The Effect of Boundary Constraints on Markov-Modulated Diffusion Processes

Lachlan Bridges

Supervised by Dr Giang Nguyen

The University of Adelaide

Vacation Research Scholarships are funded jointly by the Department of Education and Training and the Australian Mathematical Sciences Institute.
Contents

1 Abstract ................................. 1
2 Introduction ................................ 1
3 Preliminaries ............................ 2
4 Simulations .............................. 3
  4.1 Unmodulated diffusion processes ........................................ 3
  4.2 Markov Chains ......................................................... 5
  4.3 Markov-modulated diffusion processes .................................. 7
5 Occupation Time .......................... 9
6 Boundary Conditions ...................... 13
  6.1 Absorbing Boundaries .................................................. 13
  6.2 Instantaneous Jump Boundaries ........................................ 17
  6.3 Reflected Boundaries .................................................. 19
7 Applications to Finance ................. 21
  7.1 Black Scholes Model ................................................... 21
  7.2 Markov-modulated Model (McKinlay) .................................. 22
8 Conclusion ................................ 24
Appendix - MATLAB Code .................. 27
1 Abstract

Markov-modulated diffusion processes are diffusion processes in which we allow the drift and diffusion functions to change depending on an underlying Markov chain. These processes have uses in many different fields, including ecology and finance.

For example, in finance the volatility of a stock may be considered to have different “modes”: high and low volatility. Markov-modulated diffusion processes allow us to model the price of the stock as a diffusion process, with random switching between these two modes of volatility.

Boundary conditions are useful in many real world applications, allowing the model to be restricted to certain values within these boundaries, such as restricting a diffusion process modelling populations to being positive. Different types of boundary conditions are possible, such as absorbing, reflecting and instantaneous-jump boundaries, each of which may arise in different real world applications.

In this report, we use numerical methods to investigate the different boundary conditions which can be applied to Markov-modulated diffusion processes, and ascertain what properties may arise. We simulate Markov-modulated diffusion processes in MATLAB.

We also examine the probability density function for the occupation time of a Markov chain, which is the amount of time spent the Markov chain spends in a certain state. We compare this theoretical density with the empirical occupation times that we generate through Monte Carlo Markov chain simulations, and verify that the two match through a Kolmogorov-Smirnov goodness-of-fit test.

McKinlay [7] outlined a Markov-modulated model for derivatives pricing in [7], which we discuss as well as implement in MATLAB using numerical integration. We use the model to calculate the price of European options for a range of different parameters, and in doing so confirm that our results match McKinlay’s results.

2 Introduction

Diffusion processes are useful in many different applications, from physics to finance. For example, estimation of the trajectories of the Apollo 11 spacecraft can be modeled using diffusion processes. Many models in finance rely on these processes, including the Black-Scholes-Merton model, which is used to price a certain type of financial instrument known as an option. An option gives the owner the right to purchase a stock at a certain time for a certain price. In terms of underlying assets, the derivatives market is larger than the stock market itself [3]. The sheer size of this market necessitates being able to accurately price these options, and diffusion processes can be used for this purpose.

Mathematically, a diffusion process $X_t$, is a solution to a stochastic differential equation (SDE), which can be written in differential form:

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dB_t,$$

where $\{B_t\}$ is a standard Brownian motion process. The right hand side of the above SDE comprises of a deterministic term, $\mu(X_t, t)dt$, as well as a white noise term, $\sigma(X_t, t)dB_t$, where the function $\mu$ is referred to as the drift, and $\sigma$ as the diffusion.

Markov-modulated diffusions are processes where the drift and diffusions are allowed to change based on an underlying Markov chain. This flexibility can be useful for a number of different
applications. For example, in the Black-Scholes model, both $\mu$ and $\sigma$ are deterministic, but in reality the volatility of a stock may change randomly. Using a Markov-modulated diffusion process instead of an unmodulated diffusion process allows us to account for this randomness in volatility. The effect of boundary conditions on unmodulated diffusion processes has been well researched [9, 12], but there is still much to learn when it comes to those effects on Markov-modulated diffusion processes.

One reason boundary conditions are worthy of exploration is due to how often they arise in the real world. For example, a boundary condition may arise based on the physical properties of the system, such as a population going extinct at a certain level. In financial models, constraints are based on economic limitations.

In this report we use numerical methods to simulate Markov-modulated diffusion processes. We discuss boundary conditions and implement them numerically. Finally, we discuss a model from [7] and how it uses Markov-modulated diffusion processes.

3 Preliminaries

**Definition 1.** A diffusion process $\{X_t : t \geq 0\}$ is given by

$$X_t = x_0 + \int_0^t \mu(X_s, s)ds + \int_0^t \sigma(X_s, s)dB_s,$$

where $B_t$ is a standard Brownian motion process, and for some functions $\mu(X_s, s)$ and $\sigma(X_s, s)$. More commonly, they are written in differential form, which is

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dB_t, \quad t \geq 0, \quad X_0 = x_0.$$

**Definition 2.** A Markov-modulated diffusion process $\{X_t, Z_t : t \geq 0\}$ on the state space $\mathbb{R} \times S$ is given by

$$X_t = x_0 + \int_0^t \mu(X_s, Z_s, s)ds + \int_0^t \sigma(X_s, Z_s, s)dB_s, \quad Z_0 = z_0,$$

where $S = \{1, \ldots, d\}$, and $\{Z_t\}$ is a finite-state Markov chain on the state space $S$ with transition rate matrix $Q$, and $\{B_t\}$ is a standard Brownian motion process. More commonly, they are written in differential form, which is

$$dX_t = \mu(X_t, Z_t, t)dt + \sigma(X_t, Z_t, t)dB_t, \quad t \geq 0, \quad X_0 = x_0, \quad Z_0 = z_0.$$

The first integral term in **Definition 2** is Riemann, and is referred to as the drift term. The drift represents how the process will drift over time, for example, a positive drift means the process will tend to increase over time. The second integral term is a stochastic integral, since the integration is with respect to a standard Brownian motion. This term is referred to as the diffusion term, and represents how the process fluctuates over time. A large diffusion term would mean the process undergoes large random fluctuations, whereas a diffusion term of zero would mean the process is purely deterministic.

