Danmarks Tekniske Universitet

Simulation of Lévy Processes

June 2020

Authors:	***Study No:***
Nicolaj Hans Nielsen	s184335
Anton Ruby Larsen	s174356
Andreas Heidelbach Engly	s170303
Karl Emil Takeuchi-Storm	s130377

Contents

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:

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

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, σB_t, with drift, μt.

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:* $\lim_{t \to s} L_t = L_s$

[8]

Leading to the general formulation:

$$X_t = \mu t + \sigma B_t + Z_t \tag{2}$$

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

2.1.2 Compound Poisson process

Let the sequence $\{T_1, T_2, ..., T_n\}$ be the interarrival times of a number of events. $\{T_1, T_2, ..., T_n\}$ 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 λ. 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 $B : \mathbb{R} \to \mathbb{R}$ 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* $T \sim \mathrm{Exp}(\lambda)$

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

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

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

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 $\lambda_Y = 2$, $\sim Exp(\lambda_Y = 2)$. Further, we set the drift $\mu = 0.1$, the Gaussian intensity $\sigma = 0.5$, and use intensity of $\lambda_{jump} = 1$ for the Poisson process.

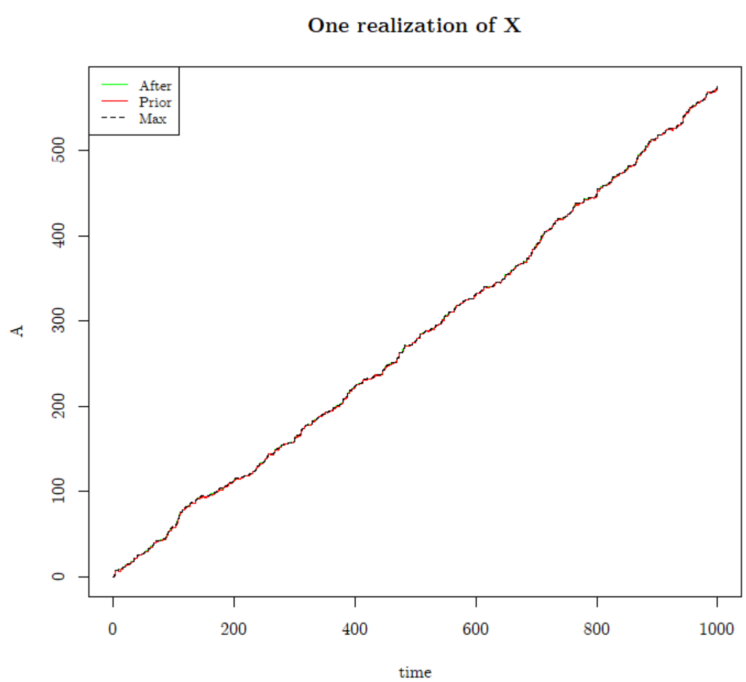

Figure 1: One simulation of 1000 arrivals with $\lambda = 1,\ \sigma = 0.5,\ \mu = 0.1,\ \lambda_Y = 2$

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.

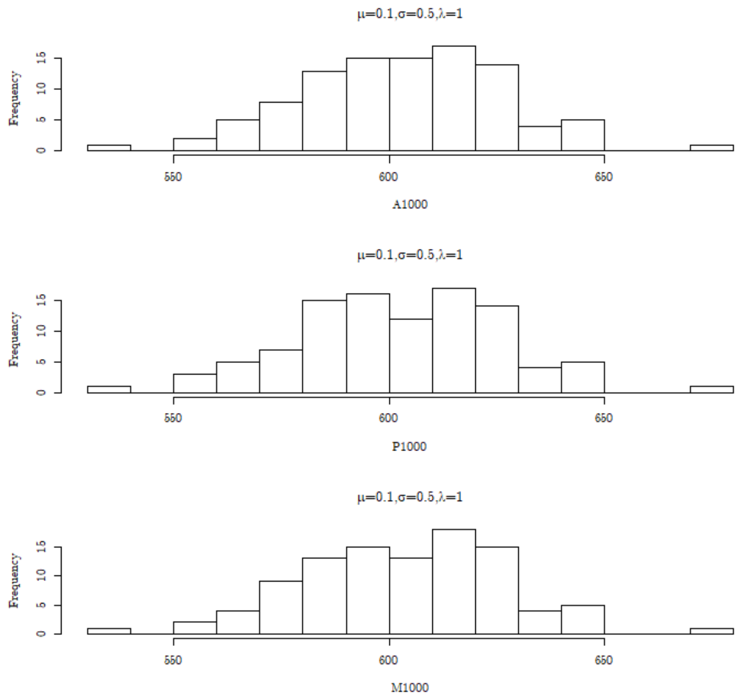

Figure 2: Histograms of 100 realizations of the 1000th arrival of the processes P, A and M

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.

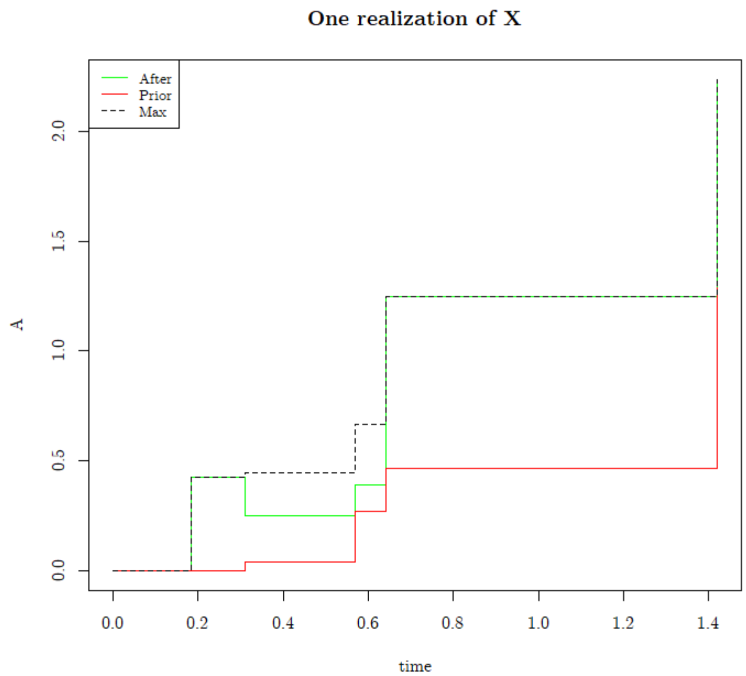

Figure 3: One simulation of 5 arrivals with $\lambda = 1,\ \sigma = 0.6,\ \mu = 0.1,\ \lambda_Y = 2$

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

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, $Y_i \sim Exp(\lambda_Y = 2)$.

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, $Mi = A_{i-1} + (V_i + W_i) + Y_i$. 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 μ, σ, and λ. As can be seen in equation 1, $\mu\, t$ simply adds a trend to the overall function, where a high μ will correspond to a large slope in an affine function. The gain of the Brownian motion is σ, which increases the amplitude of the Brownian motion. Lastly the constant λ determines how often jumps occur, as if we increase we would decreasing the time between jumps. The expectation of an exponential process is $\frac{1}{\lambda}$ which corresponds to the average time between occurrences. Therefore, a λ under 1 will on average corresponds to less that 1 jump per time unit whereas λ 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 μ

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

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

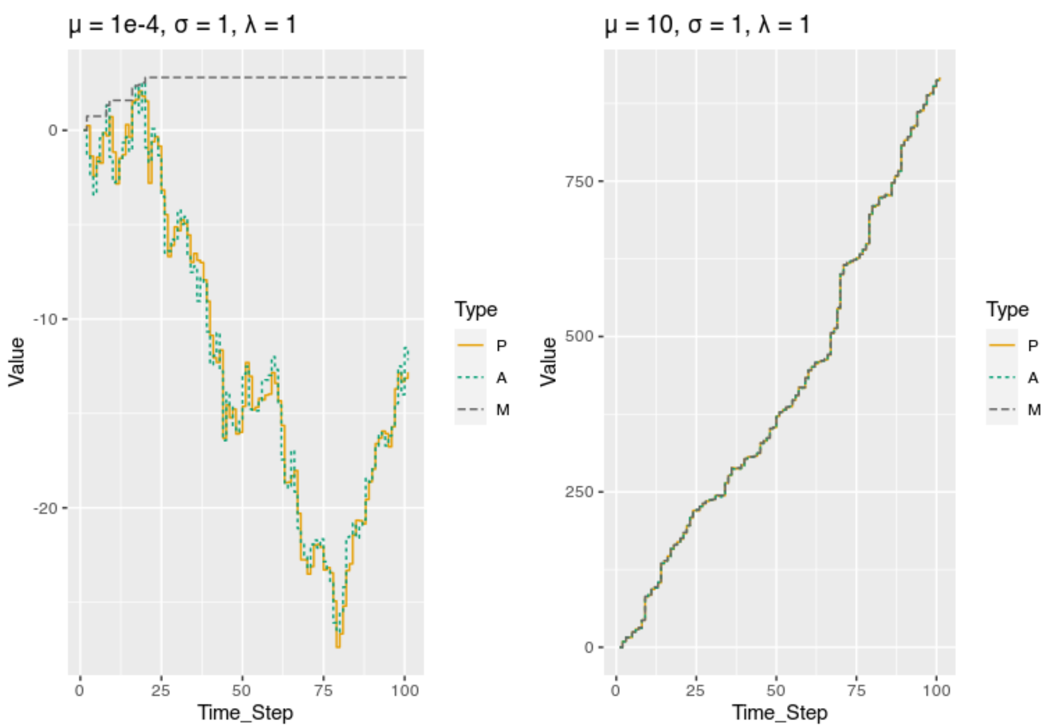

Figure 4: Two simulations of extreme μ values

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 σ and λ of 1. In figure 5 the mean value of processes after the jumps with changing μ 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 μ values.

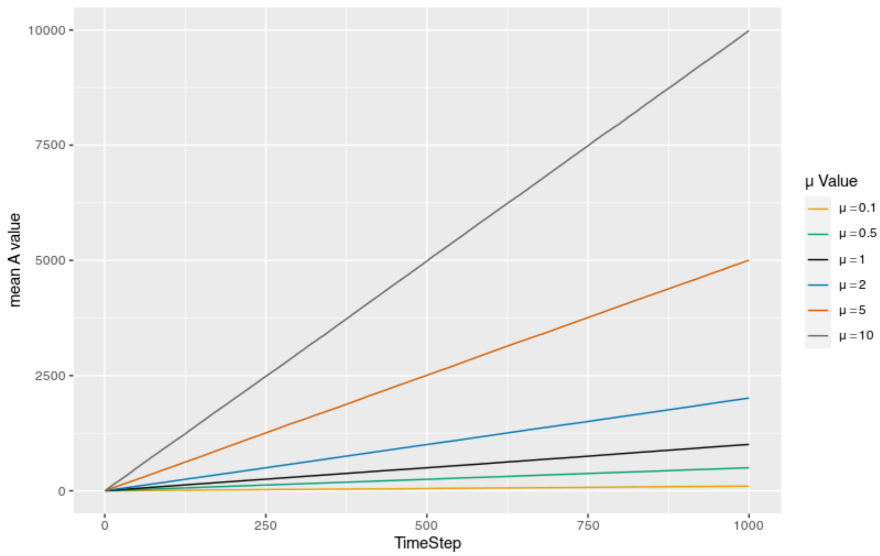

Figure 5: Average *after jump* values for 1000 simulations

2.3.2 Effects of variation of σ

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

Next we look at σ which has been plotted in Figure 6.

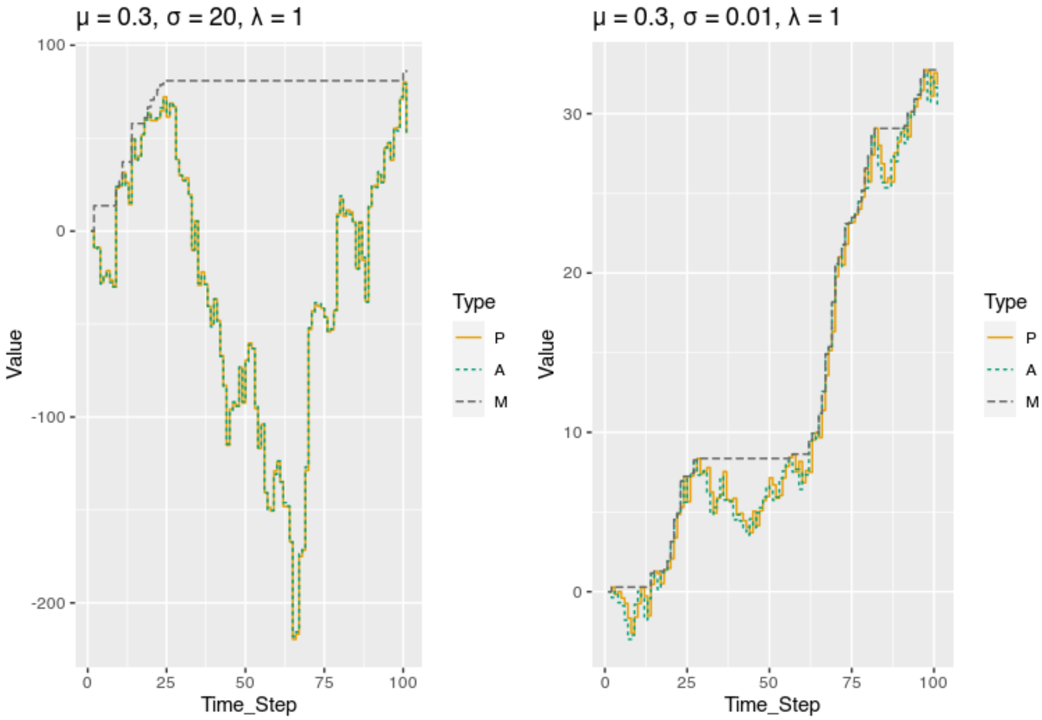

Figure 6: Two simulations of extreme σ values

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 σ, similar simulations are carried out a thousand times for a thousand time steps. The simulation has a μ of 1 and $\sigma = [1, 10, 25, 50, 100, 500]$. The plot in figure 5 shows the average maximum attained value of the Lévi process. The effect of the Brownian motion with σ 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.

