frac{1}{x} \quad u''(x) = -\frac{1}{x^2}$$

$$\begin{aligned} d\log(S_t) &= \frac{1}{S_t} dS_t - \frac{1}{2} \frac{1}{S_t^2} dS_t^2 \\ &= \frac{1}{S_t} \mu S_t dt + \sigma S_t dB_t + \frac{1}{2S_t^2} S_t^2 dB_t^2 \sigma^2 + 2S_t^2 dB_t dt \mu \sigma + S_t^2 dt^2 \mu^2 \end{aligned}$$

We know from Itô's lemma that  $dt dB_t = 0$ ,  $dt^2 = 0$  and  $dB_t^2 = dt$ . Proofs can be found in [2]. Therefore, we can reduce to above expression to:

$$\begin{aligned} d\log(S_t) &= \mu dt + \sigma dB_t - \frac{1}{2} \frac{S_t^2}{S_t^2} dt \sigma^2 \\ &= \mu dt + \sigma dB_t - \frac{1}{2} dt \sigma^2 \\ &= (\mu - \frac{1}{2} \sigma^2) dt + \sigma dB_t \end{aligned} \tag{5}$$

We now assume we know the price of the option at time 0 and rewrite equation 5.

$$\begin{aligned} \log S_t &= \log(S_0) + (\mu + \frac{1}{2} \sigma^2) t + \sigma B_t \leftrightarrow \\ S_t &= S_0 e^{(\mu + \frac{1}{2} \sigma^2) t + \sigma B_t} \end{aligned} \tag{6}$$

We will use equation 6 to model a European call option. The definition of such option is given below and taken from [7].

**A European call option** is a contract with the following conditions:

1. The holder of the option has at a prescribed time in the future, the exercise time  $T$ , the right to buy a prescribed stock for a prescribed amount, the exercise price  $K$ .
2. The holder of the option is in no way obliged to buy the underlying asset.
3. The exercise price and date are determined at the time when the option is written.

From [3] we know by use of equation 6 how to value a European call option.

$$\begin{aligned} V &= E[f(S_t)] \\ &= E[e^{-\mu T} (S_T - K)^+] \end{aligned} \tag{7}$$

We are working with a risk neutral pricing framework, defined in (7. *Black-Scholes, p. 2*)[4], and therefore  $\mu$  is the same  $\mu$  as in equation 6 which is the risk free rate. The expression  $e^{-\mu T}$  is the discounting factor.

We will experiment with specific values in section 3.3.

One problem with the crude Monte Carlo estimate is that one needs a huge number of simulations to get a descent confidence interval. One way to tackle this problem is by use of control variate variance reduction method:

### 3.1.1 Option simulation by control variate

The overall idea is still to estimate the mean of the value,  $V$ , using Monte Carlo simulation. However, to reduce the variance, we introduce another random variable,  $Y_i$ , with a known mean,  $E[Y_i]$ , and a constant  $c$ :

$$Z_i = V_i + c(Y_i - E[Y_i])$$

A nice feature is that  $E[Z_i] = E[V_i] + c(E[Y_i] - E[Y_i]) = E[V]$  and we have the variance:

$$Var[Z_i] = Var[V_i] + c^2 Var[Y_i] + 2c Cov[V_i, Y_i] \quad (8)$$

If we saw this variance estimate as a function of  $c$ ,  $h(c)$ , then we have  $h'(c) = 2c Var[Y_i] + 2Cov[V_i, Y_i]$  which can be used to find the minimum  $c^*$ :

$$c^* = \frac{-Cov[V_i, Y_i]}{Var[Y_i]}$$

If we set this into eq. 8, we get the variance:

$$Var[Z_i] = Var[V_i] - \frac{Cov[V_i, Y_i]^2}{Var[Y_i]} \quad (9)$$

We, therefore, have to pick  $Y_i$  so that it has a high correlation with  $V_i$  and thereby high covariance. Further, to hold down the computational complexity, it would also be smart to choose  $Y_i$  so that it might be some part of the simulation already.

In [1] they suggest  $Y = S_T$ .  $S_T$  is modeled by a geometric Brownian motion which means the expectation is given by  $e^{\mu T} S_0$ . We therefore get

$$E[V_{control}] = E[V] + c^*(E[S_T] - e^{\mu T} S_0) \quad (10)$$

We test this in the section 3.3.

## 3.2 Black-Scholes Model

The following is based on (*Black-Scholes Model*, p. 11)[4] and (p. 131)[6].

To compute an analytical value of the price of the call option, we use the Black-Scholes formula. There are multiple ways to derive the formula and we will outline one of them.

To make use of this, we make multiple assumptions. Most importantly, we assume that the stock price,  $S_t$ , follows a geometric Brownian motion as described in section 2.1.3 and seen in equation 3. Further we assume that the interest rate for in some bank account is constant hence the differential of an amount of money,  $M_t$ , in the account at time,  $t$ , is:

$$dM_t = rM_t dt$$

Here we also assume that there are no transaction costs and we can borrow unlimited stocks and lend unlimited amounts of cash. The price,  $V$ , of the option is a function of the time of maturity,  $T$ , the time where we buy the option,  $t$ , and the stock price,  $S_t$ . We assume that  $r$ ,  $\sigma$ , and  $K$  are constant hence we deal with them indirectly with  $S_t$  or include them as a boundary condition for equation 18. For now, we say  $V \sim V(T - t, S_t)$  hence the option price changes with time and the stock price. From Itô's lemma we know that a function applied to a geometric Brownian motion is calculated as, see [5]:

$$dV(S, t) = \left( \mu S_t \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2} \sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2} \right) dt + \sigma S_t \frac{\partial V}{\partial S} dB_t \quad (11)$$

Equation 11 is a SDE where the first term is some drift term followed by a diffusion term. The last term is what connects the option value with the underlying Brownian motion of the stock price. This is what is utilized when a

delta hedge strategy is applied in our portfolio. The idea is to hedge the risk of the option by trading the underlying stock because these two are connected. To make an optimal strategy for a portfolio,  $P$ , i.e., make it self-financing, we have to buy a special amount,  $y_t$ , of the stock. To do that, we fund it using a bank account where we assume that we pay the bank with a interest rate,  $r$ , when we borrow, and that bank will pay the interest rate when we have earned some amount,  $x_t$ , money. The value of our portfolio is therefore:

$$P_t = x_t M_t + y_t S_t \quad (12)$$

Our portfolio will change when the stock price changes. The portfolio gain is calculated under the assumption of self financing strategy hence we have:

$$dP_t = x_t dB_t + y_t dS_t \quad (13)$$

$$= rx_t M_t dt + y_t (\mu S_t dt + \sigma S_t dB_t) \quad (14)$$

$$= (rx_t M_t + y_t \mu S_t) dt + y_t \sigma S_t dB_t \quad (15)$$

This is the SDE of our portfolio and we want this to replicate the value of the option in equation 11. We, therefore, have to equal the terms of the drift and the Brownian motion in equation 11 and equation 15. When we do this, we want to find the number of stocks to buy,  $y_t$  and money to borrow,  $x_t$ :

$$\begin{aligned} \sigma S_t &= \sigma S_t \frac{\partial V}{\partial S} \\ \Rightarrow y_t &= \frac{\partial V}{\partial S} \end{aligned} \quad (16)$$

and

$$\begin{aligned} rx_t M_t + y_t \mu S_t &= \mu S_t \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \\ \Rightarrow rx_t M_t &= \frac{\partial V}{\partial t} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \end{aligned} \quad (17)$$

With the no-arbitrage possibility the price of the option has to be equal our portfolio,  $V_0 = P_0$ . With the chosen values  $x_t$  and  $y_t$ , the call option and our portfolio follows the same dynamics hence  $V_t = P_t$ . We can, therefore, substitute 16 and 17 into our portfolio equation 12. Here we have to remember the extra  $r$  in 17 hence we scale the other terms with  $r$ :

$$\begin{aligned} P_t &= V_t = x_t M_t + y_t S_t \\ \Rightarrow rV_T &= \frac{\partial V}{\partial t} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + r \frac{\partial V}{\partial S} S_t \\ \Rightarrow 0 &= \frac{\partial V}{\partial t} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + r \frac{\partial V}{\partial S} S_t - rV_T \end{aligned} \quad (18)$$

Equation 18 is referred to as the Black-Scholes PDE. Since we deal with European options we have the boundary conditions,  $V(S, T) = (S_T - K)^+$ ,  $V(0, t) = 0$  for all the initial time  $t$  where  $K$  is the strike price at time,  $t$ . Further  $V(S, t) \rightarrow S$  as  $S \rightarrow \infty$ .

The closed-form solution to equation 18 is:

$$\begin{aligned} V(S_t, t) &= S_t \Phi(d_1) - e^{-r(T-t)} K \Phi(d_2) \\ \text{where } d_1 &= \frac{\log(\frac{S_t}{K}) + (r + \sigma^2/2)(T-t)}{\sigma \sqrt{T-t}} \\ \text{and } d_2 &= d_1 - \sigma \sqrt{T-t} \end{aligned}$$

here  $\Phi(\cdot)$  is the CDF of the standard normal distribution. Given that we accept all the assumption about the stock price and option price, we can now use the Black-Scholes formula to compute the European style put option.

### 3.3 European call option experiments

We can now use the crude Monte Carlo estimator to estimate the value of equation 7, further we test variance reducing control variate method, and finally, we verify with the Black-Scholes formula. We use the parameters:

1.  $S_0 = 40$
2.  $K = 40$
3.  $T = 10$
4.  $\mu = 0.05$
5.  $\sigma = 0.3$

**Crude Monte Carlo** We simulate 100000 stocks with the above settings and get a standard deviation of 46.48448 which corresponds to a confidence interval of:

$$E[e^{-\mu T}(S_T - K)^+] = 20.82282 \pm 0.2881138$$

**Control Variate** This gives with same settings as fore the crude estimate a standard deviation of 6.097014 and a confidence interval of

$$E[V_{control}] = 21.04399 \pm 0.03778968 \quad (19)$$

This is great improvement over the crude estimate. In the next section we will estimate the value of the option by use of Black-Scholes Model instead of a simulation approach.

**Black-Scholes** We compare the results with the Black-Scholes formula to see if the Monte-Carlo estimate is correct. With the same parameters we get the European put value of:

$$V_{Black-Scholes} = 21.0267178$$

This value is definitely included in the confidence interval from the result in 19 hence we can conclude that the Monte Carlo method with control variate as variance reduction method proved effective for the chosen parameters.