For the majority of the report we will only consider two-state Markov modulated models. Let the Markov chain $Z = \{Z_t\}$ be a two-state continuous time Markov chain, which drives both the drift and diffusion functions of the diffusion process $X_t$. Assume that the process initially begins
at $x_0 \in \mathbb{R}$. We define the state space of the Markov chain $Z$ as $S \in \{1, 2\}$. Thus, the initial state of the Markov chain is denoted by $Z_0 = z_0$, where $z_0 \in \{1, 2\}$. We let $Q$ denote transition rate matrix for $Z(\cdot)$, with

$$Q = \begin{bmatrix} -\lambda & \lambda \\ \nu & -\nu \end{bmatrix},$$

for $\lambda, \nu \geq 0$. The drift function is given by

$$\mu(X_t, Z_t, t) = \begin{cases} \mu_1(X_t) & \text{if } Z_1 = 1, \\ \mu_2(X_t) & \text{if } Z_1 = 2, \end{cases}$$

and the diffusion function by

$$\sigma(X_t, Z_t, t) = \begin{cases} \sigma_1(X_t) & \text{if } Z_1 = 1, \\ \sigma_2(X_t) & \text{if } Z_1 = 2, \end{cases}$$

where $\mu_1(X_t), \mu_2(X_t), \sigma_1(X_t), \sigma_2(X_t) \in \mathbb{R}$ and $\sigma_1(X_t), \sigma_2(X_t) \in [0, \infty)$ for all $t \geq 0$. Then, the diffusion process $X_t$ is a solution to the stochastic differential equation

$$dX_t = \mu(X_t, Z_t, t)dt + \sigma(X_t, Z_t, t)dB_t, \quad t \geq 0, \quad X_0 = x_0, \quad Z_0 = z_0.$$ 

**Definition 3.** A geometric Brownian motion process $\{X_t : t \geq 0\}$ is a process that satisfies an SDE of the form

$$dX_t = \mu X_t dt + \sigma X_t dB_t,$$

where $\mu$ and $\sigma$ are constants and $\{B_t\}$ is a standard Brownian motion.

Combining Definitions 2 and 3, we can define a Markov-modulated geometric Brownian motion, which is a geometric Brownian motion in which the constants $\mu$ and $\sigma$ are dependent on an underlying Markov chain.

**Definition 4.** A Markov-modulated geometric Brownian motion process $\{X_t, Z_t : t \geq 0\}$ on the state space $\mathbb{R} \times S$ is a process that satisfies an SDE of the form

$$dX_t = \mu(Z_t, t)X_t dt + \sigma(Z_t, t)X_t dB_t, \quad Z_0 = z_0,$$

where $S = \{1, \ldots, d\}$ and $\{Z_t\}$ is a continuous time Markov chain with transition rate matrix $Q$ and $B_t$ is a standard Brownian motion.

### 4 Simulations

#### 4.1 Unmodulated diffusion processes

Diffusion processes, even when not Markov-modulated, are generally intractable, and thus are usually solved numerically. There are different numerical methods for solving these diffusion processes, such as the Euler-Maruyama and Milstein methods, which involve approximating the process at a set of discrete time intervals.
Definition 5. The Euler-Maruyama method for solving an SDE

\[ dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dB_t, \quad X_0 = x_0, \]

is to iteratively evaluate

\[ Y_{k+1} = Y_k + \mu(X_t, t)h + \sigma(X_t, t)\sqrt{h}Z_k, \]

where \( Z_1, Z_2, \ldots \sim_{\text{iid}} \mathcal{N}(0,1) \), and \( Y_0 = x_0 \), for \( k = 0, 1, 2, \ldots \), and where \( h \) is a small positive time step. Then, \( \{Y_k, k = 0, 1, 2, \ldots\} \) approximates the process \( \{X_t, t \geq 0\} \); that is \( Y_k \approx X_{hk} \), for \( k = 0, 1, 2, \ldots \).

Definition 6. The Milstein method for solving an SDE

\[ dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dB_t, \quad X_0 = x_0, \]

is to iteratively evaluate

\[ Y_{k+1} = Y_k + \mu(X_t, t)h + \sigma(X_t, t)\sqrt{h}Z_k + \sigma_x(X_t, t)\sigma(X_t, t)(Z_k^2 - 1)\frac{h}{2}, \quad (1) \]

where \( \sigma_x \) is the partial derivative of \( \sigma(X_t, t) \) with respect to \( x \), and \( Z_1, Z_2, \ldots \sim_{\text{iid}} \mathcal{N}(0,1) \), and \( Y_0 = x_0 \), for \( k = 0, 1, 2, \ldots \), and where \( h \) is some small time step. Then, \( \{Y_k, k = 0, 1, 2, \ldots\} \) approximates the process \( \{X_t, t \geq 0\} \); that is \( Y_k \approx X_{hk} \), for \( k = 0, 1, 2, \ldots \).

See Appendix (Codes 1 and 2) for implementations of the Euler-Maruyama method and the Milstein method, respectively. The Milstein method differs from the Euler-Maruyama method only through the addition of the third term on the right hand side of Equation 1.

Definition 7. [1] An approximation method is said to have strong order of convergence equal to \( \beta \) if there exists a constant \( C \) such that for all functions \( p \) in some class

\[ |E[p(X_n)] - E[p(X(\tau))]| \leq C\Delta t^\beta, \]

for any fixed \( \tau = n\Delta t \in [0, T] \) and \( \Delta t \) sufficiently small. Typically, the functions in \( p \) must satisfy smoothness and polynomial growth conditions.

Definition 8. [1] An approximation method is said to have strong order of convergence equal to \( \gamma \) if there exists a constant \( C \) such that

\[ E|X_n - X(\tau)| \leq C\Delta t^\gamma, \]

for any fixed \( \tau = n\Delta t \in [0, T] \) and \( \Delta t \) sufficiently small.