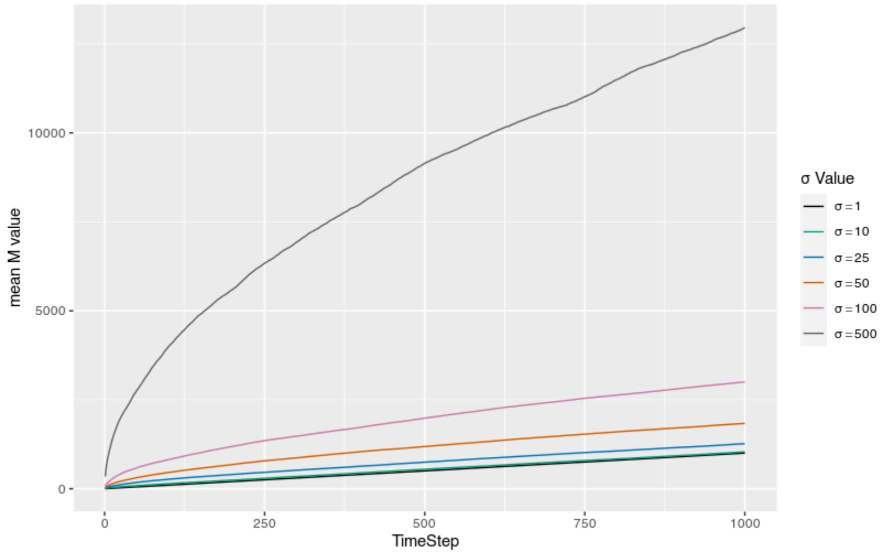

Figure 7: Average *max attained* values for 1000 simulations

2.3.3 Effects of variation of λ

Lastly, we look at λ. If we look at the time scale, we see that in the left plot where λ 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.

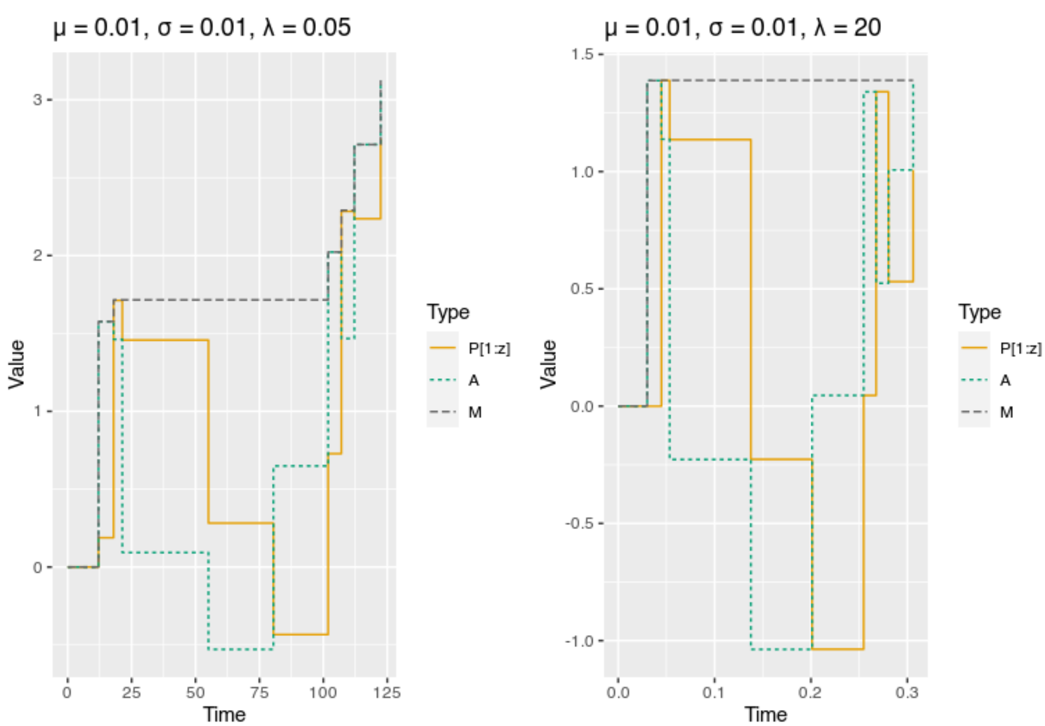

Figure 8: Two simulations of extreme λ values

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 μ 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.

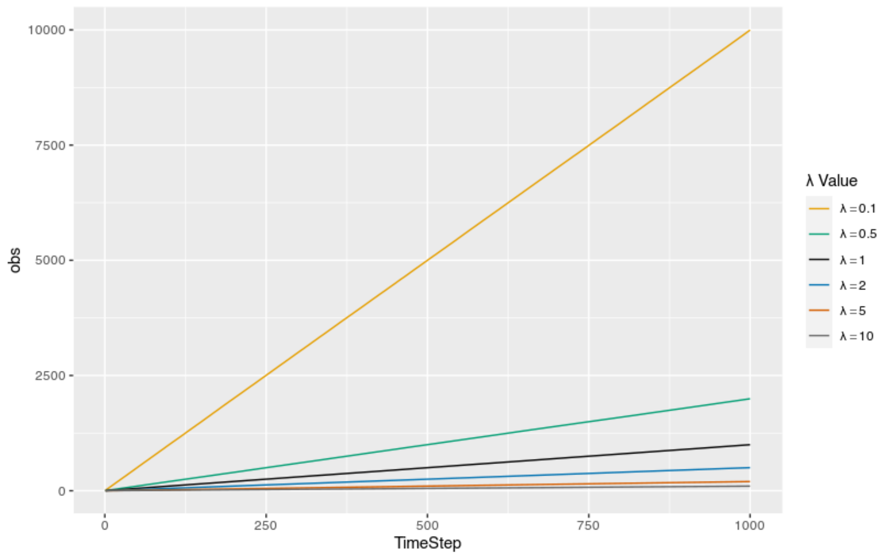

Figure 9: Average *after jump* values for 1000 simulations

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.

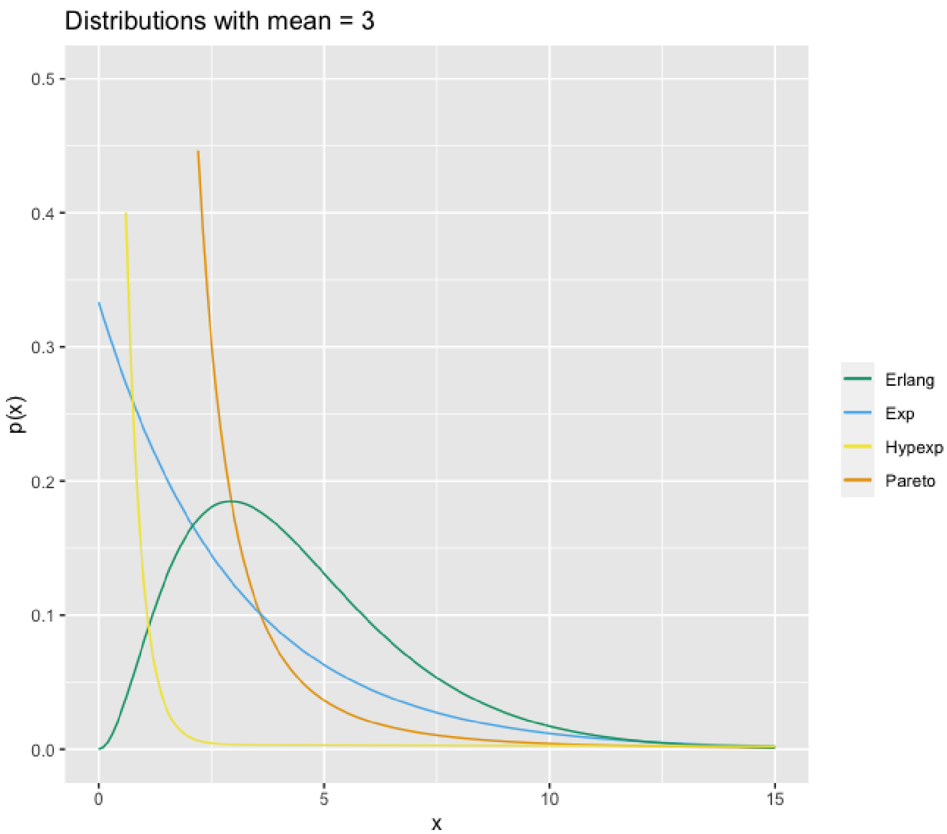

Figure 10: Probability Density Functions

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.

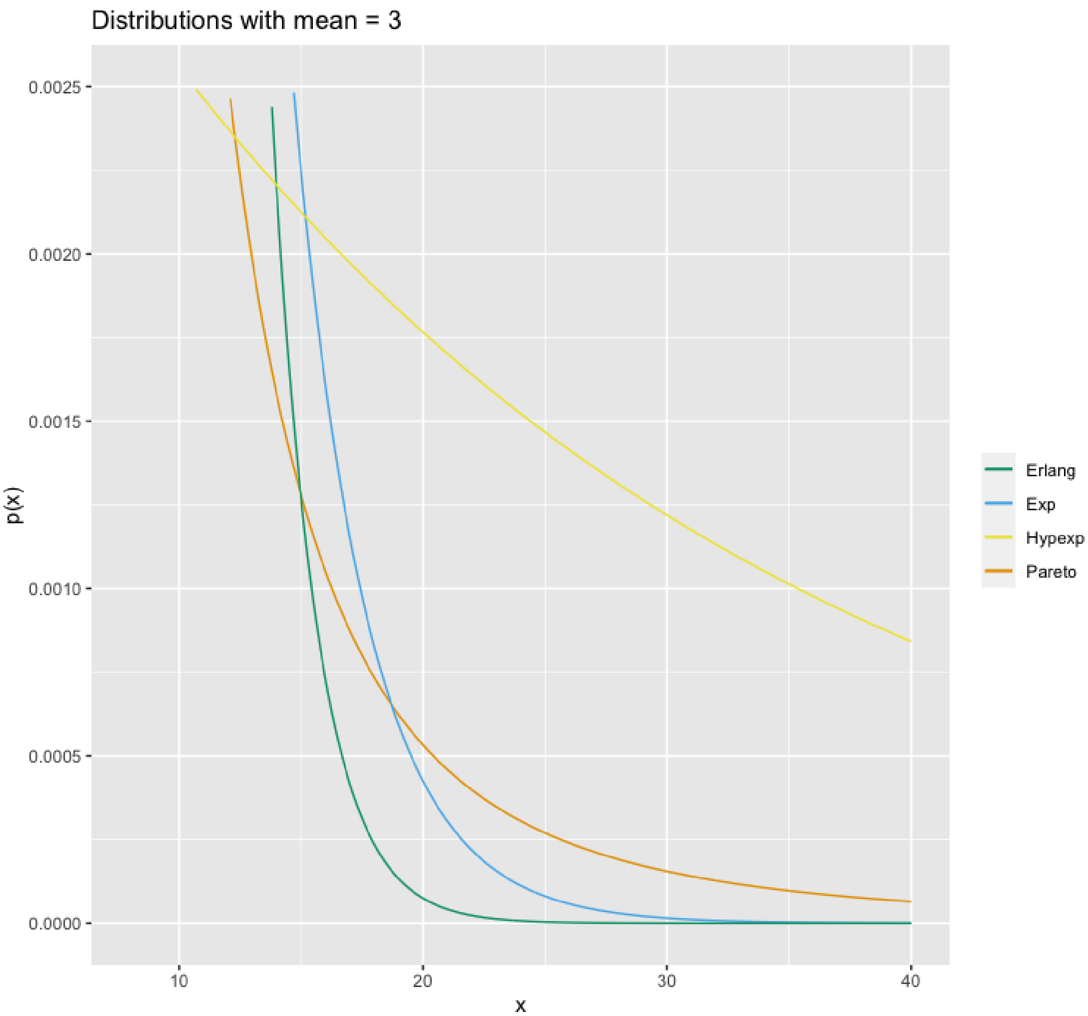

Figure 11: Tails of Probability Density Functions

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 σ, μ, and λ. In these examples we use fixed $\sigma = 0.2$, $\mu = 1$, and $\lambda = 2$.