## 4 Discussion

In the section 3, we simplified the equation so that we assumed that the underling only followed a geometric Brownian motion. This made us able to check the result of the Monte Carlo simulation, however, as stated in (*Black-Scholes Model*, p. 6)[4] and in [9] the Brownian motion and the Black-Scholes is a poor approximation to reality. Here the problem is that it only allows for local fluctuation without the characteristic big jumps in stock prices that might occur due to company announcements or critical information. Therefore, Merton - one of the figures behind the Black-Scholes - suggested to add a jump term that could imitate the non-martingale fluctuations of the stock. Here one can question which distribution the jumps follow, and here we have investigated the Erlang, Exponential, Hyperexponential, and Pareto, whereas Merton used a Gaussian distribution in his paper [9]. It allows Merton to derive some theoretical results, though, this is out of the scope of this course and, therefore, we used the other very different distributions to assess how the behavior affects the simulation. In general, the complexity increases hugely when one wants to derive theoretical results of the jump-diffusion model and therefore we instead

described in detail how changes in parameters affect the behavior of the simulation which could be used for later analysis of stock price modeling and assessment. Further, the risk analysis of a portfolio could be done using a Lévi process, where we use the first passage probability to calculate the risk. The limit for the first probability should be the buy price of the option plus our option premium. However, estimating the parameters for the Lévi process would be a huge task, why this has not been attempted.

## 5 Conclusion

In this project, we described a Lévy process and examined how each component affected the overall behavior. We used a Wiener-Hopf factorization to simulate the process. We found that  $\mu$  affects the overall trend of the model,  $\sigma$ , amplifies the variance of the Brownian motion. When we increase the intensity of the Poisson process,  $\lambda$ , jumps happens more frequently which affects the behavior a lot. Further, the distribution of the jump size affects the 1000th arrival value much. Here hyperexponential distribution gives the greatest difference which is due to the great variance of the distribution. The first passage probabilities are very affected by the distribution as well, most noticeably, the two distributions with heavy tails, Pareto and hyperexponential, are interesting because the two different tails also affect the behaviour differently. The price of the European call option was successfully estimated using Monte Carlo methods and with control variate variance reduction we obtained  $V_{control} = 21.04399 \pm 0.03778968$  which is very close to the estimate of the Black-Scholes of  $V_{Black-Scholes} = 21.0267178$ . With the detailed distribution of the Lévy process, it would be really interesting to see if it indeed is better to mimic real stock prices.

## 6 References

### References

- [1] David Cai. *Monte Carlo Simulation*. [https://www.math.nyu.edu/~cai/Courses/Derivatives/compfin\\_lecture\\_6.pdf](https://www.math.nyu.edu/~cai/Courses/Derivatives/compfin_lecture_6.pdf). 2006.
- [2] Bruce C. Dieffenbach. *Itô's Formula*. [https://www.albany.edu/~bd445/Economics\\_802\\_Financial\\_Economics\\_Slides\\_Fall\\_2013/Ito\\_Formula\\_\(Print\).pdf](https://www.albany.edu/~bd445/Economics_802_Financial_Economics_Slides_Fall_2013/Ito_Formula_(Print).pdf). 2013.
- [3] Mike Giles. *Module 2: Monte Carlo Methods*. [http://people.maths.ox.ac.uk/~gilesm/mc/module\\_2/module\\_2\\_1.pdf](http://people.maths.ox.ac.uk/~gilesm/mc/module_2/module_2_1.pdf).
- [4] Martin Haugh. *Lecture notes for IEOR E4706, Foundations of Financial Engineering, Fall 2016, IE&OR Department, Columbia University*. <https://martin-haugh.github.io/teaching/foundations-fe/>.
- [5] Peter Kempthorne et al. *Lecture notes for 18.S096, Topics in Mathematics with Applications in Finance*. Massachusetts Institute of Technology: MIT OpenCourseWare, <https://ocw.mit.edu/courses/mathematics/18-s096-topics-in-mathematics-with-applications-in-finance-fall-2013/lecture-notes/>. Fall 2013.
- [6] David Lando, Simon Ellersgaard Nielsen, and Rolf Poulsen. *Lecture Notes for Finance 1 (and More)*. 2015.
- [7] Gunilla Linde and Martin Åberg. *What is a European call option?* <http://www.it.uu.se/edu/course/homepage/projektTDB/vt04/projekt2/option.html>. 2004.
- [8] D. J. Manuge. *Lévy Processes For Finance: An Introduction In R*. 2015. arXiv: 1503.03902 [stat.AP].
- [9] Robert C. Merton. "Option pricing when underlying stock returns are discontinuous". In: *Journal of Financial Economics* 3 (1976), pp. 125–144.
- [10] Bo Friis Nielsen. *Simulation of Lévy processes, Stochastic Simulation 02443*. 2020.

## A Derivation of Statistical Moments

### A.1 Basic Properties of Expectations

In order to derive the mean and variance of the different probability distribution we will need some useful properties.

#### A.1.1 Linearity of Expectation

By definition the expectation of a continuous random variable  $X$  is given by:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} X \cdot f_x(x) dx$$

From this expression we can derive linearity.

$$\begin{aligned}\mathbb{E}[aX + bY] &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (aX + bY) \cdot f_{xy}(x, y) dx dy \\ &= a \cdot \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} X f_{xy}(x, y) dx dy + b \cdot \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} Y f_{xy}(x, y) dx dy \\ &= a \cdot \int_{-\infty}^{\infty} X f_x(x) dx + b \cdot \int_{-\infty}^{\infty} Y f_y(y) dy \\ &= a \cdot \mathbb{E}[X] + b \cdot \mathbb{E}[Y]\end{aligned}\tag{20}$$

In the above the joint probability density function is transformed into the marginal distribution function for both random variables. From that it follows that the expectation is linear.

#### A.1.2 Variance

Let the  $\mathbb{E}[X] = \mu$ . Then by expanding the definition of variance we obtain a useful relationship.

$$\begin{aligned}\mathbb{E}[(X - \mu)^2] &= \mathbb{E}[X^2 + \mu^2 - 2 \cdot X \cdot \mu] \\ &= \mathbb{E}[X^2] + \mathbb{E}[\mu^2] - 2 \cdot \mu \cdot \mathbb{E}[X] \\ &= \mathbb{E}[X^2] + \mu^2 - 2 \cdot \mu^2 \\ &= \mathbb{E}[X^2] - \mu^2\end{aligned}\tag{21}$$

In the above stated the linearity of expectation is used.

### A.2 Derivation of Statistical Moments

In the following subsections we wish to derive the statistical moments of interest for the exercise.

#### A.2.1 Pareto Distribution

The probability density function for the Pareto distribution for  $x > \beta$  is as follows:

$$f(x) = k \cdot \frac{\beta^k}{x^{k+1}}$$

Let  $X \sim \text{Pareto}(\beta, k)$  for which  $\beta > x$  and  $k > 1$ . Then

$$\begin{aligned}
\mathbb{E}[X] &= \int_0^\infty x \cdot k \cdot \frac{\beta^k}{x^{k+1}} dx \\
&= \int_0^\infty k \cdot \frac{\beta^k}{x^k} dx \\
&= k \cdot \beta^k \cdot \int_0^\infty \frac{1}{x^k} dx \\
&= k \cdot \beta^k \cdot \left( \frac{x^{-k+1}}{-k+1} \Big|_\beta^\infty \right) \\
&= k \cdot \beta^k \cdot \left( 0 - \frac{\beta^{1-k}}{1-k} \right) \\
&= \frac{k \cdot \beta}{k-1}
\end{aligned}$$

Notice how the limit of the integral "explodes" if  $k < 1$ . That will result in  $\mathbb{E}[X] = \infty$ . In order to find the centralized second moment (i.e. the variance) we will need the second moment as well.

$$\begin{aligned}
\mathbb{E}[X^2] &= \int_\beta^\infty x^2 \cdot k \cdot \frac{\beta^k}{x^{k+1}} dx \\
&= \int_\beta^\infty k \cdot \frac{\beta^k}{x^{k-1}} dx \\
&= k \cdot \beta^k \cdot \int_\beta^\infty \frac{1}{x^{k-1}} dx \\
&= k \cdot \beta^k \cdot \left( \frac{x^{-k+2}}{-k+2} \Big|_\beta^\infty \right) \\
&= k \cdot \beta^k \cdot \left( 0 - \frac{\beta^{2-k}}{2-k} \right) \\
&= \frac{k \cdot \beta^2}{k-2}
\end{aligned}$$

Notice how it is assumed that  $k > 2$ . If  $k \in (1, 2]$  then it can be shown that the variance is infinite, and furthermore that it does not exist for  $k < 1$ . Now we can use the formula for variance which we derived in an above section.

$$\begin{aligned}
\text{Var}[X] &= \mathbb{E}[X^2] - E[X]^2 \\
&= \frac{k \cdot \beta^2}{k-2} - \left( \frac{k \cdot \beta}{k-1} \right)^2 \\
&= \beta^2 \left( \frac{k}{k-2} - \frac{k^2}{(k-1)^2} \right) \\
&= \beta^2 \left( \frac{k \cdot (k-1)^2 - k^2 \cdot (k-2)}{(k-2)(k-1)^2} \right) \\
&= \frac{\beta^2 k}{(k-2)(k-1)^2}
\end{aligned} \tag{22}$$

Using these results we wish to keep a similar expectation for  $Y_i$ , such that it constitutes a better ground for comparison.