Both of these methods have weak order of convergence \( \beta = 1 \). The Euler-Maruyama method has strong order of convergence \( \gamma = 1/2 \), and the Milstein method has strong order of convergence \( \gamma = 1 \). Both approximation methods are good, although the Milstein is slightly better in terms of strong order of convergence. We will use the Milstein method rather than the Euler-Maruyama throughout this report as long as the partial derivative of \( \sigma(X_t, t) \) is known.

Due to the stochastic nature of diffusion processes, each realisation that is approximated may differ greatly from one another, especially if the diffusion term is large. However, performing many approximations and taking the average of these realisations will result in a solution which is much closer to the exact solution.
4.2 Markov Chains

These numerical methods (Euler-Maruyama and Milstein) are defined for diffusions without any regime switching. However, for a Markov-modulated model, the drift and diffusion terms are also functions of the state of the underlying Markov chain. Consequently, to use the Euler-Maruyama and Milstein methods to approximate a Markov-modulated diffusion, the state of the underlying Markov chain must be known at each discrete time step. Thus, to simulate a realisation, we first randomly generate a sample path for the underlying Markov chain. This necessitates using Monte Carlo methods to simulate a Markov chain. The algorithm (Code 3) we use to do this in MATLAB is taken from Section 5.2 of [5]. We give a more detailed explanation of how this algorithm works in this section.

There are two properties that need to be randomly generated: the time spent in each state, and the order of states that the Markov chain is in. In generating the time spent in each state, we will use a technique known as inversion sampling.

**Theorem 1.** Suppose that a random variable $X$ has a continuous distribution with cumulative distribution function $F_X$. Then the random variable $Y$ defined as

$$Y = F_X(X)$$

has a uniform distribution.

**Proof.**

$$F_Y(y) = P(Y \leq y)$$

$$= P(F_X(X) \leq y)$$

$$= P(X \leq F_X^{-1}(y))$$

$$= F_X(F_X^{-1}(y))$$

$$= y.$$ 

Thus, $F_Y$ is the cumulative distribution function of a Uniform$(0, 1)$ random variable, and therefore $Y$ has a uniform distribution on $[0, 1]$. 

For a continuous time Markov chain, the time spent in state $i$ before transitioning, which we denote as $t$, will be randomly distributed according to $t \sim \exp(-q_{ii})$. Given it is currently in state $i$, the time until the next state transition can be found according to the following method:

1. Randomly generate a number $u$ from a standard uniform distribution.
2. Compute $t$ such that $F(t) = u$, where in this case, $F(t) = 1 - e^{q_{ii}t}$.
3. Take $t$ to be the time until the next state transition.

Noting that since $F(t) \in [0, 1]$, then $1 - F(t) \in [0, 1]$ also. Thus it is equally valid to set $u = 1 - F(t)$ in the above. Finally, rearranging gives:

$$u = 1 - F(t) = e^{q_{ii}t}$$

$$\implies \log(u) = q_{ii}t$$

$$\implies t = \frac{\log(u)}{q_{ii}}.$$
Thus, after randomly selecting $u$, we find $t$ according to the above equation.

Next, given a transition occurs at time $t$, we need to determine what state the Markov chain transitions into. This is done using the associated Jump Chain of the Markov chain.

**Definition 9.** For a continuous-time Markov chain, the Jump Chain is the discrete-time Markov chain governing the movement between states, ignoring the time spent in each state. That is, conditioning on the event the Markov chain is leaving a state, the Jump Chain describes the probability of moving to any of the other states. The elements of the transition matrix of the Jump Chain are given by

$$P_{ij} = \begin{cases} \frac{q_{ij}}{\sum_{k \neq i} q_{ik}} & \text{if } i \neq j, \\ 0 & \text{otherwise.} \end{cases}$$

**Theorem 2.** For a continuous-time Markov chain with transition rate matrix $Q$, the transition matrix $P$ for the Jump Chain is given by

$$P = (-\text{diag}(Q))^{-1}Q + I,$$

where $\text{diag}(Q)$ is the diagonal matrix formed by selecting all diagonal elements of $Q$ and setting all other elements to 0.

The probability of transitioning from state $i$ to state $j$ is given by $P_{ij}$. The next state can be determined as follows:

1. Let the vector $V$ be the cumulative sum of the $i$th row of the Jump Chain matrix.
2. Randomly generate a number $u$ from a standard uniform distribution.
3. Find $j$ such that $V_{j-1} \leq u < V_j$.

**Example 1.** Consider the case where the transition matrix for the Jump Chain is given by

$$P = \begin{bmatrix} 0 & 0.4 & 0.6 \\ 0.2 & 0 & 0.8 \\ 0.5 & 0.5 & 0 \end{bmatrix}.$$  

Assume the Markov chain begins in state 2. The cumulative sum of row 2 is

$$V = [0.2 \ 0.2 \ 1].$$

Then, we randomly generated $u$ from a standard uniform distribution. As an example, if $u$ is 0.1, the minimum number greater than 0.1 is 0.2, and hence state 1 will be transitioned into. Thus, the new state is determined as follows:

- if $0 \leq u \leq 0.2$ $\implies$ transition to state 1,
- if $0.2 < u \leq 1$ $\implies$ transition to state 3,

Combining the two techniques above, the sample paths of a continuous time Markov Chain can be randomly generated. We implement this in MATLAB, which can be seen in the Appendix [Code 3].
4.3 Markov-modulated diffusion processes

Combining the Monte Carlo Markov chain generation (Section 4.2) and either the Milstein or Euler-Maruyama methods (Section 4.1) from above, we can now generate an approximate solution for Markov-modulated diffusion processes.

For each time interval between Markov chain transitions, we apply the Milstein (or Euler-Maruyama) method separately. For each interval we use a step length

\[ h = \frac{t_f - t_i}{N}, \]

where \( t_i \) and \( t_f \) are the start and end points of the time interval, and \( N \) is the number of intervals used. The advantage of simulating the process this way will be that there will never be an iteration of the Milstein method in which the state of the Markov chain changes during the interval. A potential issue of this method is that the value of \( h \) depends on the length of each time interval, and thus will change throughout the process. One way of ensuring that the size of time intervals is fairly consistent is too allow \( N \) to be different for each interval.