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

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

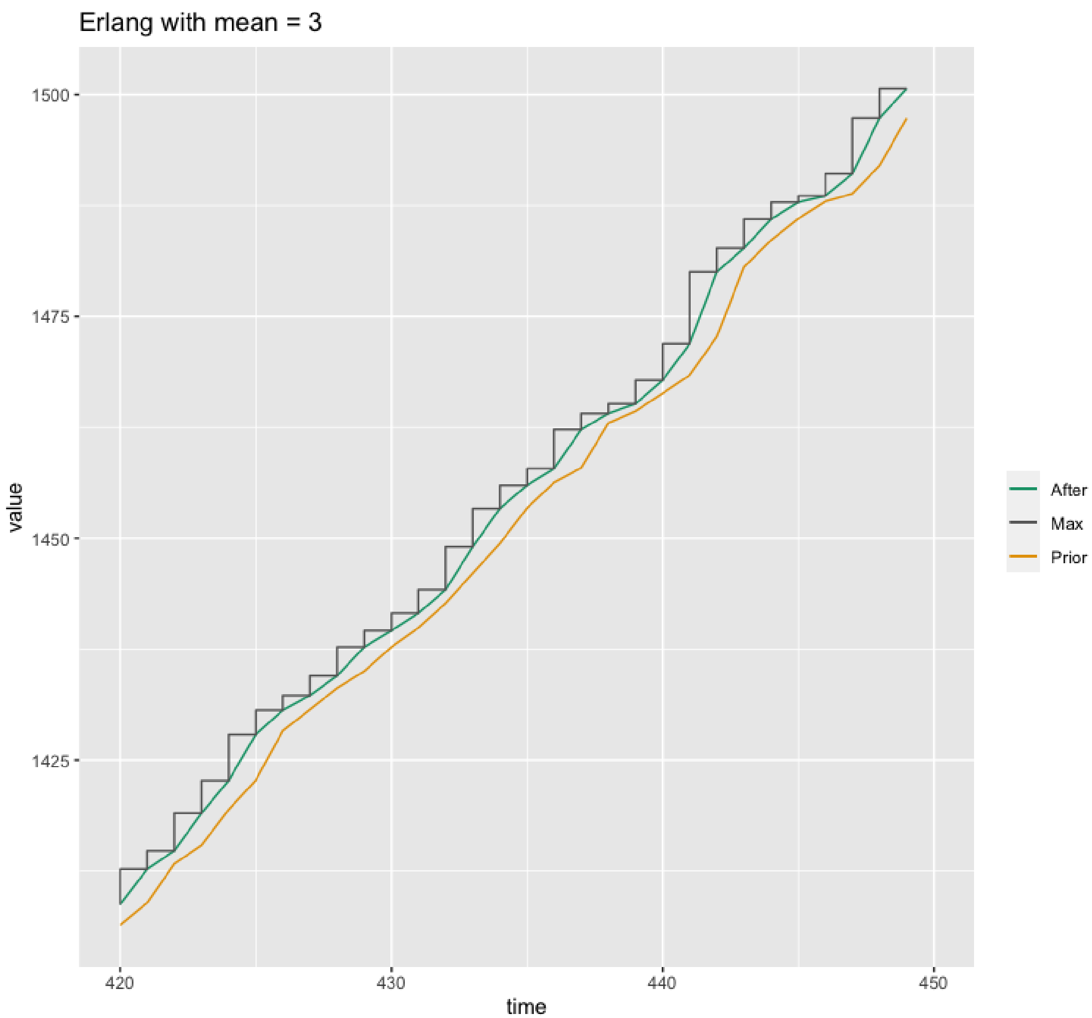

Figure 12: Realization with Erlang distributed jump sizes

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

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

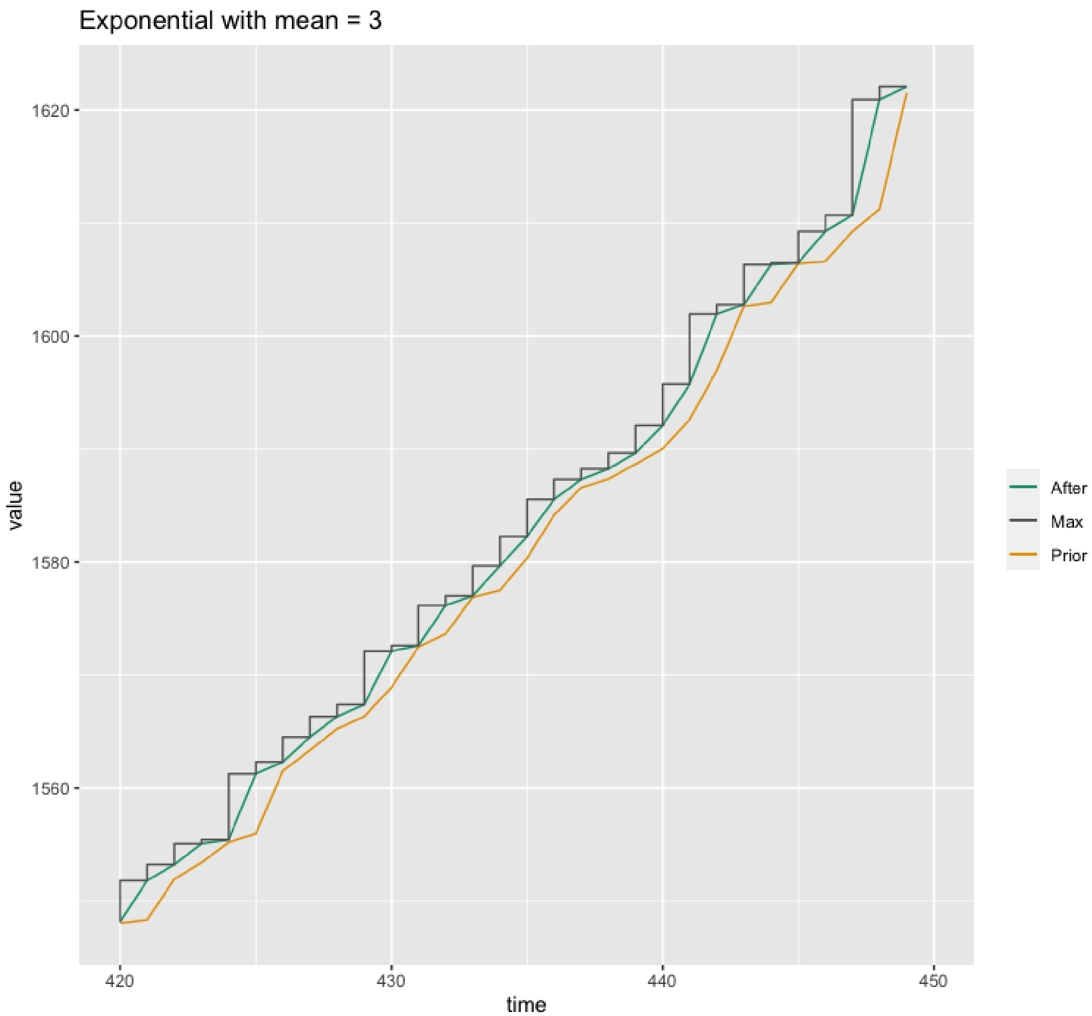

Figure 13: Jump Size distributed by Exponential

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

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

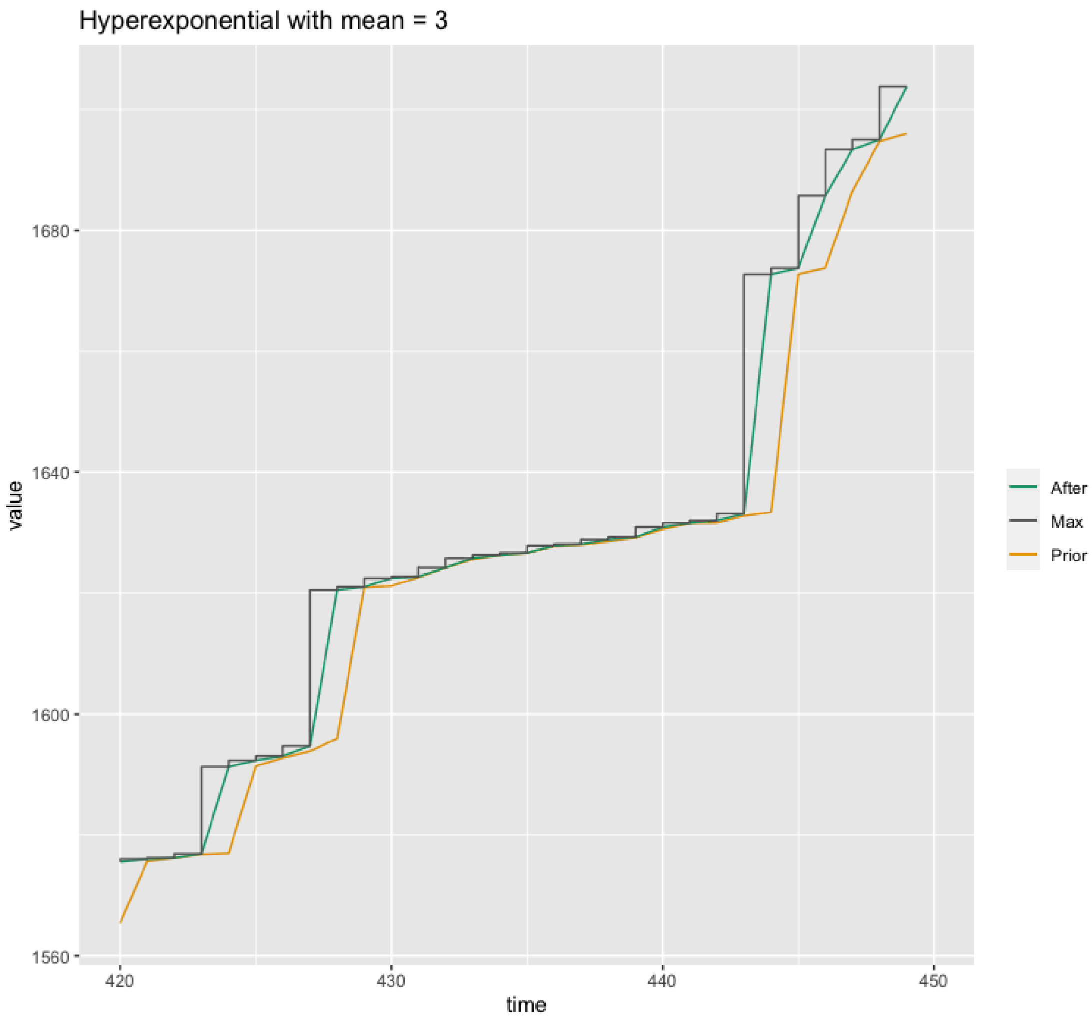

Figure 14: Jump Size distributed by Hyperexponential

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

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

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

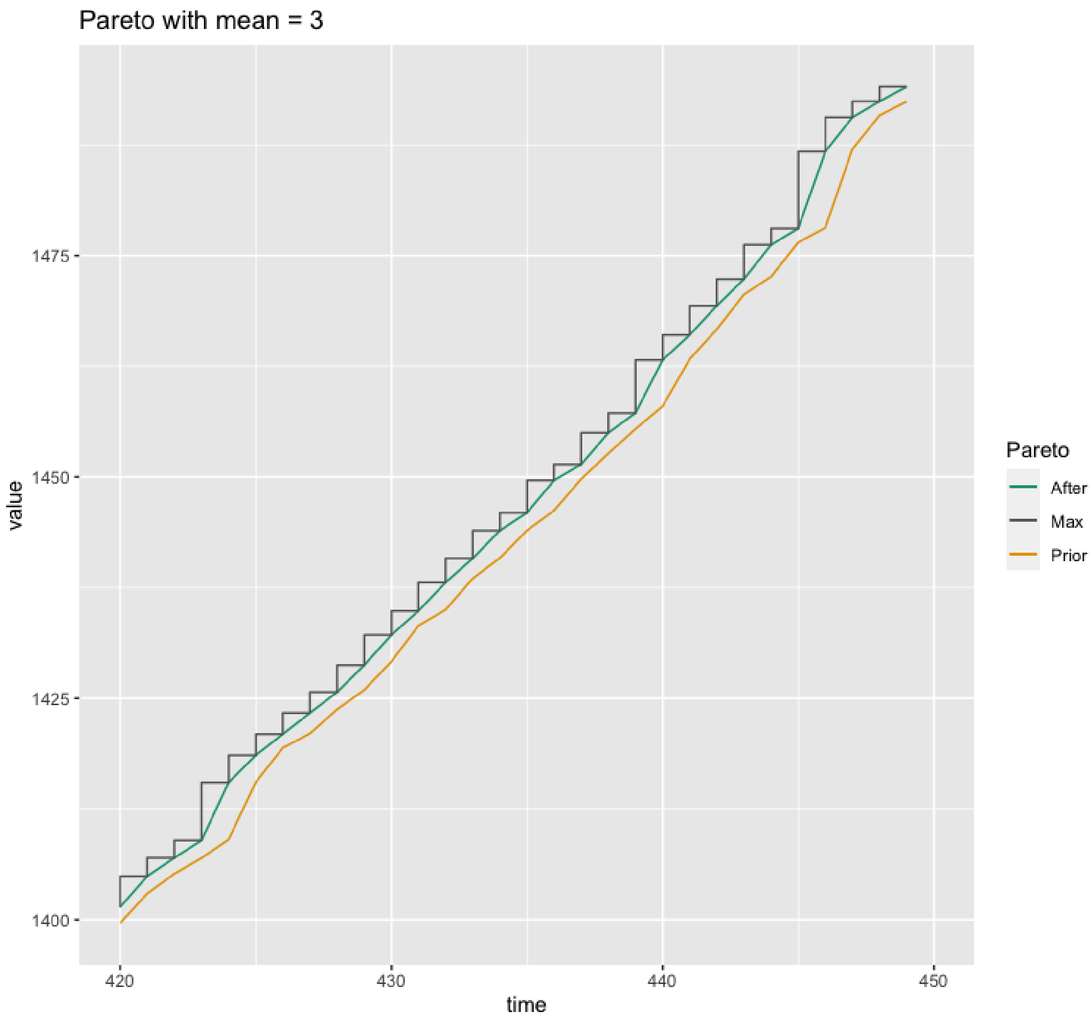

Figure 15: Jump Size distributed by Pareto

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.

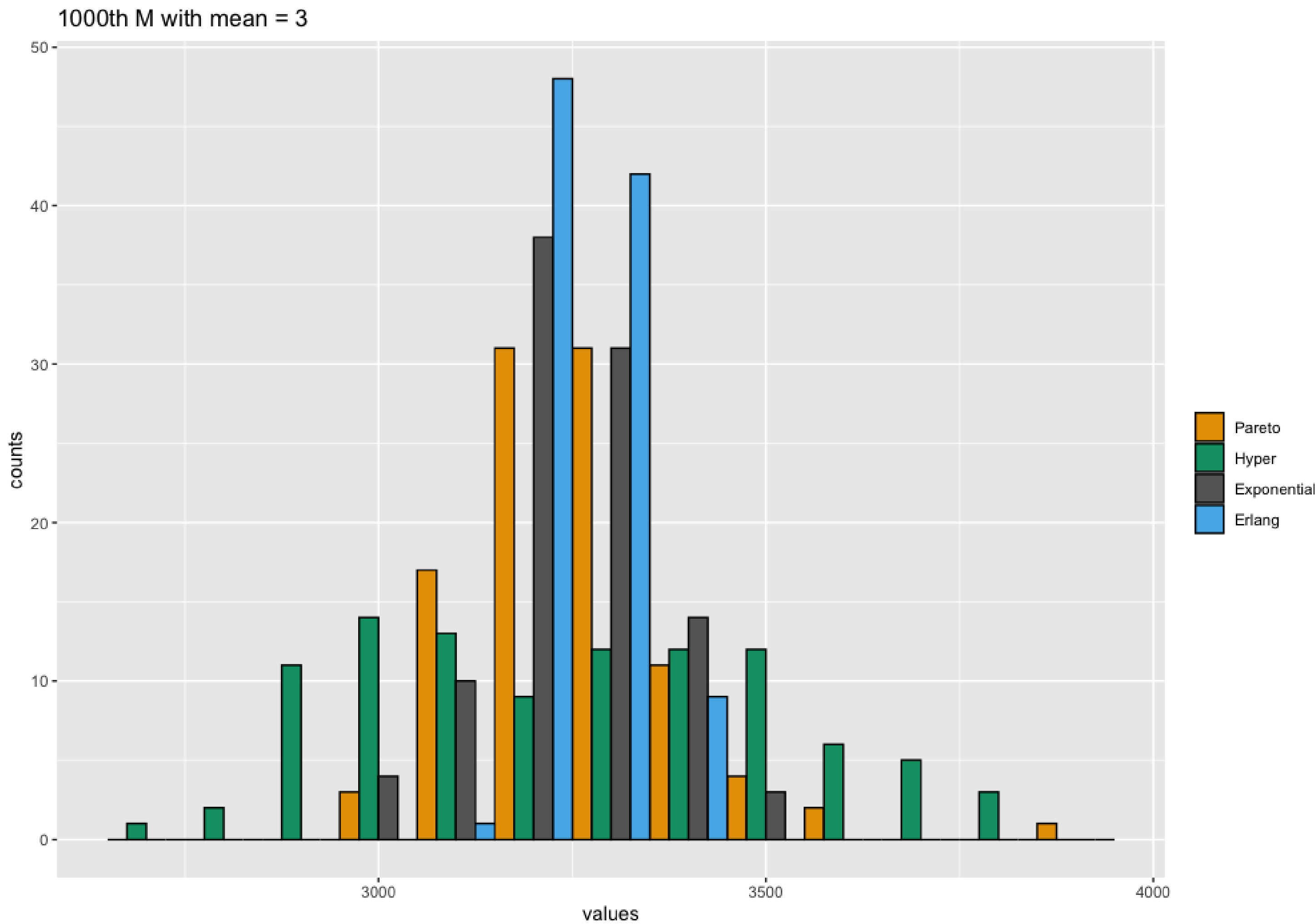

Figure 16: 1000th Step for Maximum Value