### A.2.2 Erlang Distribution

The probability density function for the Erlang distribution for  $\lambda, x \geq 0$  and  $k \in \mathbb{Z}_+$  is as follows:

$$f(x) = \frac{\lambda^k x^{k-1} e^{-\lambda \cdot x}}{(k-1)!}$$

,

Let  $X \sim \text{Erl}(\lambda, k)$ . Then we derive the expectation.

$$\begin{aligned} \mathbb{E}[X] &= \int_0^\infty x \cdot \frac{\lambda^k x^{k-1} e^{-\lambda \cdot x}}{(k-1)!} dx \\ &= \frac{k}{\lambda} \cdot \int_0^\infty \frac{\lambda^{k+1} x^k e^{-\lambda \cdot x}}{k!} dx \\ &= \frac{k}{\lambda} \cdot 1 \\ &= \frac{k}{\lambda} \end{aligned}$$

Since the probability density function is valid for any  $k \in \mathbb{Z}_+$ , we know that the integral will be 1 for  $k+1$  as well. Next we wish to derive the second moment.

$$\begin{aligned} \mathbb{E}[X^2] &= \int_0^\infty x^2 \cdot \frac{\lambda^k x^{k-1} e^{-\lambda \cdot x}}{(k-1)!} dx \\ &= \frac{k \cdot (k+1)}{\lambda^2} \cdot \int_0^\infty \frac{\lambda^{k+2} x^{k+1} e^{-\lambda \cdot x}}{(k+1)!} dx \\ &= \frac{k \cdot (k+1)}{\lambda^2} \cdot 1 \\ &= \frac{k \cdot (k+1)}{\lambda^2} \end{aligned}$$

The same reasoning about the integral is used as before. Then the variance is derived. Finally, we wish to derive the centralized second moment (i.e variance).

$$\begin{aligned}
\text{Var}[X] &= \mathbb{E}[X^2] - E[X]^2 \\
&= \frac{k \cdot (k+1)}{\lambda^2} - \left(\frac{k}{\lambda}\right)^2 \\
&= \frac{k \cdot (k+1)}{\lambda^2} - \frac{k^2}{\lambda^2} \\
&= \frac{k^2 + k - k^2}{\lambda^2} \\
&= \frac{k}{\lambda^2}
\end{aligned}$$

### A.2.3 Exponential Distribution

The probability density function for the exponential distribution for  $\lambda > 0$  and  $x \geq 0$  is as follows:

$$f(x) = \lambda e^{-\lambda \cdot x}$$

Let  $X \sim \text{Exp}(\lambda)$ . Then we derive the expectation.

$$\begin{aligned}
\mathbb{E}[X] &= \int_0^\infty x \cdot \lambda e^{-\lambda \cdot x} dx \\
&= -e^{-\lambda \cdot x} \cdot x - \int -e^{-\lambda \cdot x} dx \Big|_0^\infty \\
&= -e^{-\lambda \cdot x} \cdot x - \frac{1}{\lambda} \cdot e^{-\lambda \cdot x} \Big|_0^\infty \\
&= \lim_{x \rightarrow \infty} (-e^{-\lambda \cdot x} \cdot x - \frac{1}{\lambda} \cdot e^{-\lambda \cdot x}) - (-e^{-\lambda \cdot 0} \cdot 0 - \frac{1}{\lambda} \cdot e^{-\lambda \cdot 0}) \\
&= 0 - \left(-\frac{1}{\lambda}\right) \\
&= \frac{1}{\lambda}
\end{aligned}$$

The limit above is found under the assumption of the initially stated lower bounds for  $\lambda$  and  $x$ . Then we find the second moment.

$$\begin{aligned}
\mathbb{E}[X^2] &= \int_0^\infty x^2 \cdot \lambda e^{-\lambda \cdot x} dx \\
&= -e^{-\lambda \cdot x} \cdot x^2 - 2 \int -e^{-\lambda \cdot x} \cdot x dx \Big|_0^\infty \\
&= -e^{-\lambda \cdot x} \cdot x^2 - 2 \cdot \left( x \cdot \frac{1}{\lambda} \cdot e^{-\lambda x} + \frac{1}{\lambda^2} e^{-\lambda x} \right) \Big|_0^\infty \\
&= \lim_{x \rightarrow \infty} \left( -e^{-\lambda \cdot x} \cdot x^2 - 2 \cdot \left( x \cdot \frac{1}{\lambda} \cdot e^{-\lambda x} + \frac{1}{\lambda^2} e^{-\lambda x} \right) \right) - \left( -e^{-\lambda \cdot 0} \cdot 0^2 - 2 \cdot \left( 0 \cdot \frac{1}{\lambda} \cdot e^{-\lambda 0} + \frac{1}{\lambda^2} e^{-\lambda 0} \right) \right) \\
&= 0 - \left( -2 \cdot \frac{1}{\lambda^2} \right) \\
&= \frac{2}{\lambda^2}
\end{aligned}$$

The same reasoning about the integral is used as before. Then the variance is derived. Finally, we wish to derive the centralized second moment (i.e variance).

$$\begin{aligned}
\text{Var}[X] &= \mathbb{E}[X^2] - E[X]^2 \\
&= \frac{2}{\lambda^2} - \left( \frac{1}{\lambda} \right)^2 \\
&= \frac{1}{\lambda^2}
\end{aligned} \tag{23}$$

#### A.2.4 Hyperexponential Distribution

The probability density function for the hyperexponential distribution is a weighted sum of exponentially distributed variables. Let  $\lambda_i > 0$  and  $x \geq 0$ .

$$f(x) = \sum_{i=1}^n p_i \lambda_i e^{-\lambda_i \cdot x}$$

Let  $X \sim \text{Hyperexp}(\{\lambda_i, p_i\}_{i=1..n})$ . Then the expectation is as follows:

$$\begin{aligned}
\mathbb{E}[X] &= \int_{-\infty}^\infty x \cdot f(x) dx \\
&= \int_0^\infty x \cdot \sum_{i=1}^n p_i \lambda_i e^{-\lambda_i x} dx \\
&= \sum_{i=1}^n p_i \int_0^\infty x \cdot \lambda_i e^{-\lambda_i x} dx \\
&= \sum_{i=1}^n \frac{p_i}{\lambda_i}
\end{aligned}$$

The limit above is found under the assumption of the initially stated lower bounds for  $\lambda$  and  $x$ . Then we find the second moment.

$$\begin{aligned}
\mathbb{E}[X^2] &= \int_{-\infty}^{\infty} x^2 \cdot f(x) \, dx \\
&= \int_0^{\infty} x^2 \cdot \sum_{i=1}^n p_i \lambda_i e^{-\lambda_i x} \, dx \\
&= \sum_{i=1}^n p_i \int_0^{\infty} x^2 \cdot \lambda_i e^{-\lambda_i x} \, dx \\
&= \sum_{i=1}^n p_i \frac{2}{\lambda_i^2}
\end{aligned}$$

The same reasoning about the integral is used as before. Then the variance is derived. Finally, we wish to derive the centralized second moment (i.e variance).

$$\begin{aligned}
\text{Var}[X] &= \mathbb{E}[X^2] - E[X]^2 \\
&= \sum_{i=1}^n p_i \frac{2}{\lambda_i^2} - \left( \sum_{i=1}^n \frac{p_i}{\lambda_i} \right)^2
\end{aligned} \tag{24}$$

This concludes the moments of interest, and we will in the following section use these to investigate the meaning of  $Y_i$ .

## B Code

### B.0.1 Explanation of the algorithm

```

1  rV = function(n, mu, sigma, lambda){
2    phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
3    return(rexp(n, phi1))
4  }
5
6  rW = function(n, mu, sigma, lambda){
7    phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
8    return(rexp(n, phi2))
9  }
10
11 # Illustrating P, A, and M
12 par(mfrow = c(1,1))
13 iter = 5
14 lambda = 1/20
15 sigma = 0.01
16 mu = 0.01
17 lambda_Y = 0.1
18 time = c(0,cumsum(rexp(iter, lambda)))
19 W = rW(iter, mu, sigma, lambda)
20 V = rV(iter, mu, sigma, lambda)

```

```

21
22
23 P = c(0)
24 A = c(0)
25 M = c(0)
26
27 Y = c()
28
29
30 for (i in 1:iter){
31   Y = c(Y, rexp(1, lambda_Y))
32   P = c(P, V[i]-W[i]+A[i])
33   A = c(A, A[i]+V[i]-W[i]+Y[i])
34   M = c(M, max(M[i], A[i]+V[i], A[i+1]))
35 }
36
37 index = seq(0, iter, length.out = iter+1)
38 plot( time, A, type = "s", col = "green", main = substitute(paste(mu, "=", mu_num, ", ", sigma, "=",
39   sigma_num, ", ", lambda, "=", lambda_num ), list(lambda_num = lambda, sigma_num = sigma, mu_num =
40   mu)))
41 lines( time, P, type = "s", col = "red")
42 lines( time, M, type = "s", col = "black", lty = 2)
43 legend("topleft", legend=c("After", "Prior", "Max"),
44   col=c("green", "red", "black"), lty=c(1,1,2), cex=0.8)

```

## B.0.2 Explanation of the constants

```

1
2 library(ggplot2)
3 library(reshape2)
4
5 cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
6   "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")
7
8 rV = function(n, mu, sigma, lambda){
9   phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
10  return(rexp(n, phi1))
11 }
12
13 rW = function(n, mu, sigma, lambda){
14   phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
15   return(rexp(n, phi2))
16 }
17
18 jumpDiffusionSimYexp = function(mu, sigma, lambda, Tn, lambdaY, n){
19   A = matrix(nrow = Tn, ncol = n)
20   P = matrix(nrow = Tn, ncol = n)
21   M = matrix(nrow = Tn, ncol = n)
22
23   for(j in 1:n){
24
25     W = rW(Tn, mu, sigma, lambda)
26     V = rV(Tn, mu, sigma, lambda)
27
28     P[1,j] = 0
29     A[1,j] = 0
30     M[1,j] = 0
31     Mm1 = 0
32     Am1 = 0
33
34

```

```

35     for (i in 1:Tn){
36       Y = rnorm(1, 0,1)
37       P[i,j] = V[i]-W[i]+Aml
38       A[i,j] = Aml+V[i]-W[i]+Y
39       Mml = max(Mml, Aml+V[i], A[i,j])
40       M[i,j] = Mml
41       Aml = A[i,j]
42     }
43   }
44   return(list(P,A,M))
45 }
46
47 set.seed(23)
48 mu = c(0.1,0.5,1,2,5,10)
49 Pdf = data.frame()
50 Adf = data.frame()
51 Mdf = data.frame()
52 Tn = 1000
53 n=100
54 for (i in 1:length(mu)){
55
56   X = jumpDiffusionSimYexp(mu[i], 1, 1, Tn, 2,n)
57
58   Pdf <- rbind(Pdf,
59               data.frame(dataset=(paste("P", mu[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:Tn))
60               )
61   Adf <- rbind(Adf,
62               data.frame(dataset=(paste("A", mu[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:Tn))
63               )
64   Mdf <- rbind(Mdf,
65               data.frame(dataset=(paste("M", mu[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:Tn))
66               )
67   }
68   Adf$dataset <- as.factor(Adf$dataset)
69
70   Mulabs = c()
71   for (i in 1:length(mu)){
72     Mulabs = c(Mulabs, bquote(mu==.(mu[i])))
73   }
74   muLabel = "\u03bc Value"
75   # $
76
77   ggplot(Adf, aes(x=TimeStep)) +
78     geom_line(aes(y=obs, color=dataset))+
79     scale_color_manual(values=c(cbP[1],cbP[3],cbP[10],cbP[5],cbP[6],cbP[8]),
80                        labels=Mulabs)+
81     labs(color=muLabel)+ylab("mean A value")
82   # $
83
84   ##### 3_2 #####
85
86   library(ggplot2)
87   library(reshape2)
88
89   cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
90           "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")
91
92   rV = function(n, mu, sigma, lambda){
93     phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
94     return(rexp(n, phi1))
95   }
96
97   rW = function(n, mu, sigma, lambda){
98     phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)