We have implemented this method of simulating Markov-modulated diffusion processes in MATLAB, using both the Euler-Maruyama and Milstein methods (see Code 4 and Code 5 respectively). We demonstrate this process with Example 2.

**Example 2.** Consider the two-state model as defined in Section 3. We shall simulate the process over the time interval \([0, 5]\) using \( N = 1000 \) points. Let the parameters be defined as follow: \( x_0 = 1, \lambda = 5, \nu = 1, z_0 = 1, \)

\[ \mu(X_t, Z_t, t) = \begin{cases} 5 & \text{if } Z_t = 1, \\ -3 & \text{if } Z_t = 2, \end{cases} \]

\[ \sigma(X_t, Z_t, t) = \begin{cases} 1 & \text{if } Z_t = 1, \\ 3 & \text{if } Z_t = 2. \end{cases} \]

Note, since \( \sigma_x = 0 \) in this case, the Milstein method reduces to the Euler-Maruyama method, and thus our choice of method makes no difference.

First, we simulate a sample path for the underlying Markov chain \( \{ Z_t \} \). Using the technique discussed in Section 4.2 using MATLAB, the resulting transition times and corresponding states are given in Table 1.

<table>
<thead>
<tr>
<th>Time</th>
<th>State</th>
<th>0.0210</th>
<th>0.1692</th>
<th>0.3317</th>
<th>0.8509</th>
<th>1.0120</th>
<th>1.0563</th>
<th>1.1979</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 1: Transition times and states for the underlying Markov chain in Example 2.

The first time interval is \([0, 0.0210]\). During this interval, \( \mu(X_t, Z_t, t) = 5 \) and \( \sigma(X_t, Z_t, t) = 1 \). The length \( h \) between each iteration of the Milstein method will be

\[ h = \frac{0.0210 - 0}{1000} = 2.1 \times 10^{-5}. \]

After generating \( N \) points in this manner, the final point \( Y_N \) is used as the starting point in generating the points throughout the second interval.
The second time interval is $[0.0210, 0.1692]$. During this interval, we have $\mu(X_t, Z_t, t) = -3$ and $\sigma(X_t, Z_t, t) = 3$. The length $h$ between each iteration of the Milstein method will be

$$h = \frac{0.1692 - 0.0210}{1000} = 1.482 \times 10^{-4}.$$ 

Proceeding similarly through the rest of the time intervals up until $T = 5$, we produce an approximation of the Markov-modulated diffusion process, which can be seen in Figure 1. Since the process is random, each individual realisation is different, as can be seen in Figure 2.

Figure 1: A single trajectory generated for the Markov-modulated diffusion process in Example 2, using 100 discrete time intervals per transition. The sections of different colour represent the changes in state of $\{Z_t\}$, where red corresponds to state 1 and blue corresponds to state 2.
Figure 2: 3 different trajectories generated for the Markov-modulated diffusion process in Example 2, each using 100 discrete time intervals per transition. The solid line is the average of 100 trajectories.

5 Occupation Time

Knowing the occupation time of the underlying Markov chain can be useful in determining certain properties about a Markov-modulated diffusion process. For example, the option pricing models discussed in Section 7.2 require the occupation time to calculate the value of an option.

The occupation time for a Markov chain is the expected amount of time the Markov chain spends in a certain state. A more formal definition of the occupation time is given in Definition 10.

Definition 10 (Occupation Time). We denote by \( \tau_i \) the occupation time of state \( i \) for a Markov chain \( \{Z_t\} \) during the time interval \([0, T]\):

\[
\tau_i := \int_0^T \mathbf{1}_{\{Z(t) = i\}} \, dt, \quad i \in \mathcal{S},
\]

where \( \mathbf{1}_A \) denotes the indicator function for event \( A \), and \( \mathcal{S} \) is the set of states of the Markov chain. Then, \( T = \sum_{i \in \mathcal{S}} \tau_i \).

Theorem 3. For a two-state Markov-modulated model, the probability density function of the occupation time of state \( 1 \), given the process starts in state \( 1 \) is:

\[
f_1(x, t) = e^{-\lambda x - \nu(t-x)} \left\{ \delta(t-x) + \left( \lambda \nu x / (t-x) \right)^{1/2} I_1 \left[ 2(\lambda \nu x(t-x))^{1/2} \right] + \lambda \rho \left[ 2(\lambda \nu x(t-x))^{1/2} \right] \right\},
\]
where $\delta(\cdot)$ is the Dirac delta function and

$$I_r(z) = \sum_{k=0}^{\infty} \frac{(z/2)^{2k+r}}{k!(k+r)!},$$

which is the modified Bessel function of order $r$.

Similarly, the probability density function of the occupation time of state 1, given the process starts in state 2, is:

$$f_2(x,t) = e^{-\lambda x - \nu(t-x)} \left\{ \delta(x) + \left( \frac{\nu(t-x)}{x} \right)^{1/2} I_1 \left[ 2 \left( \frac{\nu x(t-x)}{x} \right)^{1/2} \right] + \lambda I_0 \left[ 2 \left( \frac{\nu x(t-x)}{x} \right)^{1/2} \right] \right\}.$$

This density function was first derived by Pedler in [8], and more recently by Kovchegov et al. in [4] and McKinlay in [7].

We have implemented this function in MATLAB (Code 7).

**Example 3.** Consider a two-state Markov chain $Z_t$, and let

$$Q = \begin{bmatrix} -10 & 10 \\ 5 & -5 \end{bmatrix}.$$

We consider a time period $[0, 50]$. Then, the probability density function of the time spent in state 1 will be given by

$$f_1(x, 50) = e^{-10x - 10(50-x)} \left\{ \delta(50 - x) + \left( \frac{100x}{50-x} \right)^{1/2} I_1 \left[ 2 \left( \frac{100x(50-x)}{x} \right)^{1/2} \right] + 10 I_0 \left[ 2 \left( \frac{100x(50-x)}{x} \right)^{1/2} \right] \right\}.$$

We plot this density in blue in Figure 3 as a function of $x$. The density of the occupation time in state 2 is then given by $f_1(50 - x, 50)$, since there are only two possible states for this Markov chain. We plot this density in red in Figure 3.