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 some fixed a. We will start by looking at first passage probabilities for 10 a's equidistant from 1000 to 3000. The constants μ, σ, and λ 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.

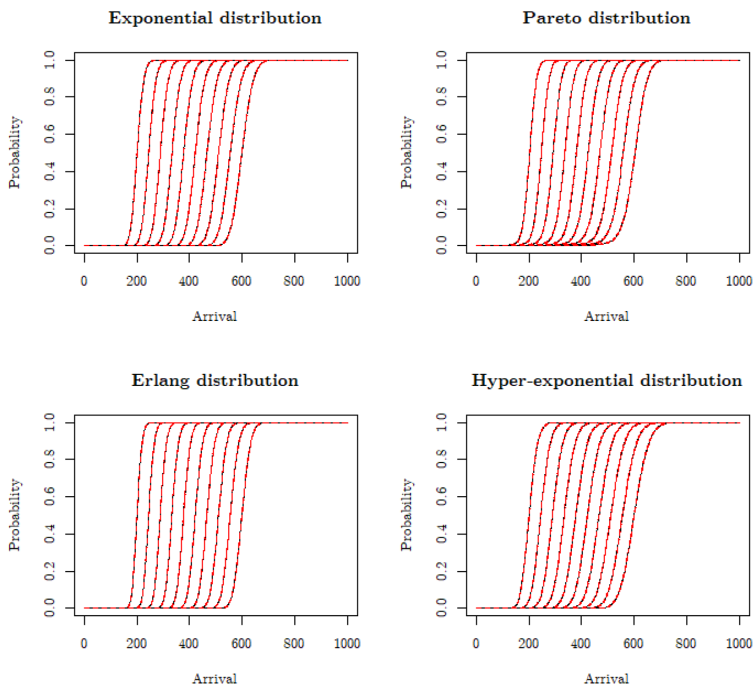

Figure 17: First passage probabilities with equidistant a's from 1000 to 3000

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 $222.\dot{2}$ 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.

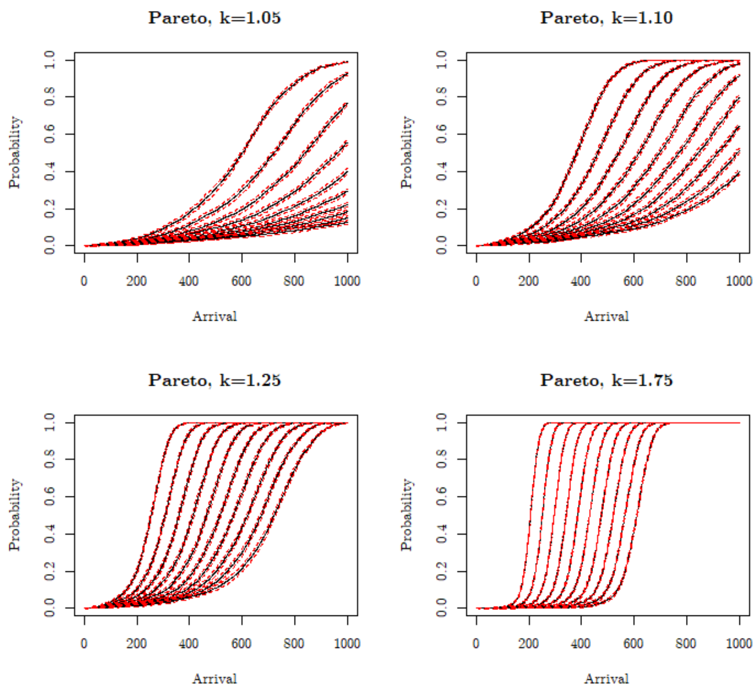

Figure 18: The parameters of the distributions were (k=1.05, β =0.2381), (k=1.10, β =0.4545), (k=1.25, β =1), and (k=1.75, β =2.1429)

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.

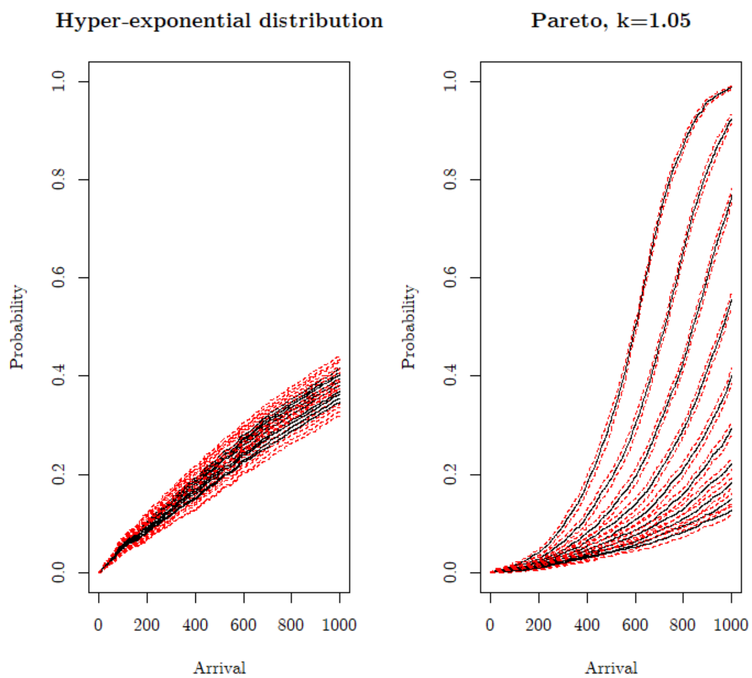

Figure 19: The hyper-exponential had p=(0.9994,0.0006) and λ =(20,0.0001177805843) and the pareto had k=1.05, β =0.2381

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.

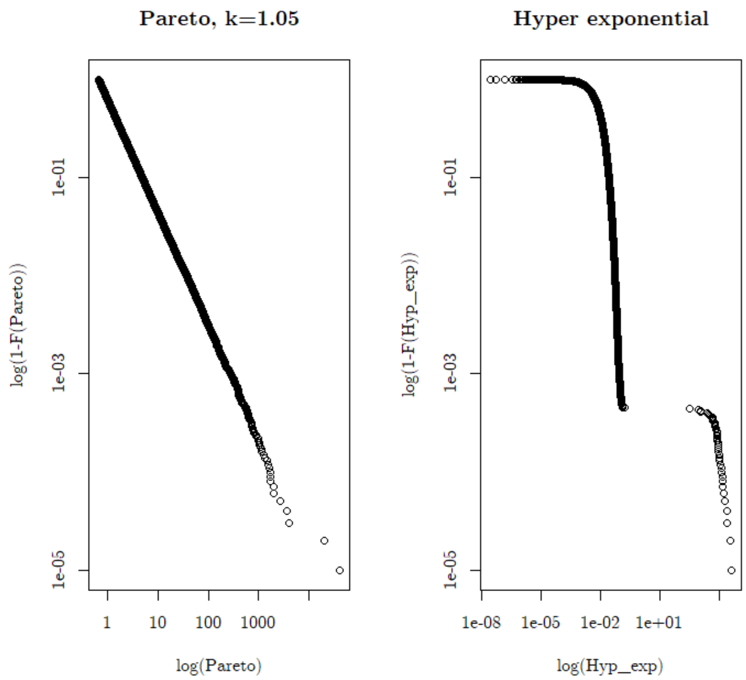

Figure 20: The hyper-exponential had p=(0.9994,0.0006) and λ =(20,0.0001177805843), while the Pareto had k=1.05, β =0.2381

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.