```

```

97   return(rexp(n, phi2))
98 }
99
100 jump_diffusion_sim_Yexp = function(mu, sigma, lambda, Tn, lambdaY, n){
101   A = matrix(nrow = Tn, ncol = n)
102   P = matrix(nrow = Tn, ncol = n)
103   M = matrix(nrow = Tn, ncol = n)
104
105   for(j in 1:n){
106
107     W = rW(Tn, mu, sigma, lambda)
108     V = rV(Tn, mu, sigma, lambda)
109
110     P[1,j] = 0
111     A[1,j] = 0
112     M[1,j] = 0
113     Mml = 0
114     Aml = 0
115
116     for (i in 1:Tn){
117       Y = rnorm(1, 0,1)
118       P[i,j] = V[i]-W[i]+Aml
119       A[i,j] = Aml+V[i]-W[i]+Y
120       Mml = max(Mml, Aml+V[i], A[i,j])
121       M[i,j] = Mml
122       Aml = A[i,j]
123     }
124   }
125   return(list(P,A,M))
126 }
127
128
129 set.seed(23)
130 sigma = c(1,10,25,50,100,500)
131 Pdf = data.frame()
132 Adf = data.frame()
133 Mdf = data.frame()
134 Tn = 1000
135 n=1000
136 for (i in 1:length(sigma)){
137
138   X = jump_diffusion_sim_Yexp(1, sigma[i], 1, Tn, 1,n)
139
140   Pdf <- rbind(Pdf,
141                data.frame(dataset=(paste("P", mu[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:Tn))
142                )
143   Adf <- rbind(Adf,
144                data.frame(dataset=(paste("A", mu[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:Tn))
145                )
146   Mdf <- rbind(Mdf,
147                data.frame(dataset=(paste("M", mu[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:Tn))
148                )
149 }
150 Mdf$dataset <- as.factor(Mdf$dataset)
151 # $
152 Sigmalabs = c()
153 for (i in 1:length(sigma)){
154   Sigmalabs = c(Sigmalabs,bquote(sigma==.(sigma[i])))
155 }
156 sigmaLabel = "\u03c3 Value"
157 # $
158 ggplot(Mdf, aes(x=TimeStep)) +
159   geom_line(aes(y=obs, color=dataset))+
160   scale_color_manual(values=c(cbP[10],cbP[3],cbP[5],cbP[6],cbP[7],cbP[8]),
161                      labels=Sigmalabs)+

```

```

159 labs(color=sigmaLabel)+
160 ylab("mean M value")
161
162 ##### 3_3 #####
163
164 library(ggplot2)
165 library(reshape2)
166
167 cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
168         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")
169
170 rV = function(n, mu, sigma, lambda){
171   phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
172   return(rexp(n, phi1))
173 }
174
175 rW = function(n, mu, sigma, lambda){
176   phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
177   return(rexp(n, phi2))
178 }
179
180 jumpDiffusionSimYexp = function(mu, sigma, lambda, Tn, lambdaY, n){
181   A = matrix(nrow = Tn, ncol = n)
182   P = matrix(nrow = Tn, ncol = n)
183   M = matrix(nrow = Tn, ncol = n)
184
185   for(j in 1:n){
186
187     W = rW(Tn, mu, sigma, lambda)
188     V = rV(Tn, mu, sigma, lambda)
189
190     P[1,j] = 0
191     A[1,j] = 0
192     M[1,j] = 0
193     Mm1 = 0
194     Aml = 0
195     l714V
196
197     for (i in 1:Tn){
198       Y = rnorm(1, 0,1)
199       P[i,j] = V[i]-W[i]+Aml
200       A[i,j] = Aml+V[i]-W[i]+Y
201       Mm1 = max(Mm1, Aml+V[i], A[i,j])
202       M[i,j] = Mm1
203       Aml = A[i,j]
204     }
205   }
206   return(list(P,A,M))
207 }
208
209 set.seed(23)
210 lambda = c(0.1,0.5,1,2,5,10)
211 Pdf = data.frame()
212 Adf = data.frame()
213 Mdf = data.frame()
214 Tn = 1000
215 n=1000
216 for (i in 1:length(lambda)){
217
218   X = jumpDiffusionSimYexp(1, 1, lambda[i], Tn, 1,n)
219
220   Pdf <- rbind(Pdf,
221               data.frame(dataset=(paste("P", lambda[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:
222               Tn)))
223   Adf <- rbind(Adf,

```



```

223     data.frame(dataset=(paste("A", lambda[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:
      Tn)))
224 Mdf <- rbind(Mdf,
225     data.frame(dataset=(paste("M", lambda[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:
      Tn)))
226 }
227 Adf$dataset <- as.factor(Adf$dataset)
228 # $
229 Lambdalabs = c()
230 for (i in 1:length(lambda)){
231   Lambdalabs = c(Lambdalabs, bquote(lambda==.(lambda[i])))
232 }
233 lamdaLabel = "\u03bb Value"
234 # $
235 ggplot(Adf, aes(x=TimeStep)) +
236   geom_line(aes(y=obs, color=dataset))+
237   scale_color_manual(values=c(cbP[1], cbP[3], cbP[10], cbP[5], cbP[6], cbP[8]),
238     labels=Lambdalabs)+
239   labs(color=lamdaLabel)
240 library(ggplot2)
241 library(reshape2)
242
243 cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
244   "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")
245
246 rV = function(n, mu, sigma, lambda){
247   phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
248   return(rexp(n, phi1))
249 }
250
251 rW = function(n, mu, sigma, lambda){
252   phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
253   return(rexp(n, phi2))
254 }
255
256 jumpDiffusionSimYexp = function(mu, sigma, lambda, Tn, lambdaY, n){
257   A = matrix(nrow = Tn, ncol = n)
258   P = matrix(nrow = Tn, ncol = n)
259   M = matrix(nrow = Tn, ncol = n)
260
261   for(j in 1:n){
262
263     W = rW(Tn, mu, sigma, lambda)
264     V = rV(Tn, mu, sigma, lambda)
265
266     P[1, j] = 0
267     A[1, j] = 0
268     M[1, j] = 0
269     Mml = 0
270     Aml = 0
271
272     for (i in 1:Tn){
273       Y = rnorm(1, 0, 1)
274       P[i, j] = V[i]-W[i]+Aml
275       A[i, j] = Aml+V[i]-W[i]+Y
276       Mml = max(Mml, Aml+V[i], A[i, j])
277       M[i, j] = Mml
278       Aml = A[i, j]
279     }
280   }
281 }
282 return(list(P, A, M))
283 }
284
285 set.seed(23)

```

```

286 mu = c(0.1,0.5,1,2,5,10)
287 Pdf = data.frame()
288 Adf = data.frame()
289 Mdf = data.frame()
290 Tn = 1000
291 n=100
292 for (i in 1:length(mu)){
293
294   X = jumpDiffusionSimYexp(mu[i], 1, 1, Tn, 2,n)
295
296   Pdf <- rbind(Pdf,
297               data.frame(dataset=(paste("P", mu[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:Tn))
298               )
299   Adf <- rbind(Adf,
300               data.frame(dataset=(paste("A", mu[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:Tn))
301               )
302   Mdf <- rbind(Mdf,
303               data.frame(dataset=(paste("M", mu[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:Tn))
304               )
305 }
306 Adf$dataset <- as.factor(Adf$dataset)
307
308 Mulabs = c()
309 for (i in 1:length(mu)){
310   Mulabs = c(Mulabs, bquote(mu==.(mu[i])))
311 }
312 muLabel = "\u03bc Value"
313 # $
314
315 ggplot(Adf, aes(x=TimeStep)) +
316   geom_line(aes(y=obs, color=dataset))+
317   scale_color_manual(values=c(cbP[1],cbP[3],cbP[10],cbP[5],cbP[6],cbP[8]),
318                     labels=Mulabs)+
319   labs(color=muLabel)+ylab("mean A value")
320 # $
321
322 ##### 3_2 #####
323
324 library(ggplot2)
325 library(reshape2)
326
327 cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
328         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")
329
330 rV = function(n, mu, sigma, lambda){
331   phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
332   return(rexp(n, phi1))
333 }
334
335 rW = function(n, mu, sigma, lambda){
336   phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
337   return(rexp(n, phi2))
338 }
339
340 jump_diffusion_sim_Yexp = function(mu, sigma, lambda, Tn, lambdaY, n){
341   A = matrix(nrow = Tn, ncol = n)
342   P = matrix(nrow = Tn, ncol = n)
343   M = matrix(nrow = Tn, ncol = n)
344
345   for(j in 1:n){
346     W = rW(Tn, mu, sigma, lambda)
347     V = rV(Tn, mu, sigma, lambda)

```

```

348   P[1,j] = 0
349   A[1,j] = 0
350   M[1,j] = 0
351   Mm1 = 0
352   Aml = 0
353
354
355   for (i in 1:Tn){
356     Y = rnorm(1, 0,1)
357     P[i,j] = V[i]-W[i]+Aml
358     A[i,j] = Aml+V[i]-W[i]+Y
359     Mm1 = max(Mm1, Aml+V[i], A[i,j])
360     M[i,j] = Mm1
361     Aml = A[i,j]
362   }
363 }
364 return(list(P,A,M))
365 }
366
367 set.seed(23)
368 sigma = c(1,10,25,50,100,500)
369 Pdf = data.frame()
370 Adf = data.frame()
371 Mdf = data.frame()
372 Tn = 1000
373 n=1000
374 for (i in 1:length(sigma)){
375
376   X = jump_diffusion_sim_Yexp(1, sigma[i], 1, Tn, 1,n)
377
378   Pdf <- rbind(Pdf,
379               data.frame(dataset=(paste("P", mu[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:Tn))
380               )
381   Adf <- rbind(Adf,
382               data.frame(dataset=(paste("A", mu[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:Tn))
383               )
384   Mdf <- rbind(Mdf,
385               data.frame(dataset=(paste("M", mu[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:Tn))
386               )
387 }
388 Mdf$dataset <- as.factor(Mdf$dataset)
389 # $
390 Sigmalabs = c()
391 for (i in 1:length(sigma)){
392   Sigmalabs = c(Sigmalabs, bquote(sigma==.(sigma[i])))
393 }
394 sigmaLabel = "\u03c3 Value"
395 # $
396 ggplot(Mdf, aes(x=TimeStep)) +
397   geom_line(aes(y=obs, color=dataset))+
398   scale_color_manual(values=c(cbP[10],cbP[3],cbP[5],cbP[6],cbP[7],cbP[8]),
399                       labels=Sigmalabs)+
400   labs(color=sigmaLabel)+
401   ylab("mean M value")
402
403 ##### 3-3 #####
404
405 library(ggplot2)
406 library(reshape2)
407
408 cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
409         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")
410
411 rV = function(n, mu, sigma, lambda){
412   phil = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)