We can verify the formulae in Theorem 3 via simulations. In particular, we can generate sample paths for the underlying Markov chain numerically (Code 3), and calculate the time spent in each state directly (Code 8). Repeating this with a larger numbers of sample paths increases the accuracy of the empirical distribution of occupation times, as the observed occupation times should converge to the distribution stated above. This appears to be true based on Figure 4 and we test it formally using the Kolmogorov-Smirnov goodness-of-fit test.
Figure 3: The theoretical distribution of the occupation time for state 1 (blue) and state 2 (red), for a two-state Markov chain with transition matrix given in Example 3, given the process started in state 1.

Figure 4: The theoretical distribution of the occupation time for state 1 (solid line) for Example 3 and the empirical occupation times (histogram), in a two-state Markov chain with transition matrix as given in Example 3, obtained from 10000 simulations.
The Kolmogorov-Smirnov test is used to assess whether or not some set of data points were sampled from a given continuous distribution [11].

**Definition 11.** Let $x_1, \ldots, x_N$ be an iid sample from some continuous distribution. The Kolmogorov-Smirnov statistic is

$$D_N = \sup_{x \in \mathbb{R}} |\hat{F}_N(x) - F(x)|,$$

where $\hat{F}_N(x)$ is the empirical cdf and $F(x)$ is the true cdf. The Kolmogorov-Smirnov statistic is the maximum distance between the empirical and true cdf.

The Kolmogorov-Smirnov test is a hypothesis test, where in this case our hypothesis is

- $H_0$: the empirical occupation time is from the theoretical occupation time distribution (Theorem 3)
- $H_a$: the empirical occupation time is not from this distribution.

In a KS test we first find $D_N$, then scale it to obtain

$$K_N = \sqrt{N} D_N.$$

Then, the p-value is the probability of observing a value of $K_N$ as extreme or more extreme than the $K_N$ that was observed.

We perform a KS test on the empirical occupation time, using Code 9. The occupation time is tested for the continuous-time Markov chain defined in Example 3 that is, with transition rate matrix

$$Q = \begin{bmatrix} -10 & 10 \\ 5 & -5 \end{bmatrix}.$$ 

We assume the Markov chain starts in state 1, and generated $N = 1000$ realisations over the time interval $[0, 50]$. The resulting value for the KS statistic is $K_N = 0.0261$, and thus the p-value is 0.4938. Therefore, there is insufficient evidence to reject the null hypothesis at the 0.05 significance level, and we must accept that the empirical occupation times could have been sampled from the theoretical distribution of occupation times. This is only true for this specific Markov chain in this example, although it is reasonable to assume that it is true in general. The theoretical distributions have been proven true [8,4,7], thus this result is a verification that the method used to empirically generate out occupation times is accurate. Knowing this, we can now use our method of empirically generating the occupation time in models with greater than 2 states. This is useful since as of now there is no known theoretical distribution for the occupation time for Markov chains with more than 2 states.
6 Boundary Conditions

Diffusion processes can take any value on the real line. In real world applications, this is not necessarily desirable. For example, if the price of something was modelled by a diffusion process, it would not make sense for the price to be negative. Boundary conditions are used to restrict diffusion processes to certain values by making the process undergo some kind of change upon hitting a boundary. Depending on the type of boundary condition, the process may undergo different effects upon reaching the boundary.

We will consider absorbing, instantaneous jump, and reflecting boundaries.

6.1 Absorbing Boundaries

An absorbing boundary is one in which the process will be “absorbed” by the boundary upon contact. That is, the process stays at the boundary forever once it reaches it.

In reality, a diffusion process with an absorbing boundary can occur in population dynamics. When a population reaches 0 it goes extinct. Once extinct, there is no chance of the population increasing, and thus it will remain at 0. Another example is the value of some kind of financial instrument. Upon reaching $0, it is valueless, and will never increase in value from there. Both of these examples have an absorbing boundary at 0. There are other real world examples of absorbing boundaries, and they are not necessarily at 0.

Definition 12. Consider a Markov-modulated diffusion process \( \{X_t\} \). We can define a Markov-modulated diffusion process \( \{Y_t\} \) with an absorbing boundary at \( L \) by

\[
dY_t = \begin{cases} 
0 & \text{if } X_t = L \\
\frac{dX_t}{dt} & \text{if } X_t \neq L
\end{cases}
\]

We implement a simulation algorithm to generate a diffusion process with an absorbing boundary given any diffusion process as input [Code 10]. For all time steps, beginning at \( t = 0 \), the value of \( X_t \) is tested to see whether \( X_t = L \) at any time during that step. If it is, then the process at all remaining values of \( t \) is set to be \( L \).

Example 4. Consider a 2-state Markov-modulated diffusion model. We let the transition rate matrix be

\[
Q = \begin{bmatrix} -3 & 3 \\ 1 & -1 \end{bmatrix}
\]

and define the drift and diffusion functions as

\[
\mu(X_t, Z_t, t) = \mu(Z_t),
\]

\[
\sigma(X_t, Z_t, t) = \sigma(Z_t),
\]

where \( \mu = (5, -3) \), \( \sigma = (1, 10) \) are the vectors representing the value of the drift and diffusion at each of the two states.

We introduce an absorbing boundary at \( L = -20 \). As can be seen in Figure 3, any individual realisation that reaches the boundary is absorbed and stays at \( Y_t = -20 \) permanently. However, the mean of \( Y_t \) across all 1000 realisations is not necessarily absorbed. The mean of \( Y_t \) still looks fairly similar to \( X_t \), though it flattens out somewhat around the boundary. Figure 6 shows a single
realisation for both $X_t$ and $Y_t$, clearly showing how the boundary condition affects the process. This specific example is implemented in Code 13.

We now introduce a second boundary, at $U = 20$. As can be seen in Figure 7, some realisations get absorbed at $L$ and some get absorbed at $U$. The mean of the resulting process $\{Y_t\}$ now remains fairly flat along 0. Figure 8 shows a single realisation of both $X_t$ and $Y_t$, which seem to be unaffected by the introduction of the upper boundary.