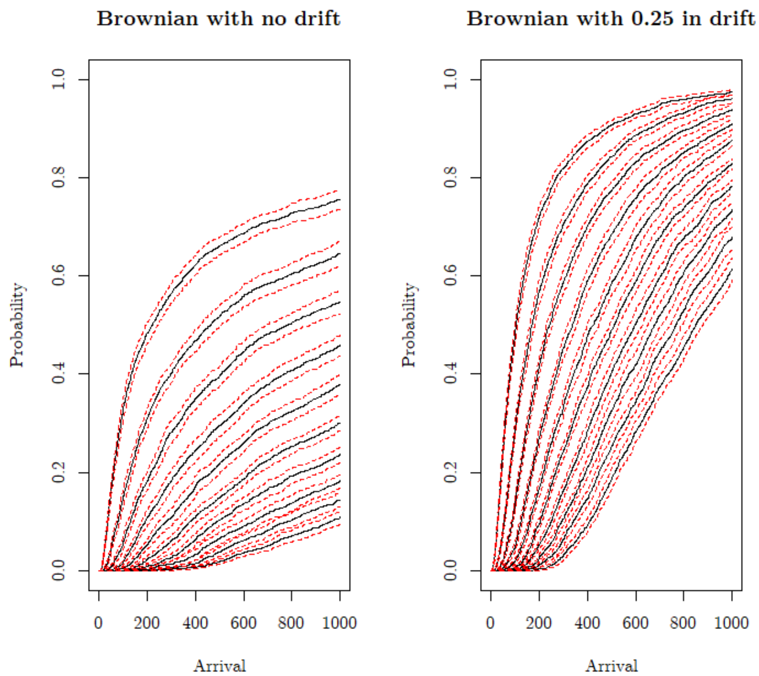

Figure 21: The a lines are from 50 to 250 and the Brownian motion has a $\sigma = 5$ and the drift can be read from the titles.

In Figure 21 we see two plots where only the Brownian motion is simulated. The jump process was still in the simulation but λ 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:

$$dS_t = \mu S_t dt + \sigma S_t dB_t \tag{3}$$

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].

$$du = u'dx + \frac{1}{2}u''dx^2 \tag{4}$$

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

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

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

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

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

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

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

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 μ is the same μ 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 know 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:

$$Var[Z_i] = Var[V_i] + c^2 Var[Y_i] + 2cCov[V_i, Y_i] \tag{8}$$

If we saw this variance estimate as a function of c, $h(c)$, then we have $h'(c) = 2cVar[Y_i] + 2Cov[V_i, Y_i]$ 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:

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

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 mean the expectation is given by $e^{\mu T} S_0$. We therefore get

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

We test this in the section 3.3.

3.2 Black-Scholes Model

The following is based on (*Black-Scholes Model, p. 1i*)[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, σ, 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]:

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

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 underling 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 underling 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:

$$P_t = x_t M_t + y_t S_t \tag{12}$$

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

$$\begin{align} dP_t &= x_t dB_t + y_t dS_t \tag{13} \\ &= r x_t M_t dt + y_t(\mu S_t dt + \sigma S_t dB_t) \tag{14} \\ &= (r x_t M_t + y_t \mu S_t)dt + y_t \sigma S_t dB_t \tag{15} \end{align}$$

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:

$$\begin{align} \sigma S_t &= \sigma S_t \frac{\partial V}{\partial S} \nonumber \\ \Rightarrow y_t &= \frac{\partial V}{\partial S} \tag{16} \\ &\text{and} \nonumber \\ r x_t M_t + y_t \mu S_t &= \mu S_t \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \nonumber \\ \Rightarrow r x_t M_t &= \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \tag{17} \end{align}$$

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:

$$\begin{align} P_t = V_t &= x_t M_t + y_t S_t \nonumber \\ \Rightarrow r V_T &= \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + r \frac{\partial V}{\partial S} S_t \nonumber \\ \Rightarrow 0 &= \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + r \frac{\partial V}{\partial S} S_t - r V_T \tag{18} \end{align}$$

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 $V(S,t) \to S$ as $S \to \infty$.

The closed-form solution to equation 18 is:

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

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

$$E[V_{control}] = 21.04399 \pm 0.03778968 \tag{19}$$

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 Leví 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 Leví 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 μ affects the overall trend of the model, σ, amplifies the variance of the Brownian motion. When we increase the intensity of the Poisson process, λ, 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. 2006.

[2] Bruce C. Dieffenbach. *Itô's Formula*. 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.

[4] Martin Haugh. *Leture 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.

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

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

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

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.

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

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.

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

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 Pareto distribution for $\lambda, x \geq 0$ and $k \in Z_+$ is as follows:

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

,

Let $X \sim Erl(\lambda, k)$. 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).

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

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 $X \sim Exp(\lambda)$. Then we derive the expectation.

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

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

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

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).

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

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 $X \sim Hyperexp(\{\lambda_i, p_i\}_{i=1..n})$. 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 λ 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).

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

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

```

rV = function(n, mu, sigma, lambda){
  phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi1))
}
rW = function(n, mu, sigma, lambda){
  phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi2))
}

# Illustrating P, A, and M
par(mfrow = c(1,1))
iter = 5
lambda = 1/20
sigma = 0.01
mu = 0.01
lambda_Y = 0.1
time = c(0,cumsum(rexp(iter, lambda)))
W = rW(iter, mu, sigma, lambda)
V = rV(iter, mu, sigma, lambda)
```

```


P = c(0)
A = c(0)
M = c(0)

Y = c()


for (i in 1:iter){
  Y = c(Y,rexp(1, lambda_Y))
  P = c(P, V[i]-W[i]+A[i])
  A = c(A, A[i]+V[i]-W[i]+Y[i])
  M = c(M, max(M[i], A[i]+V[i], A[i+1]))
}

index = seq(0,iter, length.out = iter+1)
plot( time, A, type = "s", col = "green", main = substitute(paste(mu, "=", mu_num,",", sigma, "=", sigma_num, ",",lambda, "=",lambda_num ),list(lambda_num = lambda, sigma_num = sigma, mu_num= mu)))
lines( time, P, type = "s", col = "red")
lines(time, M, type = "s", col = "black", lty = 2)
legend("topleft", legend=c("After", "Prior", "Max"),
       col=c("green", "red", "black"), lty=c(1,1,2), cex=0.8)
```

B.0.2 Explanation of the constants

```

library(ggplot2)
library(reshape2)

cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")

rV = function(n, mu, sigma, lambda){
  phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi1))
}

rW = function(n, mu, sigma, lambda){
  phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi2))
}

jumpDiffusionSimYexp = function(mu, sigma, lambda, Tn, lambdaY, n){
  A = matrix(nrow = Tn, ncol = n)
  P = matrix(nrow = Tn, ncol = n)
  M = matrix(nrow = Tn, ncol = n)

  for(j in 1:n){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    P[1,j] = 0
    A[1,j] = 0
    M[1,j] = 0
    Mm1 = 0
    Am1 = 0


```

```
    for (i in 1:Tn){
      Y = rnorm(1, 0,1)
      P[i,j] =  V[i]-W[i]+Am1
      A[i,j] = Am1+V[i]-W[i]+Y
      Mm1 =  max(Mm1, Am1+V[i], A[i,j])
      M[i,j] = Mm1
      Am1 = A[i,j]
    }
  }
  return(list(P,A,M))
}

set.seed(23)
mu = c(0.1,0.5,1,2,5,10)
Pdf = data.frame()
Adf = data.frame()
Mdf = data.frame()
Tn = 1000
n=100
for (i in 1:length(mu)){

  X = jumpDiffusionSimYexp(mu[i], 1, 1, Tn, 2,n)

  Pdf <- rbind(Pdf,
               data.frame(dataset=(paste("P", mu[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:Tn))
                   )
  Adf <- rbind(Adf,
               data.frame(dataset=(paste("A", mu[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:Tn))
                   )
  Mdf <- rbind(Mdf,
               data.frame(dataset=(paste("M", mu[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:Tn))
                   )
}
Adf$dataset <- as.factor(Adf$dataset)

Mulabs = c()
for (i in 1:length(mu)){
  Mulabs = c(Mulabs,bquote(mu==.(mu[i])))
}
muLabel = "\u03bc Value"
#$

ggplot(Adf, aes(x=TimeStep)) +
  geom_line(aes(y=obs, color=dataset))+
  scale_color_manual(values=c(cbP[1],cbP[3],cbP[10],cbP[5],cbP[6],cbP[8]),
                     labels=Mulabs)+
  labs(color=muLabel)+ylab("mean A value")
#$


################################ 3_2 ################################

library(ggplot2)
library(reshape2)

cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")

rV = function(n, mu, sigma, lambda){
  phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi1))
}

rW = function(n, mu, sigma, lambda){
  phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
```

```
  return(rexp(n, phi2))
}

jump_diffusion_sim_Yexp = function(mu, sigma, lambda, Tn, lambdaY, n){
  A = matrix(nrow = Tn, ncol = n)
  P = matrix(nrow = Tn, ncol = n)
  M = matrix(nrow = Tn, ncol = n)

  for(j in 1:n){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    P[1,j] = 0
    A[1,j] = 0
    M[1,j] = 0
    Mm1 = 0
    Am1 = 0


    for (i in 1:Tn){
      Y = rnorm(1, 0,1)
      P[i,j] =  V[i]-W[i]+Am1
      A[i,j] = Am1+V[i]-W[i]+Y
      Mm1 =  max(Mm1, Am1+V[i], A[i,j])
      M[i,j] = Mm1
      Am1 = A[i,j]
    }
  }
  return(list(P,A,M))
}

set.seed(23)
sigma = c(1,10,25,50,100,500)
Pdf = data.frame()
Adf = data.frame()
Mdf = data.frame()
Tn = 1000
n=1000
for (i in 1:length(sigma)){

  X = jump_diffusion_sim_Yexp(1, sigma[i], 1, Tn, 1,n)

  Pdf <- rbind(Pdf,
               data.frame(dataset=(paste("P", mu[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:Tn)))
  Adf <- rbind(Adf,
               data.frame(dataset=(paste("A", mu[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:Tn)))
  Mdf <- rbind(Mdf,
               data.frame(dataset=(paste("M", mu[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:Tn)))
}
Mdf$dataset <- as.factor(Mdf$dataset)
#$
Sigmalabs = c()
for (i in 1:length(sigma)){
  Sigmalabs = c(Sigmalabs,bquote(sigma==.(sigma[i])))
}
sigmaLabel = "\u03c3 Value"
#$
ggplot(Mdf, aes(x=TimeStep)) +
  geom_line(aes(y=obs, color=dataset))+
  scale_color_manual(values=c(cbP[10],cbP[3],cbP[5],cbP[6],cbP[7],cbP[8]),
                     labels=Sigmalabs)+
```

```
  labs(color=sigmaLabel)+
  ylab("mean M value")

############################## 3_3 ##############################

library(ggplot2)
library(reshape2)

cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")

rV = function(n, mu, sigma, lambda){
  phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi1))
}

rW = function(n, mu, sigma, lambda){
  phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi2))
}

jumpDiffusionSimYexp = function(mu, sigma, lambda, Tn, lambdaY, n){
  A = matrix(nrow = Tn, ncol = n)
  P = matrix(nrow = Tn, ncol = n)
  M = matrix(nrow = Tn, ncol = n)

  for(j in 1:n){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    P[1,j] = 0
    A[1,j] = 0
    M[1,j] = 0
    Mm1 = 0
    Am1 = 0
    l714V

    for (i in 1:Tn){
      Y = rnorm(1, 0,1)
      P[i,j] = V[i]-W[i]+Am1
      A[i,j] = Am1+V[i]-W[i]+Y
      Mm1 = max(Mm1, Am1+V[i], A[i,j])
      M[i,j] = Mm1
      Am1 = A[i,j]
    }
  }
  return(list(P,A,M))
}

set.seed(23)
lambda = c(0.1,0.5,1,2,5,10)
Pdf = data.frame()
Adf = data.frame()
Mdf = data.frame()
Tn = 1000
n=1000
for (i in 1:length(lambda)){

  X = jumpDiffusionSimYexp(1, 1, lambda[i], Tn, 1,n)

  Pdf <- rbind(Pdf,
               data.frame(dataset=(paste("P", lambda[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:
                  Tn)))
  Adf <- rbind(Adf,
```

```
                data.frame(dataset=(paste("A", lambda[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:
                    Tn)))
  Mdf <- rbind(Mdf,
                data.frame(dataset=(paste("M", lambda[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:
                    Tn)))
}
Adf$dataset <- as.factor(Adf$dataset)
#$
Lambdalabs = c()
for (i in 1:length(lambda)){
  Lambdalabs = c(Lambdalabs,bquote(lambda==.(lambda[i])))
}
lamdaLabel = "\u03bb Value"
#$
ggplot(Adf, aes(x=TimeStep)) +
  geom_line(aes(y=obs, color=dataset))+
  scale_color_manual(values=c(cbP[1],cbP[3],cbP[10],cbP[5],cbP[6],cbP[8]),
                     labels=Lambdalabs)+
  labs(color=lamdaLabel)
library(ggplot2)
library(reshape2)

cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")

rV = function(n, mu, sigma, lambda){
  phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi1))
}

rW = function(n, mu, sigma, lambda){
  phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi2))
}

jumpDiffusionSimYexp = function(mu, sigma, lambda, Tn, lambdaY, n){
  A = matrix(nrow = Tn, ncol = n)
  P = matrix(nrow = Tn, ncol = n)
  M = matrix(nrow = Tn, ncol = n)

  for(j in 1:n){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    P[1,j] = 0
    A[1,j] = 0
    M[1,j] = 0
    Mm1 = 0
    Am1 = 0


    for (i in 1:Tn){
      Y = rnorm(1, 0,1)
      P[i,j] =  V[i]-W[i]+Am1
      A[i,j] = Am1+V[i]-W[i]+Y
      Mm1 =  max(Mm1, Am1+V[i], A[i,j])
      M[i,j] = Mm1
      Am1 = A[i,j]
    }
  }
  return(list(P,A,M))
}

set.seed(23)
```

```
mu = c(0.1,0.5,1,2,5,10)
Pdf = data.frame()
Adf = data.frame()
Mdf = data.frame()
Tn = 1000
n=100
for (i in 1:length(mu)){

  X = jumpDiffusionSimYexp(mu[i], 1, 1, Tn, 2,n)

  Pdf <- rbind(Pdf,
               data.frame(dataset=(paste("P", mu[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:Tn))
                   )
  Adf <- rbind(Adf,
               data.frame(dataset=(paste("A", mu[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:Tn))
                   )
  Mdf <- rbind(Mdf,
               data.frame(dataset=(paste("M", mu[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:Tn))
                   )
}
Adf$dataset <- as.factor(Adf$dataset)

Mulabs = c()
for (i in 1:length(mu)){
  Mulabs = c(Mulabs,bquote(mu==.(mu[i])))
}
muLabel = "\u03bc Value"
#$

ggplot(Adf, aes(x=TimeStep)) +
  geom_line(aes(y=obs, color=dataset))+
  scale_color_manual(values=c(cbP[1],cbP[3],cbP[10],cbP[5],cbP[6],cbP[8]),
                     labels=Mulabs)+
  labs(color=muLabel)+ylab("mean A value")
#$


############################## 3_2 ##############################

library(ggplot2)
library(reshape2)

cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")

rV = function(n, mu, sigma, lambda){
  phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi1))
}

rW = function(n, mu, sigma, lambda){
  phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi2))
}

jump_diffusion_sim_Yexp = function(mu, sigma, lambda, Tn, lambdaY, n){
  A = matrix(nrow = Tn, ncol = n)
  P = matrix(nrow = Tn, ncol = n)
  M = matrix(nrow = Tn, ncol = n)

  for(j in 1:n){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