```

```

410 |   return(rexp(n, phi1))
411 | }
412 |
413 | rW = function(n, mu, sigma, lambda){
414 |   phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
415 |   return(rexp(n, phi2))
416 | }
417 |
418 | jumpDiffusionSimYexp = function(mu, sigma, lambda, Tn, lambdaY, n){
419 |   A = matrix(nrow = Tn, ncol = n)
420 |   P = matrix(nrow = Tn, ncol = n)
421 |   M = matrix(nrow = Tn, ncol = n)
422 |
423 |   for(j in 1:n){
424 |
425 |     W = rW(Tn, mu, sigma, lambda)
426 |     V = rV(Tn, mu, sigma, lambda)
427 |
428 |     P[1,j] = 0
429 |     A[1,j] = 0
430 |     M[1,j] = 0
431 |     Mml = 0
432 |     Aml = 0
433 |     l714V
434 |
435 |     for (i in 1:Tn){
436 |       Y = rnorm(1, 0,1)
437 |       P[i,j] = V[i]-W[i]+Aml
438 |       A[i,j] = Aml+V[i]-W[i]+Y
439 |       Mml = max(Mml, Aml+V[i], A[i,j])
440 |       M[i,j] = Mml
441 |       Aml = A[i,j]
442 |     }
443 |   }
444 |   return(list(P,A,M))
445 | }
446 |
447 | set.seed(23)
448 | lambda = c(0.1,0.5,1,2,5,10)
449 | Pdf = data.frame()
450 | Adf = data.frame()
451 | Mdf = data.frame()
452 | Tn = 1000
453 | n=1000
454 | for (i in 1:length(lambda)){
455 |
456 |   X = jumpDiffusionSimYexp(1, 1, lambda[i], Tn, 1,n)
457 |
458 |   Pdf <- rbind(Pdf,
459 |               data.frame(dataset=(paste("P", lambda[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:
460 |               Tn)))
461 |   Adf <- rbind(Adf,
462 |               data.frame(dataset=(paste("A", lambda[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:
463 |               Tn)))
464 |   Mdf <- rbind(Mdf,
465 |               data.frame(dataset=(paste("M", lambda[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:
466 |               Tn)))
467 | }
468 | Adf$dataset <- as.factor(Adf$dataset)
469 | # $
470 | Lambdalabs = c()
471 | for (i in 1:length(lambda)){
472 |   Lambdalabs = c(Lambdalabs, bquote(lambda==.(lambda[i])))
473 | }
474 | lamdaLabel = "\u03bb Value"

```

```

472 # $
473 ggplot(Adf, aes(x=TimeStep)) +
474   geom_line(aes(y=obs, color=dataset))+
475   scale_color_manual(values=c(cbP[1], cbP[3], cbP[10], cbP[5], cbP[6], cbP[8]),
476                       labels=Lambdalabs)+
477   labs(color=lamdaLabel)

```

### B.0.3 Jump size distribution

```

1  # IMPORTS
2
3
4  library(ggplot2)
5
6  # ----- PROJECT : Levy Processes [IMPLEMENTATION] -----
7
8  # ----- ##### EXERCISE 4 ##### -----
9
10
11 # ----- ##### COLOR SCHEME ##### -----
12
13 cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
14         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")
15
16 # ----- ##### DISTRIBUTIONS ##### -----
17
18 ##### PDF's #####
19
20 pareto_pdf <- function(beta = 1, k = 1, x){
21   return(k * ((beta^k)/(x^(k + 1))))
22 }
23
24 exp_pdf <- function(lambda = 1/2, x){
25   return(lambda*exp(-lambda*x))
26 }
27
28 erlang_pdf <- function(k = 1, lambda = 1/2, x){
29
30   numerator <- (lambda^k) * (x^(k - 1)) * exp(-lambda*x)
31   denominator <- factorial(k - 1)
32   return(numerator/denominator)
33 }
34
35
36 hyperexp_pdf <- function(p1 = 0.8, lambda1 = 3/2, p2 = 0.2, lambda2 = 2/3, x){
37
38   term1 <- p1 * lambda1 * exp(-lambda1 * x)
39   term2 <- p2 * lambda2 * exp(-lambda2 * x)
40   return(term1 + term2)
41 }
42
43
44 ### GENERATORS #####
45
46 ##### GENERATORS #####
47
48
49 # 1. ----- Pareto Distribution -----
50
51 # Mean:
52 # Variance:

```

```

53 |
54 | # Description:
55 |
56 | gen_pareto <- function(n = 1000, beta = 1, k = 2.05){
57 |
58 |   U <- runif(n)
59 |   X <- beta * U^(-1/k)
60 |   return(X)
61 | }
62 |
63 | # 2. ----- Normal Distribution -----
64 |
65 | # Mean: mu
66 | # Variance: sigma^2
67 |
68 | # Description:
69 |
70 | gen_normal <- function(n = 1, mean = 0, sd = 1) {
71 |
72 |   return(rnorm(n, mean = mean, sd = sd))
73 |
74 | }
75 |
76 |
77 | # 3. ----- Exponential -----
78 |
79 | gen_exp <- function(n = 1, rate = 1){
80 |
81 |   return(rexp(n, rate = rate))
82 |
83 | }
84 |
85 | # 4. ----- Hyperexponential -----
86 |
87 | gen_hyperexp <- function(n = 1, rate1 = 1, p1 = 1/2, rate2 = 1, p2 = 1/2){
88 |
89 |   output <- c()
90 |
91 |   for (i in 1:n) {
92 |
93 |     U_1 <- runif(1, 0, 1)
94 |
95 |     if (U_1 <= p1) {
96 |
97 |       output[i] <- rexp(1, rate = rate1)
98 |
99 |     } else {
100 |
101 |       output[i] <- rexp(1, rate = rate2)
102 |
103 |     }
104 |   }
105 |
106 |   return(output)
107 | }
108 |
109 |
110 | # 5. ----- Erlang -----
111 |
112 | gen_erlang = function(n = 1, lambda = 3, k = 2){
113 |
114 |   output = rep(0, n)
115 |
116 |   for (i in 1:k) {
117 |     output = output + rexp(n, lambda)

```

```

118 }
119
120 return(output)
121
122 }
123
124 ##### OTHER IMPLEMENTATION NEEDED #####
125
126 set.seed(23)
127
128 generate_V <- function(n = 1000, mu = 1, sigma = 0.2, lambda = 2){
129
130   theta_1 <- (-1)*(mu/(sigma^2)) + sqrt((mu^2/(sigma^4)) + ((2*lambda)/(sigma^2)))
131   V <- rexp(n, rate = theta_1)
132   return(V)
133 }
134
135
136 generate_W <- function(n = 1000, mu = 1, sigma = 0.2, lambda = 2){
137
138   theta_2 <- (mu/(sigma^2)) + sqrt((mu^2/(sigma^4)) + ((2*lambda)/(sigma^2)))
139   W <- rexp(n, rate = theta_2)
140   return(W)
141 }
142
143
144 generate_Y <- function(n = 1000, distribution = "pareto", set_mean = 3) {
145
146   if (distribution == "pareto"){
147
148     k = 2.05
149
150     #Determines the mean
151     beta = set_mean * (k - 1) / k
152
153     return(gen_pareto(n = n, beta = beta, k = k))
154   }
155
156   if (distribution == "exponential"){
157
158     # Determines the mean
159     rate = 1/(set_mean)
160
161     return(gen_exp(n = n, rate = rate))
162   }
163
164   if (distribution == "hyperexponential") {
165
166     p_1 = 0.8
167     p_2 = (1 - p_1)
168     rate_1 = set_mean
169     # Determines the mean
170     rate_2 = (1 - p_1) / (set_mean - p_1*(1/rate_1))
171
172     return(gen_hyperexp(n = n, rate1 = rate_1, p1 = p_1, rate2 = rate_2, p2 = p_2))
173   }
174
175   if (distribution == "erlang"){
176
177     k <- 3
178     lambda <- k / set_mean
179
180
181
182

```

```

183   # Determines the mean
184
185   return(gen_erlang(n = n, lambda = lambda, k = k))
186
187 }
188
189 }
190
191 algorithm_1 <- function(jumps = 1000, distribution_y = "normal", mean_y = 3){
192
193   V <- generate_V(n = (jumps + 1), mu = 1, sigma = 0.2, lambda = 2)
194   W <- generate_W(n = (jumps + 1), mu = 1, sigma = 0.2, lambda = 2)
195   Y <- generate_Y(n = (jumps + 1), distribution = distribution_y, set_mean = mean_y)
196
197   P <- rep(0, (jumps + 1))
198   A <- rep(0, (jumps + 1))
199   M <- rep(0, (jumps + 1))
200   time <- c(0: jumps)
201
202   for (j in 1:jumps){
203
204     i <- j + 1 # Displace s.t. the indexing looks like in the algorithm
205
206     P[i] <- A[i - 1] + (V[i] - W[i])
207     A[i] <- A[i - 1] + (V[i] - W[i]) + Y[i]
208     M[i] <- max(c(M[i - 1], A[i - 1] + V[i], A[i - 1] + (V[i] - W[i]) + Y[i]))
209
210   }
211
212   result <- data.frame(time, P, A, M)
213
214 }
215
216 algorithm_1_extended <- function(jumps = 1000, repeats = 100, distribution_y = "normal", mean_y =
217   3){
218
219   P_n <- c()
220   A_n <- c()
221   M_n <- c()
222   Y_mean <- c()
223
224   for (k in 1:repeats) {
225
226     P <- rep(0, jumps)
227     A <- rep(0, jumps)
228     M <- rep(0, jumps)
229
230     V <- generate_V(n = (jumps + 1), mu = 1)
231     W <- generate_W(n = (jumps + 1), mu = 1)
232     Y <- generate_Y(n = (jumps + 1), distribution = distribution_y, set_mean = mean_y)
233
234     for (j in 1:jumps){
235
236       i <- j + 1 # Displace s.t. the indexing looks like in the algorithm
237
238       P[i] <- A[i - 1] + (V[i] - W[i])
239       A[i] <- A[i - 1] + (V[i] - W[i]) + Y[i]
240       M[i] <- max(c(M[i - 1], A[i - 1] + V[i], A[i - 1] + (V[i] - W[i]) + Y[i]))
241
242     }
243
244     P <- P[-1] # Remove P_0 = 0
245     A <- A[-1] # Remove A_0 = 0
246     M <- M[-1] # Remove M_0 = 0