Figure 5: The diffusion process $X_t$ (top two figures) from Example 4 compared with the absorbing process $Y_t$ (bottom two figures), which has an absorbing boundary at $L = -20$. 
Figure 6: A single realisation of $X_t$ (blue) and $Y_t$ (red), from Example 4.

Figure 7: The diffusion process $X_t$ (top two figures) from Example 4 compared with the absorbing process $Y_t$ (bottom two figures), which has two absorbing boundaries at $L = -20$ and $U = 20$. 
Figure 8: A single realisation of $X_t$ (blue) and $Y_t$ (red), from Example 4.
6.2 Instantaneous Jump Boundaries

An instantaneous jump boundary is one in which the process jumps to some other value instantaneously upon hitting the boundary. Such a boundary could also occur in population dynamics, for example. Upon a population reaching a certain level, it may be policy for a certain group to either reintroduce more of the population into the wild from captivity (jumping to a higher level) or to cull some of the population (jumping to a lower level). This boundary condition could also appear in finance, where, for example, upon a company’s money supply dropping to a certain amount, they take out a loan. There are many other real world examples for instantaneous jump boundaries.

**Definition 13.** Consider a Markov-modulated diffusion process \( \{X_t\} \). We define \( \tau_i \) as the \( i \)th time the diffusion process hits the boundary \( L \). Then we can define

\[
Y_t = \begin{cases} 
X_t + i(M - L) & \text{if } \tau_i \leq t \leq \tau_{i+1}, \\
X_t & \text{if } t \leq \tau_i.
\end{cases}
\]

and we say \( \{Y_t\} \) is Markov-modulated diffusion process with an instantaneous boundary at \( L \) which jumps to \( M \).

We have used MATLAB and **Definition 13** to generate a diffusion process with an instantaneous jump boundary given any diffusion process as input (**Code 11**).

**Example 5.** Beginning with the same model for \( X_t \) as in **Example 4**, we introduce an instantaneous jump boundary at \( L = -20 \), which causes the process to jump to \( M = -10 \).

As can be seen in **Figure 9**, the resulting process \( Y_t \) has a mean not too dissimilar to the absorbing boundary condition (**Figure 5** and **Figure 7**). However, looking at the plot of individual realisations, it can be seen that unlike the absorbing process, this process still fluctuates around even though it stays close to the boundary. **Figure 10** shows a plot of a single realisation of both \( X_t \) and \( Y_t \), clearly showing the large jumps that \( Y_t \) undergoes. We implement this specific example in **Code 13**.
Figure 9: The diffusion process $X_t$ (top two figures) from Example 5, compared with the instantaneous jump process $Y_t$ (bottom two figures), which has an instantaneous jump boundary at $L = -20$ which jumps to $M = -10$.

Figure 10: A single realisation of both $X_t$ (blue) and $Y_t$ (red), from Example 5.
6.3 Reflected Boundaries

A reflected boundary is a boundary which appears to cause the diffusion process to “reflect” upon hitting it. One real world application of reflected Brownian motion is in computing the bankrupt time distribution for insurance companies ([2]). The surplus level of the company is modelled by a diffusion process, and whenever the surplus reaches a certain barrier the company pays dividends to its shareholders, thus lowering and effectively “reflecting” the surplus.

**Definition 14.** Consider a Markov-modulated diffusion process \( \{X_t\} \). We can define a process \( \{Y_t\} \) by

\[
Y_t = X_t + \max\left\{ 0, -\inf_{s \leq t} X_s \right\},
\]

we say \( \{Y_t\} \) is a Markov-modulated diffusion process reflected at 0. Alternatively, we can define

\[
Y_t = X_t + \max\left\{ 0, -\inf_{s \leq t} (X_s - L) \right\},
\]

and we say \( \{Y_t\} \) is a Markov-modulated diffusion process reflected at \( L \).

**Example 6.** Beginning with the same model for \( X_t \) as in [Example 4] we introduce a reflecting boundary at \( L = -20 \).

As can be seen in Figure 11, the mean of the resulting process \( Y_t \) looks similar to the other boundary types, where the downward trend of the original process is flattening out as it approaches the boundary. The plot of individual realisations looks similar to the instantaneous jump case (Figure 9), although the fluctuations appear to be getting smaller as time goes on. Figure 12 shows a single realisation of both \( X_t \) and \( Y_t \) for a more detailed comparison.
Figure 11: The diffusion process $X_t$ (top two figures) from Example 6 compared with the reflected process $Y_t$ (bottom two figures), which has a reflecting boundary at $L = -20$.

Figure 12: A single realisation of both $X_t$ (blue) and $Y_t$ (red), from Example 6.
7 Applications to Finance

7.1 Black-Scholes Model

A European call option is a contract which gives the owner the right to buy a stock at time $T$ (known as the expiry time) for the price $K$ (known as the strike price). If the price of the stock at time $T$, denoted $S(T)$ is greater than $K$, then the owner of the option can buy the stock at the cheaper price $K$ and immediately sell at price $S(T)$, making a profit of $S(T) - K$. On the other hand, if the stock price is less than the strike price at expiry, the owner would only make a loss by exercising the contract, thus the option would be worthless in this scenario. Thus we can define the value of a European option at expiry (see Definition 15).

The Black-Scholes model is a mathematical model of a financial market containing derivatives. Using the Black-Scholes model, we can derive the price of a European call option at any time before expiry (see Definition 16).

**Definition 15.** A *European call option* with maturity $T$ and strike price $K$ has the value at expiry, $C_T$ given by

$$C_T = (S(T) - K)^+,$$

where $(\cdot)^+ := \max(0, \cdot)$.

The Black-Scholes model makes a few assumptions about the nature of assets and of the market:

- The rate of return on a riskless asset, referred to as the risk-free interest rate, is constant.
- There is no arbitrage opportunity. That is, there is no way to make a riskless profit.
- The underlying stock is assumed not to pay a dividend. (Although trivial extensions can be added to the model to allow for a dividend.)
- It is possible to borrow and lend any amount of cash at the riskless rate. It is also possible to buy and sell (including short selling) any amount of the stock. This includes fractional amounts of cash and stock.
- Transactions do not incur any fees.
- The instantaneous log return of the stock price is a geometric Brownian motion (see Definition 4) with constant drift and volatility. That is

$$dS_t = \mu S_t dt + \sigma S_t dB_t,$$

where $S_t$ is the price of the stock at time $t$.