```

```r
    P[1,j] = 0
    A[1,j] = 0
    M[1,j] = 0
    Mm1 = 0
    Am1 = 0


    for (i in 1:Tn){
      Y = rnorm(1, 0,1)
      P[i,j] =  V[i]-W[i]+Am1
      A[i,j] = Am1+V[i]-W[i]+Y
      Mm1 =  max(Mm1, Am1+V[i], A[i,j])
      M[i,j] = Mm1
      Am1 = A[i,j]
    }
  }
  return(list(P,A,M))
}

set.seed(23)
sigma = c(1,10,25,50,100,500)
Pdf = data.frame()
Adf = data.frame()
Mdf = data.frame()
Tn = 1000
n=1000
for (i in 1:length(sigma)){

  X = jump_diffusion_sim_Yexp(1, sigma[i], 1, Tn, 1,n)

  Pdf <- rbind(Pdf,
               data.frame(dataset=(paste("P", mu[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:Tn)))
  Adf <- rbind(Adf,
               data.frame(dataset=(paste("A", mu[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:Tn)))
  Mdf <- rbind(Mdf,
               data.frame(dataset=(paste("M", mu[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:Tn)))
}
Mdf$dataset <- as.factor(Mdf$dataset)
#$
Sigmalabs = c()
for (i in 1:length(sigma)){
  Sigmalabs = c(Sigmalabs,bquote(sigma==.(sigma[i])))
}
sigmaLabel = "\u03c3 Value"
#$
ggplot(Mdf, aes(x=TimeStep)) +
  geom_line(aes(y=obs, color=dataset))+
  scale_color_manual(values=c(cbP[10],cbP[3],cbP[5],cbP[6],cbP[7],cbP[8]),
                     labels=Sigmalabs)+
  labs(color=sigmaLabel)+
  ylab("mean M value")

############################## 3_3 ###############################

library(ggplot2)
library(reshape2)

cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")

rV = function(n, mu, sigma, lambda){
  phi1 = -mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
```

```
  return(rexp(n, phi1))
}

rW = function(n, mu, sigma, lambda){
  phi2 = mu/(sigma^2)+sqrt(mu^2/sigma^4+2*lambda/sigma^2)
  return(rexp(n, phi2))
}

jumpDiffusionSimYexp = function(mu, sigma, lambda, Tn, lambdaY, n){
  A = matrix(nrow = Tn, ncol = n)
  P = matrix(nrow = Tn, ncol = n)
  M = matrix(nrow = Tn, ncol = n)

  for(j in 1:n){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    P[1,j] = 0
    A[1,j] = 0
    M[1,j] = 0
    Mm1 = 0
    Am1 = 0
    1714V

    for (i in 1:Tn){
      Y = rnorm(1, 0,1)
      P[i,j] =  V[i]-W[i]+Am1
      A[i,j] = Am1+V[i]-W[i]+Y
      Mm1 =  max(Mm1, Am1+V[i], A[i,j])
      M[i,j] = Mm1
      Am1 = A[i,j]
    }
  }
  return(list(P,A,M))
}

set.seed(23)
lambda = c(0.1,0.5,1,2,5,10)
Pdf = data.frame()
Adf = data.frame()
Mdf = data.frame()
Tn = 1000
n=1000
for (i in 1:length(lambda)){

  X = jumpDiffusionSimYexp(1, 1, lambda[i], Tn, 1,n)

  Pdf <- rbind(Pdf,
               data.frame(dataset=(paste("P", lambda[i])), obs=rowMeans(X[[1]]), TimeStep = seq(1:
                   Tn)))
  Adf <- rbind(Adf,
               data.frame(dataset=(paste("A", lambda[i])), obs=rowMeans(X[[2]]), TimeStep = seq(1:
                   Tn)))
  Mdf <- rbind(Mdf,
               data.frame(dataset=(paste("M", lambda[i])), obs=rowMeans(X[[3]]), TimeStep = seq(1:
                   Tn)))
}
Adf$dataset <- as.factor(Adf$dataset)
#$
Lambdalabs = c()
for (i in 1:length(lambda)){
  Lambdalabs = c(Lambdalabs,bquote(lambda==.(lambda[i])))
}
lamdaLabel = "\u03bb Value"
```

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

B.0.3 Jump size distribution

```r

# IMPORTS

library(ggplot2)

# ------------- PROJECT : Levy Processes [IMPLEMENTATION] ----------------------

# ---------- ########## EXERCISE 4 ############ ----------------


# ---------- ########## COLOR SCHEME ############ ----------------

cbP <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
         "#CC79A7", "#666666", "#004f7c", "#000000", "#999999")

# ---------- ########## DISTRIBUTIONS ############ ----------------

##### PDF's #######

pareto_pdf <- function(beta = 1, k = 1, x){
  return(k * ((beta^k)/(x^(k + 1))))
}

exp_pdf <- function(lambda = 1/2, x){
  return(lambda*exp(-lambda*x))
}

erlang_pdf <- function(k = 1 ,lambda = 1/2, x){

  numerator <- (lambda^k) * (x^(k - 1)) * exp(-lambda*x)
  denominator <- factorial(k - 1)
  return(numerator/denominator)

}

hyperexp_pdf <- function(p1 = 0.8, lambda1 = 3/2, p2 = 0.2, lambda2 = 2/3, x){

  term1 <- p1 * lambda1 * exp(-lambda1 * x)
  term2 <- p2 * lambda2 * exp(-lambda2 * x)
  return(term1 + term2)

}

### GENERATORS #####

##### GENERATORS #########


# 1. --------- Pareto Distribution ------------

# Mean:
# Variance:
```

```
# Description:

gen_pareto <- function(n = 1000, beta = 1, k = 2.05){

  U <- runif(n)
  X <- beta * U^(-1/k)
  return(X)
}

# 2. --------- Normal Distribution ------------

# Mean: mu
# Variance: sigma^2

# Description:

gen_normal <- function(n = 1, mean = 0, sd = 1) {

  return(rnorm(n, mean = mean, sd = sd))

}


# 3. --------- Exponential ------------

gen_exp <- function(n = 1, rate = 1){

  return(rexp(n, rate = rate))

}

# 4. --------- Hyperexponential ------------

gen_hyperexp <- function(n = 1, rate1 = 1, p1 = 1/2, rate2 = 1, p2 = 1/2){

  output <- c()

  for (i in 1:n) {

    U_1 <- runif(1, 0, 1)

    if (U_1 <= p1) {

      output[i] <- rexp(1, rate = rate1)

    } else {

      output[i] <- rexp(1, rate = rate2)

    }
  }

  return(output)

}

# 5. --------- Erlang ------------

gen_erlang = function(n = 1, lambda = 3, k = 2){

  output = rep(0, n)

  for (i in 1:k) {
    output = output + rexp(n, lambda)
```

```r
  }

  return(output)

}

#### OTHER IMPLEMENTATION NEEDED ######

set.seed(23)

generate_V <- function(n = 1000, mu = 1, sigma = 0.2, lambda = 2){

  theta_1 <- (-1)*(mu/(sigma^2)) + sqrt((mu^2/(sigma^4)) + ((2*lambda)/(sigma^2)))
  V <- rexp(n, rate = theta_1)
  return(V)

}

generate_W <- function(n = 1000, mu = 1, sigma = 0.2, lambda = 2){

  theta_2 <- (mu/(sigma^2)) + sqrt((mu^2/(sigma^4)) + ((2*lambda)/(sigma^2)))
  W <- rexp(n, rate = theta_2)
  return(W)

}

generate_Y <- function(n = 1000, distribution = "pareto", set_mean = 3) {


  if (distribution == "pareto"){

    k = 2.05

    #Determines the mean
    beta = set_mean * (k - 1) / k

    return(gen_pareto(n = n, beta = beta, k = k))

  }

  if (distribution == "exponential"){

    # Determines the mean
    rate = 1/(set_mean)

    return(gen_exp(n = n, rate = rate))

  }

  if (distribution == "hyperexponential") {

    p_1 = 0.8
    p_2 = (1 - p_1)
    rate_1 = set_mean
    # Determines the mean
    rate_2 = (1 - p_1) / (set_mean - p_1*(1/rate_1))

    return(gen_hyperexp(n = n, rate1 = rate_1, p1 = p_1, rate2 = rate_2, p2 = p_2))

  }

  if (distribution == "erlang"){

    k <- 3
    lambda <- k / set_mean
```

```
    # Determines the mean

    return(gen_erlang(n = n, lambda = lambda, k = k))

  }

}

algorithm_1 <- function(jumps = 1000, distribution_y = "normal", mean_y = 3){

  V <- generate_V(n = (jumps + 1), mu = 1, sigma = 0.2, lambda = 2)
  W <- generate_W(n = (jumps + 1), mu = 1, sigma = 0.2, lambda = 2)
  Y <- generate_Y(n = (jumps + 1), distribution = distribution_y, set_mean = mean_y)

  P <- rep(0, (jumps + 1))
  A <- rep(0, (jumps + 1))
  M <- rep(0, (jumps + 1))
  time <- c(0: jumps)

  for (j in 1:jumps){

    i <- j + 1 # Displace s.t. the indexing looks like in the algorithm

    P[i] <- A[i - 1] + (V[i] - W[i])
    A[i] <- A[i - 1] + (V[i] - W[i]) + Y[i]
    M[i] <- max(c(M[i - 1], A[i - 1] + V[i], A[i - 1] + (V[i] - W[i]) + Y[i]))

  }

  result <- data.frame(time, P, A, M)

}

algorithm_1_extended <- function(jumps = 1000, repeats = 100, distribution_y = "normal", mean_y =
    3){

  P_n <- c()
  A_n <- c()
  M_n <- c()
  Y_mean <- c()

  for (k in 1:repeats) {

    P <- rep(0, jumps)
    A <- rep(0, jumps)
    M <- rep(0, jumps)

    V <- generate_V(n = (jumps + 1), mu = 1)
    W <- generate_W(n = (jumps + 1), mu = 1)
    Y <- generate_Y(n = (jumps + 1), distribution = distribution_y, set_mean = mean_y)

    for (j in 1:jumps){

      i <- j + 1 # Displace s.t. the indexing looks like in the algorithm

      P[i] <- A[i - 1] + (V[i] - W[i])
      A[i] <- A[i - 1] + (V[i] - W[i]) + Y[i]
      M[i] <- max(c(M[i - 1], A[i - 1] + V[i], A[i - 1] + (V[i] - W[i]) + Y[i]))

    }

    P <- P[-1] # Remove P_0 = 0
    A <- A[-1] # Remove A_0 = 0
    M <- M[-1] # Remove M_0 = 0