```



```

247     P_n[k] <- P[jumps]
248     A_n[k] <- A[jumps]
249     M_n[k] <- M[jumps]
250     Y_mean[k] <- mean(Y)
251
252 }
253
254 return(data.frame(P_n, A_n, M_n, Y_mean))
255
256 }
257
258 # ----- PROJECT : Levy Processes [ANALYSIS] -----
259
260
261 # ----- ##### EXERCISE 4 ##### -----
262
263 set.seed(23)
264 SET_MEAN <- 3
265
266 ### 1) Plotting Distributions
267
268 ### OVERALL #####
269
270 ### DEFINE DATA-POINTS #####
271
272 # We ensure that they all have mean mu = 5
273
274 x <- seq(0, 15, 0.1)
275
276 set_mean <- SET_MEAN
277
278 # PARETO
279
280 pareto_k <- 2.05
281 pareto_beta <- set_mean * (pareto_k - 1) / pareto_k
282 y_pareto <- pareto.pdf(beta = pareto_beta, k = pareto_k, x)
283
284 variance_pareto = (pareto_k * pareto_beta^2) / ((pareto_k - 2) * (pareto_k - 1)^2)
285 print(variance_pareto)
286
287 # EXP
288
289 exp_lambda <- 1 / set_mean
290 y_exp <- exp.pdf(lambda = exp_lambda, x)
291
292 variance_exp <- 1 / exp_lambda^2
293 print(variance_exp)
294
295 # ERLANG
296
297 erlang_k = 3
298 erlang_lambda <- k / set_mean
299 y_erl <- erlang.pdf(k = erlang_k, lambda = erlang_lambda, x)
300
301 variance_erlang <- erlang_k / erlang_lambda^2
302 print(variance_erlang)
303
304 # HYPER
305
306 exp_p1 = 0.8
307 exp_p2 = (1 - p_1)
308 exp_rate_1 = SET_MEAN
309 exp_rate_2 = (1 - p_1) / (set_mean - p_1*(1/rate_1))
310 y_hypexp <- hyperexp.pdf(p1 = exp_p1, lambda1 = exp_rate_1, p2 = exp_p2, lambda2 = exp_rate_2, x)
311

```

```

312 df <- data.frame(x, y_pareto, y_exp, y_erl, y_hypexp)
313
314 p <- ggplot(data = df, mapping = aes(x = x)) +
315   geom_line(mapping=aes(y=y_pareto, color = "Pareto")) +
316   geom_line(mapping=aes(y=y_exp, color = "Exp")) +
317   geom_line(mapping=aes(y=y_erl, color = "Erlang")) +
318   geom_line(mapping=aes(y=y_hypexp, color = "Hypexp")) +
319   scale_color_manual(values = c("Pareto" = cbP[1], 'Exp' = cbP[2], 'Erlang' = cbP[3], 'Hypexp' =
320     cbP[4])) +
321   labs(title = sprintf("Distributions with mean = %s", SET_MEAN), y = "p(x)", x= "x", color = ''
322     ) +
323   ylim(c(0, 0.5))
324
325 p
326
327 variance_hyper <- (exp_p1 * 2 / exp_rate_1^2 + exp_p2 * 2 / exp_rate_2^2) - (exp_p1 / exp_rate_1 +
328   exp_p2 / exp_rate_2)^2
329 print(variance_hyper)
330
331 ### TAILS #####
332
333 x <- seq(8, 40, 0.1)
334
335 set_mean <- SET_MEAN
336
337 # PARETO
338
339 y_pareto <- pareto_pdf(beta = pareto_beta, k = pareto_k, x)
340
341 # EXP
342
343 y_exp <- exp_pdf(lambda = exp_lambda, x)
344
345 # ERLANG
346
347 y_erl <- erlang_pdf(k = erlang_k, lambda = erlang_lambda, x)
348
349 # HYPER
350
351 y_hypexp <- hyperexp_pdf(p1 = exp_p1, lambda1 = exp_rate_1, p2 = exp_p2, lambda2 = exp_rate_2, x)
352
353 df <- data.frame(x, y_pareto, y_exp, y_erl, y_hypexp)
354
355 p <- ggplot(data = df, mapping = aes(x = x)) +
356   geom_line(mapping=aes(y=y_pareto, color = "Pareto")) +
357   geom_line(mapping=aes(y=y_exp, color = "Exp")) +
358   geom_line(mapping=aes(y=y_erl, color = "Erlang")) +
359   geom_line(mapping=aes(y=y_hypexp, color = "Hypexp")) +
360   scale_color_manual(values = c("Pareto" = cbP[1], 'Exp' = cbP[2], 'Erlang' = cbP[3], 'Hypexp' =
361     cbP[4])) +
362   labs(title = sprintf("Distributions with mean = %s", SET_MEAN), y = "p(x)", x= "x", color = '')
363   +
364   ylim(c(0, 0.0025))
365
366 p
367
368 ### 2) Plotting Runs
369
370 # ----- PLOTTING A SINGLE RUN -----
371
372 data_line_plot_pareto <- algorithm_1(jumps = 1000, mean_y = SET_MEAN, distribution_y = "pareto")
373 data_line_plot_erlang <- algorithm_1(jumps = 1000, mean_y = SET_MEAN, distribution_y = "erlang")
374 data_line_plot_exponential <- algorithm_1(jumps = 1000, mean_y = SET_MEAN, distribution_y = "
  exponential")

```

```

370 data_line_plot_hyperexponential <- algorithm_1(jumps = 1000, mean_y = SET_MEAN, distribution_y = "
    hyperexponential")
371
372 low_x <- 420
373 upper_x <- 450
374
375 low_y <- data_line_plot_pareto$P[low_x + 1]
376 upper_y <- data_line_plot_pareto$M[upper_x]
377
378 p_pareto = ggplot() +
379   geom_line(data = data_line_plot_pareto, aes(x = time, y = P, colour = "Prior")) +
380   geom_line(data = data_line_plot_pareto, aes(x = time, y = A, colour = "After")) +
381   geom_step(data = data_line_plot_pareto, aes(x = time, y = M, colour = "Max"), direction = "vh")
382   +
383   xlim(low_x, upper_x) +
384   ylim(low_y, upper_y) +
385   scale_colour_manual("Pareto", values = c("Prior"=cbP[1], "After"=cbP[3], "Max"=cbP[8])) +
386   xlab('time') +
387   ylab('value') +
388   labs(title = sprintf("Pareto with mean = %s", SET_MEAN))
389
390 print(p_pareto)
391
392 low_x <- 420
393 upper_x <- 450
394
395 low_y <- data_line_plot_erlang$P[low_x + 1]
396 upper_y <- data_line_plot_erlang$M[upper_x]
397
398 p_erlang = ggplot() +
399   geom_line(data = data_line_plot_erlang, aes(x = time, y = P, colour = "Prior")) +
400   geom_line(data = data_line_plot_erlang, aes(x = time, y = A, colour = "After")) +
401   geom_step(data = data_line_plot_erlang, aes(x = time, y = M, colour = "Max"), direction = "vh")
402   +
403   xlim(low_x, upper_x) +
404   ylim(low_y, upper_y) +
405   scale_colour_manual("", values = c("Prior"=cbP[1], "After"=cbP[3], "Max"=cbP[8])) +
406   xlab('time') +
407   ylab('value') +
408   labs(title = sprintf("Erlang with mean = %s", SET_MEAN))
409
410 print(p_erlang)
411
412 low_x <- 420
413 upper_x <- 450
414
415 low_y <- data_line_plot_exponential$P[low_x + 1]
416 upper_y <- data_line_plot_exponential$M[upper_x]
417
418 p_exponential = ggplot() +
419   geom_line(data = data_line_plot_exponential, aes(x = time, y = P, colour = "Prior")) +
420   geom_line(data = data_line_plot_exponential, aes(x = time, y = A, colour = "After")) +
421   geom_step(data = data_line_plot_exponential, aes(x = time, y = M, colour = "Max"), direction = "
    vh") +
422   xlim(low_x, upper_x) +
423   ylim(low_y, upper_y) +
424   scale_colour_manual("", values = c("Prior"=cbP[1], "After"=cbP[3], "Max"=cbP[8])) +
425   xlab('time') +
426   ylab('value') +
427   labs(title = sprintf("Exponential with mean = %s", SET_MEAN))
428
429 print(p_exponential)
430
431 low_x <- 420
432 upper_x <- 450

```

```

431 low_y <- data_line_plot_hyperexponential$P[low_x + 1]
432 upper_y <- data_line_plot_hyperexponential$M[upper_x]
433
434
435 p_hyperexponential = ggplot() +
436   geom_line(data = data_line_plot_hyperexponential, aes(x = time, y = P, colour = "Prior")) +
437   geom_line(data = data_line_plot_hyperexponential, aes(x = time, y = A, colour = "After")) +
438   geom_step(data = data_line_plot_hyperexponential, aes(x = time, y = M, colour = "Max"),
439     direction = "vh") +
440   xlim(low_x, upper_x) +
441   ylim(low_y, upper_y) +
442   scale_colour_manual("", values = c("Prior"=cbP[1], "After"=cbP[3], "Max"=cbP[8])) +
443   xlab('time') +
444   ylab('value') +
445   labs(title = sprintf("Hyperexponential with mean = %s", SET_MEAN))
446
447 print(p_hyperexponential)
448
449 # 3) Defining algorithm to capture the value of the 1000th step in 100 runs
450
451 # ----- GENERATE RESULTS -----
452
453 results_pareto <- algorithm_1_extended(jump = 1000, distribution_y = "pareto", mean_y = SET_MEAN)
454
455 results_hyper <- algorithm_1_extended(jump = 1000, distribution_y = "hyperexponential", mean_y =
456   SET_MEAN)
457
458 results_exponential <- algorithm_1_extended(jump = 1000, distribution_y = "exponential", mean_y =
459   SET_MEAN)
460
461 results_erlang <- algorithm_1_extended(jump = 1000, distribution_y = "erlang", mean_y = SET_MEAN)
462
463 # ----- HISTOGRAM OF P -----
464
465 DF_P <- rbind(data.frame(dataset=1, obs=results_pareto$P_n),
466   data.frame(dataset=2, obs=results_hyper$P_n),
467   data.frame(dataset=3, obs=results_exponential$P_n),
468   data.frame(dataset=4, obs=results_erlang$P_n))
469
470 DF_P$dataset <- as.factor(DF_P$dataset)
471
472 ggplot(DF_P, aes(x=obs, fill=dataset)) +
473   geom_histogram(binwidth=100, colour="black", position="dodge") +
474   scale_fill_manual(breaks=1:4, values=c(cbP[1],cbP[3],cbP[8], cbP[2]),
475     labels=c("Pareto", "Hyper", "Exponential", "Erlang")) + labs(title = sprintf("
476     1000th P with mean = %s", SET_MEAN), x = "values", y = "counts", fill="")
477
478 # ----- HISTOGRAM OF A -----
479
480 DF_A <- rbind(data.frame(dataset=1, obs=results_pareto$A_n),
481   data.frame(dataset=2, obs=results_hyper$A_n),
482   data.frame(dataset=3, obs=results_exponential$A_n),
483   data.frame(dataset=4, obs=results_erlang$A_n))
484
485 DF_A$dataset <- as.factor(DF_A$dataset)
486
487 ggplot(DF_A, aes(x=obs, fill=dataset)) +
488   geom_histogram(binwidth=100, colour="black", position="dodge") +
489   scale_fill_manual(breaks=1:4, values=c(cbP[1],cbP[3],cbP[8], cbP[2]),
490     labels=c("Pareto", "Hyper", "Exponential", "Erlang")) + labs(title = sprintf("
491     1000th A with mean = %s", SET_MEAN), x = "values", y = "counts", fill="")