**Definition 16.** The *Black-Scholes price* of a European call option is given by

$$C^*(S, K, r, T, \sigma) = S \Phi(d_1) - Ke^{-rT} \Phi(d_1 - \sigma \sqrt{T}),$$

where $\Phi$ is the standard normal cumulative distribution function and

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)T}{\sigma \sqrt{T}},$$

$S$ is the current price of the stock price, $K$ is the strike price of the option (assumed to be positive), $r$ is the risk-free interest rate, $T$ is the time until expiry, and $\sigma$ is the volatility of the stock price.
Although useful, the Black-Scholes model has its flaws. The prices found by the Black-Scholes formula have been shown empirically to not match the real prices of the options \[10\]. This may be attributed to the fact that the assumptions of constant volatility (or even that the volatility is not random) is false.

7.2 Markov-modulated Model (McKinlay)

McKinlay \[7\] constructed a new model based on the Definition 16, although the author removed the assumption that the underlying stock price followed a geometric Brownian motion. Instead, this new model assumes the underlying stock followed a Markov-modulated geometric Brownian motion, allowing for the existence of different levels of volatility in the stock at different times.

**Theorem 4** (Price of a European option). For a two-state Markov-modulated diffusion model, the price of a European option is given by

\[
C(t) = \int_0^{T-t} C^* \left[ S(t), K, T-t, \sqrt{\frac{v_j(t)(s)}{T-t}} \right] f_{Z(t)}(s) ds,
\]

where \( C^*(\cdot) \) is the Black-Scholes formula for call options (Definition 16), \( f_j(s) \) is the conditional density of the occupation time of state \( j \) by \( Z(\cdot) \) during the time interval \([t,T]\), given \( Z(t) = j \) (see Theorem 3), and, for \( 0 \leq s \leq T-t \),

\[
v_j(s) = \begin{cases} 
\sigma_1^2 s + \sigma_2^2 (T-t-s) & \text{if } j = 1, \\
\sigma_2^2 s + \sigma_1^2 (T-t-s) & \text{if } j = 2.
\end{cases}
\]

In deriving the pricing formula in Theorem 4 McKinlay used the assumption that the market is arbitrage-free to derive an Equivalent Martingale Measure for the price of the stock price over time. The proof of this result in Theorem 5 is omitted (see \[7\] for proof).

**Theorem 5.** Let \( S_t \) be the price of a risky asset. If we assume \( S_t \) evolves according to the SDE

\[
dS_t = \mu(t) S_t dt + \sigma(t) S_t d\tilde{B}_t, \quad S_0 \geq 0, \quad t \in [0,T],
\]

then there exists an Equivalent Martingale Measure \( \mathbb{Q} \) such that

\[
dS_t = r S_t dt + \sigma(t) S_t d\tilde{B}_t,
\]

where \( \{\tilde{B}\} \) is a \( \mathbb{Q} \)-Brownian motion.

The result in Theorem 5 is useful in that we can use our methods of generating Markov-modulated diffusion processes from Section 4 to model the stock price over time. We represent the risk free interest rate, \( r \), with the function \( \mu(Z_t, t) \), but this time the state of the Markov chain does not affect the value. That is

\[
\mu(Z_t, t) = \begin{cases} 
r & \text{if } Z_t = 1, \\
r & \text{if } Z_t = 2.
\end{cases}
\]
Table 2: The parameters for the three models outlined in Example 7.

<table>
<thead>
<tr>
<th></th>
<th>Model 1</th>
<th>Model 2</th>
<th>Model 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>1</td>
<td>$\lambda$</td>
<td>3</td>
</tr>
<tr>
<td>$\nu$</td>
<td>1</td>
<td>$\nu$</td>
<td>1</td>
</tr>
<tr>
<td>$\sigma_1$</td>
<td>0.25</td>
<td>$\sigma_1$</td>
<td>0.25</td>
</tr>
<tr>
<td>$\sigma_2$</td>
<td>0.55</td>
<td>$\sigma_2$</td>
<td>0.55</td>
</tr>
</tbody>
</table>

The diffusion term is used to represent the volatility of the stock price. Unlike the interest rate, this term is Markov-modulated. We can define two different levels of volatility, $\sigma_1$ and $\sigma_2$ (for example, low and high volatility), and the diffusion term will be given by

$$\sigma(Z_t, t) = \begin{cases} \sigma_1, & \text{if } Z_t = 1, \\ \sigma_2, & \text{if } Z_t = 2. \end{cases}$$

McKinlay outlined three examples in which he tested his two-state Markov chain model. He implemented a method in which the price of a European option is evaluated numerically from the equation in Theorem 4. He also devised a second method involving a Fast Fourier Transform, which was designed to allow for efficient computations even when using models with a large number of states. McKinlay showed that the two methods produced the same results for all of the models tested.

We implement McKinlay’s model (Code 14) using numerical integration in a similar way to McKinlay. We test it using each of his three examples and solve for the price, comparing our results to his (Code 15).

**Example 7.** For all three of the example models we let the initial price of the stock $S_0 = 50$, the risk-free interest rate $r = 0.1$, and the starting state of the underlying Markov chain $Z(\cdot) = 1$. We price an option at time $t = 0$ for an expiry at time $T = 1$.

The parameters of the three models are given in Table 2 noting that the transition matrix for $Z(\cdot)$ is

$$Q = \begin{bmatrix} -\lambda & \lambda \\ \nu & -\nu \end{bmatrix}.$$ 

Each of the models is tested with seven different values for the strike price, $K$. Table 7 shows each strike price with the corresponding option price found by McKinlay.

The output of our code (Code 15) used in testing our implementation is given below.
Table 3: The strike prices tested by McKinlay [7], along with the corresponding option prices that were calculated.