```

```r
    P_n[k] <- P[jumps]
    A_n[k] <- A[jumps]
    M_n[k] <- M[jumps]
    Y_mean[k] <- mean(Y)

  }

  return(data.frame(P_n, A_n, M_n, Y_mean))

}

# ------------- PROJECT : Levy Processes [ANALYSIS] ----------------------


# ---------- ########## EXERCISE 4 ############ ----------------

set.seed(23)
SET_MEAN <- 3

### 1) Plotting Distributions

### OVERALL #####

### DEFINE DATA-POINTS #####

# We ensure that they all have mean mu = 5

x <- seq(0, 15, 0.1)

set_mean <- SET_MEAN

# PARETO

pareto_k <- 2.05
pareto_beta <- set_mean * (pareto_k - 1) / pareto_k
y_pareto <- pareto_pdf(beta = pareto_beta, k = pareto_k, x)

variance_pareto = (pareto_k * pareto_beta^2) / ((pareto_k - 2) * (pareto_k - 1)^2)
print(variance_pareto)

# EXP

exp_lambda <- 1 / set_mean
y_exp <- exp_pdf(lambda = exp_lambda, x)

variance_exp <- 1 / exp_lambda^2
print(variance_exp)

# ERLANG

erlang_k = 3
erlang_lambda <- k / set_mean
y_erl <- erlang_pdf(k = erlang_k,lambda = erlang_lambda, x)

variance_erlang <- erlang_k / erlang_lambda^2
print(variance_erlang)

# HYPER

exp_p1 = 0.8
exp_p2 = (1 - p_1)
exp_rate_1 = SET_MEAN
exp_rate_2 = (1 - p_1) / (set_mean - p_1*(1/rate_1))
y_hypexp <- hyperexp_pdf(p1 = exp_p1, lambda1 = exp_rate_1, p2 = exp_p2, lambda2 = exp_rate_2, x)

```

```r
df <- data.frame(x, y_pareto, y_exp, y_erl, y_hypexp)

p <- ggplot(data = df, mapping = aes(x = x)) +
    geom_line(mapping=aes(y=y_pareto, color = "Pareto")) +
    geom_line(mapping=aes(y=y_exp, color = "Exp")) +
    geom_line(mapping=aes(y=y_erl, color = "Erlang")) +
    geom_line(mapping=aes(y=y_hypexp, color = "Hypexp")) +
    scale_color_manual(values = c("Pareto" = cbP[1], 'Exp' = cbP[2], 'Erlang' = cbP[3], 'Hypexp' = cbP[4])) +
    labs(title = sprintf("Distributions with mean = %s", SET_MEAN), y = "p(x)", x= "x", color = '') +
    ylim(c(0, 0.5))

p

variance_hyper <- (exp_p1 * 2 / exp_rate_1^2 + exp_p2 * 2 / exp_rate_2^2) - (exp_p1 / exp_rate_1 + exp_p2 / exp_rate_2)^2
print(variance_hyper)

### TAILS #####

x <- seq(8, 40, 0.1)

set_mean <- SET_MEAN

# PARETO

y_pareto <- pareto_pdf(beta = pareto_beta, k = pareto_k, x)

# EXP

y_exp <- exp_pdf(lambda = exp_lambda, x)

# ERLANG

y_erl <- erlang_pdf(k = erlang_k,lambda = erlang_lambda, x)

# HYPER

y_hypexp <- hyperexp_pdf(p1 = exp_p1, lambda1 = exp_rate_1, p2 = exp_p2, lambda2 = exp_rate_2, x)

df <- data.frame(x, y_pareto, y_exp, y_erl, y_hypexp)

p <- ggplot(data = df, mapping = aes(x = x)) +
  geom_line(mapping=aes(y=y_pareto, color = "Pareto")) +
  geom_line(mapping=aes(y=y_exp, color = "Exp")) +
  geom_line(mapping=aes(y=y_erl, color = "Erlang")) +
  geom_line(mapping=aes(y=y_hypexp, color = "Hypexp")) +
  scale_color_manual(values = c("Pareto" = cbP[1], 'Exp' = cbP[2], 'Erlang' = cbP[3], 'Hypexp' = cbP[4])) +
  labs(title = sprintf("Distributions with mean = %s", SET_MEAN), y = "p(x)", x= "x", color = '') +
  ylim(c(0, 0.0025))

p

### 2) Plotting Runs

# -------- PLOTTING A SINGLE RUN --------------

data_line_plot_pareto <- algorithm_1(jumps = 1000, mean_y = SET_MEAN, distribution_y = "pareto")
data_line_plot_erlang <- algorithm_1(jumps = 1000, mean_y = SET_MEAN, distribution_y = "erlang")
data_line_plot_exponential <- algorithm_1(jumps = 1000, mean_y = SET_MEAN, distribution_y = "exponential")
```

```
data_line_plot_hyperexponential <- algorithm_1(jumps = 1000, mean_y = SET_MEAN, distribution_y = "
    hyperexponential")

low_x <- 420
upper_x <- 450

low_y <- data_line_plot_pareto$P[low_x + 1]
upper_y <- data_line_plot_pareto$M[upper_x]

p_pareto = ggplot() +
  geom_line(data = data_line_plot_pareto, aes(x = time, y = P, colour = "Prior")) +
  geom_line(data = data_line_plot_pareto, aes(x = time, y = A, colour = "After")) +
  geom_step(data = data_line_plot_pareto, aes(x = time, y = M, colour = "Max"), direction = "vh")
    +
  xlim(low_x, upper_x) +
  ylim(low_y, upper_y) +
  scale_colour_manual("Pareto", values = c("Prior"=cbP[1], "After"=cbP[3], "Max"=cbP[8])) +
  xlab('time') +
  ylab('value') +
  labs(title = sprintf("Pareto with mean = %s", SET_MEAN))

print(p_pareto)

low_x <- 420
upper_x <- 450

low_y <- data_line_plot_erlang$P[low_x + 1]
upper_y <- data_line_plot_erlang$M[upper_x]

p_erlang = ggplot() +
  geom_line(data = data_line_plot_erlang, aes(x = time, y = P, colour = "Prior")) +
  geom_line(data = data_line_plot_erlang, aes(x = time, y = A, colour = "After")) +
  geom_step(data = data_line_plot_erlang, aes(x = time, y = M, colour = "Max"), direction = "vh")
    +
  xlim(low_x, upper_x) +
  ylim(low_y, upper_y) +
  scale_colour_manual("", values = c("Prior"=cbP[1], "After"=cbP[3], "Max"=cbP[8])) +
  xlab('time') +
  ylab('value') +
  labs(title = sprintf("Erlang with mean = %s", SET_MEAN))

print(p_erlang)

low_x <- 420
upper_x <- 450

low_y <- data_line_plot_exponential$P[low_x + 1]
upper_y <- data_line_plot_exponential$M[upper_x]

p_exponential = ggplot() +
  geom_line(data = data_line_plot_exponential, aes(x = time, y = P, colour = "Prior")) +
  geom_line(data = data_line_plot_exponential, aes(x = time, y = A, colour = "After")) +
  geom_step(data = data_line_plot_exponential, aes(x = time, y = M, colour = "Max"), direction = "
    vh") +
  xlim(low_x, upper_x) +
  ylim(low_y, upper_y) +
  scale_colour_manual("", values = c("Prior"=cbP[1], "After"=cbP[3], "Max"=cbP[8])) +
  xlab('time') +
  ylab('value') +
  labs(title = sprintf("Exponential with mean = %s", SET_MEAN))

print(p_exponential)

low_x <- 420
upper_x <- 450
```

```r

low_y <- data_line_plot_hyperexponential$P[low_x + 1]
upper_y <- data_line_plot_hyperexponential$M[upper_x]

p_hyperexponential = ggplot() +
  geom_line(data = data_line_plot_hyperexponential, aes(x = time, y = P, colour = "Prior")) +
  geom_line(data = data_line_plot_hyperexponential, aes(x = time, y = A, colour = "After")) +
  geom_step(data = data_line_plot_hyperexponential, aes(x = time, y = M, colour = "Max"),
      direction = "vh") +
  xlim(low_x, upper_x) +
  ylim(low_y, upper_y) +
  scale_colour_manual("", values = c("Prior"=cbP[1], "After"=cbP[3], "Max"=cbP[8])) +
  xlab('time') +
  ylab('value') +
  labs(title = sprintf("Hyperexponential with mean = %s", SET_MEAN))

print(p_hyperexponential)

# 3) Defining algorithm to capture the value of the 1000th step in 100 runs


# -------- GENERATE RESULTS ------------

results_pareto <- algorithm_1_extended(jump = 1000, distribution_y = "pareto", mean_y = SET_MEAN)

results_hyper <- algorithm_1_extended(jump = 1000, distribution_y = "hyperexponential", mean_y =
    SET_MEAN)

results_exponential <- algorithm_1_extended(jump = 1000, distribution_y = "exponential", mean_y =
    SET_MEAN)

results_erlang <- algorithm_1_extended(jump = 1000, distribution_y = "erlang", mean_y = SET_MEAN)


# ----------- HISTOGRAM OF P -----------

DF_P <- rbind(data.frame(dataset=1, obs=results_pareto$P_n),
              data.frame(dataset=2, obs=results_hyper$P_n),
              data.frame(dataset=3, obs=results_exponential$P_n),
              data.frame(dataset=4, obs=results_erlang$P_n))

DF_P$dataset <- as.factor(DF_P$dataset)

ggplot(DF_P, aes(x=obs, fill=dataset)) +
  geom_histogram(binwidth=100, colour="black", position="dodge") +
  scale_fill_manual(breaks=1:4, values=c(cbP[1],cbP[3],cbP[8], cbP[2]),
                    labels=c("Pareto", "Hyper", "Exponential", "Erlang")) + labs(title = sprintf("
                        1000th P with mean = %s", SET_MEAN), x = "values", y = "counts", fill="")

# ----------- HISTOGRAM OF A -----------

DF_A <- rbind(data.frame(dataset=1, obs=results_pareto$A_n),
              data.frame(dataset=2, obs=results_hyper$A_n),
              data.frame(dataset=3, obs=results_exponential$A_n),
              data.frame(dataset=4, obs=results_erlang$A_n))

DF_A$dataset <- as.factor(DF_A$dataset)

ggplot(DF_A, aes(x=obs, fill=dataset)) +
  geom_histogram(binwidth=100, colour="black", position="dodge") +
  scale_fill_manual(breaks=1:4, values=c(cbP[1],cbP[3],cbP[8], cbP[2]),
                    labels=c("Pareto", "Hyper", "Exponential", "Erlang")) + labs(title = sprintf("
                        1000th A with mean = %s", SET_MEAN), x = "values", y = "counts", fill="")


```

```
# ----------- HISTOGRAM AND DENSITY OF M -----------

DF_M <- rbind(data.frame(dataset=1, obs=results_pareto$M_n),
              data.frame(dataset=2, obs=results_hyper$M_n),
              data.frame(dataset=3, obs=results_exponential$M_n),
              data.frame(dataset=4, obs=results_erlang$M_n))

DF_M$dataset <- as.factor(DF_M$dataset)

ggplot(DF_M, aes(x=obs, fill=dataset)) +
  geom_histogram(binwidth=100, colour="black", position="dodge") +
  scale_fill_manual(breaks=1:4, values=c(cbP[1],cbP[3],cbP[8], cbP[2],cbP[5]),
                    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

```

########## First passage probabilities ##########
jump_diffusion_max_sim_Yexp = function(mu, sigma, lambda, Tn, lambdaY, internal_rotation){
  M_matrix = matrix(0, internal_rotation, Tn)

  for(j in 1:internal_rotation){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    Am1 = 0
    A = 0
    M = c(0)

    for (i in 1:Tn){
      Y = rexp(1, lambdaY)
      A = Am1+V[i]-W[i]+Y
      M = c(M, max(M[i], Am1+V[i], A))
      Am1 = A
    }