```

```

491 # ----- HISTOGRAM AND DENSITY OF M -----
492
493 DF_M <- rbind(data.frame(dataset=1, obs=results_pareto$M_n),
494               data.frame(dataset=2, obs=results_hyper$M_n),
495               data.frame(dataset=3, obs=results_exponential$M_n),
496               data.frame(dataset=4, obs=results_erlang$M_n))
497
498 DF_M$dataset <- as.factor(DF_M$dataset)
499
500 ggplot(DF_M, aes(x=obs, fill=dataset)) +
501   geom_histogram(binwidth=100, colour="black", position="dodge") +
502   scale_fill_manual(breaks=1:4, values=c(cbP[1],cbP[3],cbP[8], cbP[2],cbP[5]),
503                     labels=c("Pareto", "Hyper", "Exponential", "Erlang")) + labs(title = sprintf("
1000th M with mean = %s", SET_MEAN), x = "values", y = "counts", fill="")

```

## B.0.4 First passage probabilities

```

1
2 ##### First passage probabilities #####
3 jump_diffusion_max_sim_Yexp = function(mu, sigma, lambda, Tn, lambdaY, internal_rotation){
4   M_matrix = matrix(0, internal_rotation, Tn)
5
6   for(j in 1:internal_rotation){
7
8     W = rW(Tn, mu, sigma, lambda)
9     V = rV(Tn, mu, sigma, lambda)
10
11     Aml = 0
12     A = 0
13     M = c(0)
14
15     for (i in 1:Tn){
16       Y = rexp(1, lambdaY)
17       A = Aml+V[i]-W[i]+Y
18       M = c(M, max(M[i], Aml+V[i], A))
19       Aml = A
20     }
21
22     M_matrix[j,] = M[-1]
23   }
24   return(M_matrix)
25 }
26 jump_diffusion_max_sim_Ypareto = function(mu, sigma, lambda, Tn, betaY, kY, internal_rotation){
27   M_matrix = matrix(0, internal_rotation, Tn)
28
29   for(j in 1:internal_rotation){
30
31     W = rW(Tn, mu, sigma, lambda)
32     V = rV(Tn, mu, sigma, lambda)
33
34     Aml = 0
35     A = 0
36     M = c(0)
37
38     for (i in 1:Tn){
39       Y = rPareto(1, betaY, kY)
40       A = Aml+V[i]-W[i]+Y
41       M = c(M, max(M[i], Aml+V[i], A))
42       Aml = A
43     }
44

```

```

45     M_matrix[j,] = M[-1]
46 }
47 return(M_matrix)
48 }
49 jump_diffusion_max_sim_Yhyp = function(mu, sigma, lambda, Tn, probsY, lambdasY, internal_rotation)
50 {
51     M_matrix = matrix(0, internal_rotation, Tn)
52     for(j in 1:internal_rotation){
53
54         W = rW(Tn, mu, sigma, lambda)
55         V = rV(Tn, mu, sigma, lambda)
56
57         Aml = 0
58         A = 0
59         M = c(0)
60
61         for (i in 1:Tn){
62             Y = rhyper2(1, probsY, lambdasY)
63             A = Aml+V[i]-W[i]+Y
64             M = c(M, max(M[i], Aml+V[i], A))
65             Aml = A
66         }
67
68         M_matrix[j,] = M[-1]
69     }
70     return(M_matrix)
71 }
72 jump_diffusion_max_sim_Yerlang = function(mu, sigma, lambda, Tn, lambdaY, kY, internal_rotation){
73     M_matrix = matrix(0, internal_rotation, Tn)
74
75     for(j in 1:internal_rotation){
76
77         W = rW(Tn, mu, sigma, lambda)
78         V = rV(Tn, mu, sigma, lambda)
79
80         Aml = 0
81         A = 0
82         M = c(0)
83
84         for (i in 1:Tn){
85             Y = rErlang(1, lambdaY, kY)
86             A = Aml+V[i]-W[i]+Y
87             M = c(M, max(M[i], Aml+V[i], A))
88             Aml = A
89         }
90
91         M_matrix[j,] = M[-1]
92     }
93     return(M_matrix)
94 }
95
96 # Confidence intervals for a-contours
97 conf_int_exp = function(mu, sigma, lambda, Tn, lambdaY, internal_rot, external_rot, a, alpha,
98     plotting = TRUE){
99     internal_rotation = internal_rot
100     external_rotation = external_rot
101     Ms = array(0, c(internal_rot, Tn, external_rot))
102     M_prob = array(0, c(external_rotation, Tn, length(a)))
103
104     elapsed_time = 0
105     for (j in 1:external_rotation) {
106         time = Sys.time()
107         Ms[,j]=jump_diffusion_max_sim_Yexp(mu, sigma, lambda, Tn, lambdaY, internal_rotation)
108         end_time = as.numeric(Sys.time()-time)

```

```

108     elapsed_time = elapsed_time + end_time
109     print(paste("External round number:",j,"/",external_rotation))
110     print(paste("Elapsed time:",round(elapsed_time,2),"sec","; Expected total time", round(end_
111         time*(external_rotation-j)+elapsed_time,2),"sec"))
112 }
113 for (j in 1:length(a)){
114     for(i in 1:external_rot){
115         M_prob[i,,j] = colSums(Ms[,i]>a[j])/internal_rotation
116     }
117 }
118
119 means = matrix(0,Tn,length(a))
120 sds = matrix(0,Tn,length(a))
121 for (i in 1:length(a)) {
122     means[,i] = colMeans(M_prob[, ,i])
123     sds[,i] = colSds(M_prob[, ,i])
124 }
125
126 if(alpha<0.5){
127     quant = abs(qnorm(alpha/2))
128 }else{quant = abs(qnorm((1-alpha)/2))}
129
130 high = matrix(0, Tn, length(a))
131 low = matrix(0, Tn, length(a))
132 for (i in 1:length(a)){
133     high[,i] = means[,i]+quant*sds[,i]/sqrt(external_rotation)
134     low[,i] = means[,i]-quant*sds[,i]/sqrt(external_rotation)
135 }
136
137 return(list(upper = high, lower = low, mean = means))
138 }
139 conf_int_pareto = function(mu, sigma, lambda, Tn, betaY, kY, internal_rot, external_rot, a, alpha
140     , plotting = TRUE){
141     internal_rotation = internal_rot
142     external_rotation = external_rot
143     Ms = array(0, c(internal_rot,Tn,external_rot))
144     M_prob = array(0,c(external_rotation,Tn,length(a)))
145
146     elapsed_time = 0
147     for (j in 1:external_rotation) {
148         time = Sys.time()
149         Ms[, ,j]=jump_diffusion_max_sim_Ypareto(mu,sigma,lambda,Tn,betaY, kY,internal_rotation)
150         end_time = as.numeric(Sys.time()-time)
151         elapsed_time = elapsed_time + end_time
152         print(paste("External round number:",j,"/",external_rotation))
153         print(paste("Elapsed time:",round(elapsed_time,2),"sec","; Expected total time", round(end_
154             time*(external_rotation-j)+elapsed_time,2),"sec"))
155     }
156
157     for (j in 1:length(a)){
158         for(i in 1:external_rot){
159             M_prob[i,,j] = colSums(Ms[,i]>a[j])/internal_rotation
160         }
161     }
162
163     means = matrix(0,Tn,length(a))
164     sds = matrix(0,Tn,length(a))
165     for (i in 1:length(a)) {
166         means[,i] = colMeans(M_prob[, ,i])
167         sds[,i] = colSds(M_prob[, ,i])
168     }
169
170     if(alpha<0.5){
171         quant = abs(qnorm(alpha/2))