    M_matrix[j,] = M[-1]
  }
  return(M_matrix)
}
jump_diffusion_max_sim_Ypareto = function(mu, sigma, lambda, Tn, betaY, kY, internal_rotation){
  M_matrix = matrix(0, internal_rotation, Tn)

  for(j in 1:internal_rotation){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    Am1 = 0
    A = 0
    M = c(0)

    for (i in 1:Tn){
      Y = rPareto(1, betaY, kY)
      A = Am1+V[i]-W[i]+Y
      M = c(M, max(M[i], Am1+V[i], A))
      Am1 = A
    }

```

```
    M_matrix[j,] = M[-1]
  }
  return(M_matrix)
}
jump_diffusion_max_sim_Yhyp = function(mu, sigma, lambda, Tn, probsY, lambdasY, internal_rotation)
    {
  M_matrix = matrix(0, internal_rotation, Tn)

  for(j in 1:internal_rotation){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    Am1 = 0
    A = 0
    M = c(0)

    for (i in 1:Tn){
      Y = rhyper2(1, probsY, lambdasY)
      A = Am1+V[i]-W[i]+Y
      M = c(M, max(M[i], Am1+V[i], A))
      Am1 = A
    }

    M_matrix[j,] = M[-1]
  }
  return(M_matrix)
}
jump_diffusion_max_sim_Yerlang = function(mu, sigma, lambda, Tn, lambdaY, kY, internal_rotation){
  M_matrix = matrix(0, internal_rotation, Tn)

  for(j in 1:internal_rotation){

    W = rW(Tn, mu, sigma, lambda)
    V = rV(Tn, mu, sigma, lambda)

    Am1 = 0
    A = 0
    M = c(0)

    for (i in 1:Tn){
      Y = rErlang(1, lambdaY, kY)
      A = Am1+V[i]-W[i]+Y
      M = c(M, max(M[i], Am1+V[i], A))
      Am1 = A
    }

    M_matrix[j,] = M[-1]
  }
  return(M_matrix)
}

# Confidence intervals for a-contours
conf_int_exp = function(mu, sigma, lambda, Tn, lambdaY, internal_rot, external_rot, a, alpha,
    ploting = TRUE){
  internal_rotation = internal_rot
  external_rotation = external_rot
  Ms = array(0, c(internal_rot,Tn,external_rot))
  M_prob = array(0,c(external_rotation,Tn,length(a)))

  elapsed_time = 0
  for (j in 1:external_rotation) {
    time = Sys.time()
    Ms[,,j]=jump_diffusion_max_sim_Yexp(mu,sigma,lambda,Tn,lambdaY,internal_rotation)
    end_time = as.numeric(Sys.time()-time)
```

```
    elapsed_time = elapsed_time + end_time
    print(paste("External round number:",j,"/",external_rotation))
    print(paste("Elapsed time:",round(elapsed_time,2),"sec","; Expected total time", round(end_time*(external_rotation-j)+elapsed_time,2),"sec"))
  }

  for (j in 1:length(a)){
    for(i in 1:external_rot){
      M_prob[i,,j] = colSums(Ms[,,i]>a[j])/internal_rotation
    }
  }

  means = matrix(0,Tn,length(a))
  sds = matrix(0,Tn,length(a))
  for (i in 1:length(a)) {
    means[,i] = colMeans(M_prob[,,i])
    sds[,i] = colSds(M_prob[,,i])
  }

  if(alpha<0.5){
    quant = abs(qnorm(alpha/2))
  }else{quant = abs(qnorm((1-alpha)/2))}

  high = matrix(0, Tn, length(a))
  low = matrix(0, Tn, length(a))
  for (i in 1:length(a)){
    high[,i] = means[,i]+quant*sds[,i]/sqrt(external_rotation)
    low[,i] = means[,i]-quant*sds[,i]/sqrt(external_rotation)
  }

  return(list(upper = high, lower = low, mean = means))
}
conf_int_pareto = function(mu, sigma, lambda, Tn,  betaY, kY, internal_rot, external_rot, a, alpha, ploting = TRUE){
  internal_rotation = internal_rot
  external_rotation = external_rot
  Ms = array(0, c(internal_rot,Tn,external_rot))
  M_prob = array(0,c(external_rotation,Tn,length(a)))

  elapsed_time = 0
  for (j in 1:external_rotation) {
    time = Sys.time()
    Ms[,,j]=jump_diffusion_max_sim_Ypareto(mu,sigma,lambda,Tn,betaY, kY,internal_rotation)
    end_time = as.numeric(Sys.time()-time)
    elapsed_time = elapsed_time + end_time
    print(paste("External round number:",j,"/",external_rotation))
    print(paste("Elapsed time:",round(elapsed_time,2),"sec","; Expected total time", round(end_time*(external_rotation-j)+elapsed_time,2),"sec"))
  }

  for (j in 1:length(a)){
    for(i in 1:external_rot){
      M_prob[i,,j] = colSums(Ms[,,i]>a[j])/internal_rotation
    }
  }

  means = matrix(0,Tn,length(a))
  sds = matrix(0,Tn,length(a))
  for (i in 1:length(a)) {
    means[,i] = colMeans(M_prob[,,i])
    sds[,i] = colSds(M_prob[,,i])
  }

  if(alpha<0.5){
    quant = abs(qnorm(alpha/2))
```

```
  }else{quant = abs(qnorm((1-alpha)/2))}

  high = matrix(0, Tn, length(a))
  low = matrix(0, Tn, length(a))
  for (i in 1:length(a)){
    high[,i] = means[,i]+quant*sds[,i]/sqrt(external_rotation)
    low[,i] = means[,i]-quant*sds[,i]/sqrt(external_rotation)
  }

  return(list(upper = high, lower = low, mean = means))
}
conf_int_hyp = function(mu, sigma, lambda, Tn,  probsY, lambdasY, internal_rot, external_rot, a,
    alpha, ploting = TRUE){
  internal_rotation = internal_rot
  external_rotation = external_rot
  Ms = array(0, c(internal_rot,Tn,external_rot))
  M_prob = array(0,c(external_rotation,Tn,length(a)))

  elapsed_time = 0
  for (j in 1:external_rotation) {
    time = Sys.time()
    Ms[,,j]=jump_diffusion_max_sim_Yhyp(mu,sigma,lambda,Tn,probsY, lambdasY,internal_rotation)
    end_time = as.numeric(Sys.time()-time)
    elapsed_time = elapsed_time + end_time
    print(paste("External round number:",j,"/",external_rotation))
    print(paste("Elapsed time:",round(elapsed_time,2),"sec","; Expected total time", round(end_
        time*(external_rotation-j)+elapsed_time,2),"sec"))
  }

  for (j in 1:length(a)){
    for(i in 1:external_rot){
      M_prob[i,,j] = colSums(Ms[,,i]>a[j])/internal_rotation
    }
  }

  means = matrix(0,Tn,length(a))
  sds = matrix(0,Tn,length(a))
  for (i in 1:length(a)) {
    means[,i] = colMeans(M_prob[,,i])
    sds[,i] = colSds(M_prob[,,i])
  }

  if(alpha<0.5){
    quant = abs(qnorm(alpha/2))
  }else{quant = abs(qnorm((1-alpha)/2))}

  high = matrix(0, Tn, length(a))
  low = matrix(0, Tn, length(a))
  for (i in 1:length(a)){
    high[,i] = means[,i]+quant*sds[,i]/sqrt(external_rotation)
    low[,i] = means[,i]-quant*sds[,i]/sqrt(external_rotation)
  }

  return(list(upper = high, lower = low, mean = means))
}
conf_int_erlang = function(mu, sigma, lambda, Tn,  lambdaY, kY, internal_rot, external_rot, a,
    alpha, ploting = TRUE){
  internal_rotation = internal_rot
  external_rotation = external_rot
  Ms = array(0, c(internal_rot,Tn,external_rot))
  M_prob = array(0,c(external_rotation,Tn,length(a)))

  elapsed_time = 0
  for (j in 1:external_rotation) {
    time = Sys.time()
```

```
    Ms[,,j]=jump_diffusion_max_sim_Yerlang(mu,sigma,lambda,Tn,lambdaY, kY,internal_rotation)
    end_time = as.numeric(Sys.time()-time)
    elapsed_time = elapsed_time + end_time
    print(paste("External round number:",j,"/",external_rotation))
    print(paste("Elapsed time:",round(elapsed_time,2),"sec","; Expected total time", round(end_time*(external_rotation-j)+elapsed_time,2),"sec"))
  }

  for (j in 1:length(a)){
    for(i in 1:external_rot){
      M_prob[i,,j] = colSums(Ms[,,i]>a[j])/internal_rotation
    }
  }

  means = matrix(0,Tn,length(a))
  sds = matrix(0,Tn,length(a))
  for (i in 1:length(a)) {
    means[,i] = colMeans(M_prob[,,i])
    sds[,i] = colSds(M_prob[,,i])
  }

  if(alpha<0.5){
    quant = abs(qnorm(alpha/2))
  }else{quant = abs(qnorm((1-alpha)/2))}

  high = matrix(0, Tn, length(a))
  low = matrix(0, Tn, length(a))
  for (i in 1:length(a)){
    high[,i] = means[,i]+quant*sds[,i]/sqrt(external_rotation)
    low[,i] = means[,i]-quant*sds[,i]/sqrt(external_rotation)
  }

  return(list(upper = high, lower = low, mean = means))
}

# Plotting function
plotting = function(data, alow, ahigh, by){
  by = (ahigh-alow)/(by-1)
  plot(0,0,ylim = c(0,1), xlim = c(1,1000), type = "l", main =
         substitute(paste("First passage probability for a between ",alow," to ", ahigh,", by ",by),
                    list(alow = alow, ahigh = ahigh, by = by)),
       ylab = "Probability", xlab = "Arrival")

  locx = c()
  locy = c()
  labels = c()
  for (i in 1:ncol(data$mean)) {
    lines(1:1000,data$mean[,i], lty = 1)
    lines(1:1000,data$upper[,i], lty = 2)
    lines(1:1000,data$lower[,i], lty = 2)
    #locx = c(locx, 1090)
    #locy = c(locy, data$mean[,i])
    #labels = c(labels, paste("a = ", (alow+(i-1)*by)))
  }

  legend("topleft", legend = c("Mean", "Confidence interval"), col = c("black", "black"), lty = 1:2)
  #text(locx,locy,labels)
}


Exp_confs = conf_int_exp(mu = 0, sigma = 5, lambda = 1, Tn = 1000, lambdaY = 1/5, internal_rot = 100, external_rot = 100, a = seq(1000,3000, length.out = 10), alpha = 0.05)
```

```
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)
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)
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)


# All pretty similar with mean 5.
par(mfrow = c(1,1))
plotting(Exp_confs,1000,3000, 10)
plotting(Pareto_confs,1000,3000, 10)
plotting(Erlang_confs,1000,3000, 10)
plotting(Hyp_confs,1000,3000, 10)


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)
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)
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)
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)
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)
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)
Pareto_confs205 = Pareto_confs

# We see a heavy tail for small values of k results in different graphs. (All with mean 5)
plotting(Pareto_confs105,1000,3000, 10)
plotting(Pareto_confs110,1000,3000, 10)
plotting(Pareto_confs115,1000,3000, 10)
plotting(Pareto_confs125,1000,3000, 10)
plotting(Pareto_confs150,1000,3000, 10)
plotting(Pareto_confs175,1000,3000, 10)
plotting(Pareto_confs205,1000,3000, 10)



# Only brownian motion without drift
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)

# Hyper
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)

plotting(Hyp_confs_pareto,1000,3000,10)
```

B.0.5 European Call option

```
####### Ex 6 - European option valuation #########
set.seed(69)
exercise_time = 10
strike = 40
sigma = 0.3
rf = 0.05
S0 = 40
runs = 100000

# Rate
R= (rf-1/2*sigma^2)*exercise_time

# Standard deviation
SD = sigma*sqrt(exercise_time)

# Stock price at T
STS = S0 * exp(R+SD*rnorm(runs,0,1))


## Crude
V = pmax(STS-strike,0)*exp(-rf*exercise_time)
mean(V)
sd(V)
sd(V)/sqrt(runs)*1.96


## Control
Y = STS
mu_Y = rep(exp(rf*exercise_time)*S0, runs)
c = -cov(V, STS)/var(STS)

VC = V+c*(Y-mu_Y)
mean(VC)
sd(VC)
sd(VC)/sqrt(runs)*1.96


## Black-Scholes
BlackSch = function(S0, strike, SD, rf,exercise_time,initT){
  # rf = risk free rate

  d1 = (log(S0/strike)+(rf+SD^2*1/2)*(exercise_time-initT))/(SD*sqrt(exercise_time-initT))
  d2 = d1 - SD*sqrt(exercise_time-initT)
  cost = S0*pnorm(d1,mean=0,sd = 1)-strike*exp(-rf*(exercise_time-initT))*pnorm(d2,mean=0,sd = 1)
  return(cost)
  #print(paste('The cost is',cost))
}

# tested with results from:
# http://www.cliffordang.com/models/optionsMC.pdf

BlackSch(S0=S0,strike=strike,SD=sigma,rf=rf,exercise_time=exercise_time,initT=0)
```