```

```

170 }else{quant = abs(qnorm((1-alpha)/2))}
171
172 high = matrix(0, Tn, length(a))
173 low = matrix(0, Tn, length(a))
174 for (i in 1:length(a)){
175     high[,i] = means[,i]+quant*sds[,i]/sqrt(external_rotation)
176     low[,i] = means[,i]-quant*sds[,i]/sqrt(external_rotation)
177 }
178
179 return(list(upper = high, lower = low, mean = means))
180 }
181 conf_int_hyp = function(mu, sigma, lambda, Tn, probsY, lambdasY, internal_rot, external_rot, a,
182     alpha, plotting = TRUE){
183     internal_rotation = internal_rot
184     external_rotation = external_rot
185     Ms = array(0, c(internal_rot, Tn, external_rot))
186     M_prob = array(0, c(external_rotation, Tn, length(a)))
187
188     elapsed_time = 0
189     for (j in 1:external_rotation) {
190         time = Sys.time()
191         Ms[, , j] = jump_diffusion_max_sim_Yhyp(mu, sigma, lambda, Tn, probsY, lambdasY, internal_rotation)
192         end_time = as.numeric(Sys.time()-time)
193         elapsed_time = elapsed_time + end_time
194         print(paste("External round number:", j, "/", external_rotation))
195         print(paste("Elapsed time:", round(elapsed_time, 2), "sec", "; Expected total time", round(end_
196             time*(external_rotation-j)+elapsed_time, 2), "sec"))
197     }
198
199     for (j in 1:length(a)){
200         for (i in 1:external_rot){
201             M_prob[i, , j] = colSums(Ms[, , i]>a[j])/internal_rotation
202         }
203     }
204
205     means = matrix(0, Tn, length(a))
206     sds = matrix(0, Tn, length(a))
207     for (i in 1:length(a)) {
208         means[, i] = colMeans(M_prob[, , i])
209         sds[, i] = colSds(M_prob[, , i])
210     }
211
212     if(alpha<0.5){
213         quant = abs(qnorm(alpha/2))
214     }else{quant = abs(qnorm((1-alpha)/2))}
215
216     high = matrix(0, Tn, length(a))
217     low = matrix(0, Tn, length(a))
218     for (i in 1:length(a)){
219         high[,i] = means[,i]+quant*sds[,i]/sqrt(external_rotation)
220         low[,i] = means[,i]-quant*sds[,i]/sqrt(external_rotation)
221     }
222
223     return(list(upper = high, lower = low, mean = means))
224 }
225 conf_int_erlang = function(mu, sigma, lambda, Tn, lambdaY, kY, internal_rot, external_rot, a,
226     alpha, plotting = TRUE){
227     internal_rotation = internal_rot
228     external_rotation = external_rot
229     Ms = array(0, c(internal_rot, Tn, external_rot))
230     M_prob = array(0, c(external_rotation, Tn, length(a)))
231
232     elapsed_time = 0
233     for (j in 1:external_rotation) {
234         time = Sys.time()

```



```

232 Ms[, ,j]=jump_diffusion_max_sim_Yerlang(mu,sigma,lambda,Tn,lambdaY, kY, internal_rotation)
233 end_time = as.numeric(Sys.time()-time)
234 elapsed_time = elapsed_time + end_time
235 print(paste("External round number:",j,"/",external_rotation))
236 print(paste("Elapsed time:",round(elapsed_time,2),"sec","; Expected total time", round(end_
    time*(external_rotation-j)+elapsed_time,2),"sec"))
237 }
238
239 for (j in 1:length(a)){
240   for(i in 1:external_rot){
241     M_prob[, ,j] = colSums(Ms[, ,i]>a[j])/internal_rotation
242   }
243 }
244
245 means = matrix(0,Tn,length(a))
246 sds = matrix(0,Tn,length(a))
247 for (i in 1:length(a)) {
248   means[, i] = colMeans(M_prob[, ,i])
249   sds[, i] = colSds(M_prob[, ,i])
250 }
251
252 if(alpha<0.5){
253   quant = abs(qnorm(alpha/2))
254 }else{quant = abs(qnorm((1-alpha)/2))}
255
256 high = matrix(0, Tn, length(a))
257 low = matrix(0, Tn, length(a))
258 for (i in 1:length(a)){
259   high[, i] = means[, i]+quant*sds[, i]/sqrt(external_rotation)
260   low[, i] = means[, i]-quant*sds[, i]/sqrt(external_rotation)
261 }
262
263 return(list(upper = high, lower = low, mean = means))
264 }
265
266 # Plotting function
267 plotting = function(data, alow, ahigh, by){
268   by = (ahigh-alow)/(by-1)
269   plot(0,0,ylim = c(0,1), xlim = c(1,1000), type = "l", main =
270     substitute(paste("First passage probability for a between ",alow," to ", ahigh," , by ",by
271       )),
272     list(alow = alow, ahigh = ahigh, by = by)),
273   ylab = "Probability", xlab = "Arrival")
274
275 locx = c()
276 locy = c()
277 labels = c()
278 for (i in 1:ncol(data$mean)) {
279   lines(1:1000,data$mean[, i], lty = 1)
280   lines(1:1000,data$upper[, i], lty = 2)
281   lines(1:1000,data$lower[, i], lty = 2)
282   #locx = c(locx, 1090)
283   #locy = c(locy, data$mean[, i])
284   #labels = c(labels, paste("a = ", (alow+(i-1)*by)))
285 }
286
287 legend("topleft", legend = c("Mean", "Confidence interval"), col = c("black", "black"), lty =
288   1:2)
289 #text(locx, locy, labels)
290 }
291
292 Exp_confs = conf_int_exp(mu = 0, sigma = 5, lambda = 1, Tn = 1000, lambdaY = 1/5, internal_rot =
293   100, external_rot = 100, a = seq(1000,3000, length.out = 10), alpha = 0.05)

```

```

292 Pareto_confs = conf_int_pareto(mu = 0, sigma = 5, lambda = 1, Tn = 1000, betaY = 2.560975610, kY =
    2.05, internal_rot = 100, external_rot = 100, a = seq(1000,3000, length.out = 10), alpha =
    0.05)
293 Erlang_confs = conf_int_erlang(mu = 0, sigma = 5, lambda = 1, Tn = 1000, lambdaY = 1, kY = 5,
    internal_rot = 100, external_rot = 100, a = seq(1000,3000, length.out = 10), alpha = 0.05)
294 Hyp_confs = conf_int_hyp(mu = 0, sigma = 5, lambda = 1, Tn = 1000, probsY = c(0.4,0.6), lambdasY
    = c(0.1,0.6), internal_rot = 100, external_rot = 100, a = seq(1000,3000, length.out = 10),
    alpha = 0.05)
295
296
297 # All pretty similar with mean 5.
298 par(mfrow = c(1,1))
299 plotting(Exp_confs,1000,3000, 10)
300 plotting(Pareto_confs,1000,3000, 10)
301 plotting(Erlang_confs,1000,3000, 10)
302 plotting(Hyp_confs,1000,3000, 10)
303
304
305 Pareto_confs105 = conf_int_pareto(mu = 0, sigma = 5, lambda = 1, Tn = 1000, betaY = 0.2381, kY =
    1.05, internal_rot = 150, external_rot = 20, a = seq(1000,3000, length.out = 10), alpha =
    0.05)
306 Pareto_confs110 = conf_int_pareto(mu = 0, sigma = 5, lambda = 1, Tn = 1000, betaY = 0.4545454545,
    kY = 1.10, internal_rot = 150, external_rot = 20, a = seq(1000,3000, length.out = 10), alpha =
    0.05)
307 Pareto_confs115 = conf_int_pareto(mu = 0, sigma = 5, lambda = 1, Tn = 1000, betaY = 0.6521739130,
    kY = 1.15, internal_rot = 150, external_rot = 20, a = seq(1000,3000, length.out = 10), alpha =
    0.05)
308 Pareto_confs125 = conf_int_pareto(mu = 0, sigma = 5, lambda = 1, Tn = 1000, betaY = 1, kY = 1.25,
    internal_rot = 150, external_rot = 20, a = seq(1000,3000, length.out = 10), alpha = 0.05)
309 Pareto_confs150 = conf_int_pareto(mu = 0, sigma = 5, lambda = 1, Tn = 1000, betaY = 1.666666667,
    kY = 1.5, internal_rot = 150, external_rot = 20, a = seq(1000,3000, length.out = 10), alpha =
    0.05)
310 Pareto_confs175 = conf_int_pareto(mu = 0, sigma = 5, lambda = 1, Tn = 1000, betaY = 2.142857143,
    kY = 1.75, internal_rot = 150, external_rot = 20, a = seq(1000,3000, length.out = 10), alpha =
    0.05)
311 Pareto_confs205 = Pareto_confs
312
313 # We see a heavy tail for small values of k results in different graphs. (All with mean 5)
314 plotting(Pareto_confs105,1000,3000, 10)
315 plotting(Pareto_confs110,1000,3000, 10)
316 plotting(Pareto_confs115,1000,3000, 10)
317 plotting(Pareto_confs125,1000,3000, 10)
318 plotting(Pareto_confs150,1000,3000, 10)
319 plotting(Pareto_confs175,1000,3000, 10)
320 plotting(Pareto_confs205,1000,3000, 10)
321
322
323
324 # Only brownian motion without drift
325 plotting(conf_int_exp(mu = 0, sigma = 5, lambda = 1, Tn = 1000, lambdaY = 1000, internal_rot =
    100, external_rot = 20, a = seq(10,100, length.out = 10), alpha = 0.05), 10, 100, 10)
326
327 # Hyper
328 Hyp_confs_pareto = conf_int_hyp(mu = 0, sigma = 5, lambda = 1, Tn = 1000, probsY = c
    (0.9995,1-0.9995), lambdasY = c(90,0.0001002226055), internal_rot = 100, external_rot = 20, a
    = seq(1000,3000, length.out = 10), alpha = 0.05)
329
330 plotting(Hyp_confs_pareto,1000,3000,10)

```

## B.0.5 European Call option

```

1 ##### Ex 6 – European option valuation #####
2 set.seed(69)
3 exercise_time = 10
4 strike = 40
5 sigma = 0.3
6 rf = 0.05
7 S0 = 40
8 runs = 100000
9
10 # Rate
11 R= (rf-1/2*sigma^2)*exercise_time
12
13 # Standard deviation
14 SD = sigma*sqrt(exercise_time)
15
16 # Stock price at T
17 STS = S0 * exp(R+SD*rnorm(runs,0,1))
18
19
20 ## Crude
21 V = pmax(STS-strike,0)*exp(-rf*exercise_time)
22 mean(V)
23 sd(V)
24 sd(V)/sqrt(runs)*1.96
25
26
27 ## Control
28 Y = STS
29 mu_Y = rep(exp(rf*exercise_time)*S0, runs)
30 c = -cov(V, STS)/var(STS)
31
32 VC = V+c*(Y-mu_Y)
33 mean(VC)
34 sd(VC)
35 sd(VC)/sqrt(runs)*1.96
36
37
38 ## Black-Scholes
39 BlackSch = function(S0, strike, SD, rf, exercise_time, initT){
40   # rf = risk free rate
41
42   d1 = (log(S0/strike)+(rf+SD^2*1/2)*(exercise_time-initT))/(SD*sqrt(exercise_time-initT))
43   d2 = d1 - SD*sqrt(exercise_time-initT)
44   cost = S0*pnorm(d1,mean=0,sd = 1)-strike*exp(-rf*(exercise_time-initT))*pnorm(d2,mean=0,sd = 1)
45   return(cost)
46   #print(paste('The cost is ',cost))
47 }
48
49 # tested with results from:
50 # http://www.cliffordang.com/models/optionsMC.pdf
51
52 BlackSch(S0=S0, strike=strike, SD=sigma, rf=rf, exercise_time=exercise_time, initT=0)

```