<table>
<thead>
<tr>
<th>Strike Price</th>
<th>Model 1</th>
<th>Model 2</th>
<th>Model 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>29.943175</td>
<td>23.229596</td>
<td>23.568903</td>
<td>23.104774</td>
</tr>
<tr>
<td>39.952443</td>
<td>15.354187</td>
<td>16.332901</td>
<td>15.020033</td>
</tr>
<tr>
<td>49.802698</td>
<td>9.315028</td>
<td>10.919362</td>
<td>8.8040807</td>
</tr>
<tr>
<td>59.890066</td>
<td>5.284886</td>
<td>7.0734063</td>
<td>4.7166456</td>
</tr>
<tr>
<td>69.829658</td>
<td>2.9956081</td>
<td>4.5946202</td>
<td>2.4739770</td>
</tr>
<tr>
<td>79.921838</td>
<td>1.7274812</td>
<td>2.9876057</td>
<td>1.3000675</td>
</tr>
</tbody>
</table>

**MODEL 1:**

McKinlay’s price: 23.229596, Our Price: 23.229596, Difference: 0.000000

McKinlay’s price: 15.354187, Our Price: 15.354187, Difference: 0.000000

McKinlay’s price: 9.315028, Our Price: 9.315028, Difference: 0.000000

McKinlay’s price: 5.284886, Our Price: 5.284886, Difference: 0.000000

McKinlay’s price: 2.9956081, Our Price: 2.9956081, Difference: 0.000000

**MODEL 2:**

McKinlay’s price: 23.568903, Our Price: 23.568903, Difference: 0.000000

McKinlay’s price: 16.353291, Our Price: 16.353291, Difference: 0.000000

McKinlay’s price: 10.919362, Our Price: 10.919362, Difference: 0.000000

McKinlay’s price: 7.073406, Our Price: 7.073406, Difference: 0.000000

McKinlay’s price: 4.594620, Our Price: 4.594620, Difference: 0.000000

McKinlay’s price: 2.987606, Our Price: 2.987606, Difference: 0.000000

**MODEL 3:**

McKinlay’s price: 23.104774, Our Price: 23.104774, Difference: 0.000000

McKinlay’s price: 15.020033, Our Price: 15.020033, Difference: 0.000000

McKinlay’s price: 8.8040807, Our Price: 8.8040807, Difference: 0.000000

McKinlay’s price: 4.7166456, Our Price: 4.7166456, Difference: 0.000000

McKinlay’s price: 2.473977, Our Price: 2.473977, Difference: 0.000000

McKinlay’s price: 1.3000675, Our Price: 1.3000675, Difference: 0.000000

As can be seen by the output, our option prices match McKinlay’s prices for at least 6 decimal places.

**8 Conclusion**

In this report we have discussed and implemented a method of simulating Markov-modulated diffusion processes, using a combination of the Milstein method and a random Markov Chain generation.
We investigated a theorem for the occupation time of a Markov chain, and verified that it is consistent with our numerical simulations using a Kolmogorov-Smirnov goodness-of-fit test.

Some real world applications of boundary conditions for Markov-modulated diffusion processes were discussed, and three types of boundary conditions investigated: reflected, absorbing, and instantaneous jump. For each of these boundaries we simulated some processes with and without these boundaries applied, and compared the processes against each other.

Finally, we looked at how Markov-modulated diffusion processes can be used in finance. Particularly, we look at how the Black-Scholes model can be improved by instead using Markov-modulated diffusion processes, as is done by McKinlay in [7]. We implemented the model from [7] in MATLAB and verified that we produced the same results as [7].

Moving forward, the next step would be to implement a numerical model for modelling European options which uses Markov-modulated diffusion process with more than two states. This can be done in combination with the results of McKinlay’s thesis ([7]) and the method we used in this report of finding an empirical occupation time. Eventually this could be scaled up to using $n \in \mathbb{N}$ state Markov chains, which would allow a volatility to be chosen which is effectively continuous. The other area which is yet to be investigated is the effect that these boundary conditions can have on the pricing model and hence the price of an option.
References


Appendix - MATLAB Code

MATLAB Code 1: EulerMaruyama.m

function [t, y] = EulerMaruyama(reps,a,b,x0,T,N,seed)
    %EulerMaruyama uses the Euler Maruyama method to solve an SDE
    %
    % reps is number of simulations to repeat
    % a is deterministic term - function of t and y
    % b is random term - function of t and y
    % x0 is starting pos
    % T is final time
    % N is number of discrete time intervals to use
    % seed is rng seed
    %
    % t is the 1xN vector of time steps that are used
    % y is the (reps)xN vector of solutions at time steps
    %
    % Lachlan Bridges
    % 06/01/17

    randn('seed',seed)

    y=zeros(reps,N+1);
    dt=T/N;
    t=0:dt:T;
    y(:,1)=repmat(x0,reps,1);
    Z=randn(reps,N);
    for i=2:N+1
        y(:,i)=y(:,i-1)+a(y(:,i-1),i*dt)*dt + b(y(:,i-1),i*dt)*sqrt(dt).*Z(:,i-1);
    end

MATLAB Code 2: Milstein.m

function [t, y] = Milstein(reps,a,b,dbdy,x0,T,N,seed)
    %Milstein uses the Milstein method to solve an SDE
    %
    % reps is number of simulations to repeat
    % a is deterministic term - function of t and y
    % b is random term - function of t and y
    % dbdy is the derivative of b w.r.t y - function of t and y
    % x0 is starting pos
    % T is final time
    % N is number of discrete time intervals to use
    % seed is rng seed
    %
    % t is the 1xN vector of time steps that are used
    % y is the (reps)xN vector of solutions at time steps
    %

    y(:,1)=repmat(x0,reps,1);
    dt=T/N;
    t=0:dt:T;
    y(:,1)=y(:,1)+a(y(:,1),0)*dt + b(y(:,1),0)*sqrt(dt).*Z(:,1);
    Z=randn(reps,N);
    for i=2:N
        y(:,i)=y(:,i-1)+a(y(:,i-1),i*dt)*dt + b(y(:,i-1),i*dt)*sqrt(dt).*Z(:,i-1)+
    end

27
function [tt, yy] = CTMC(Q,y0,T,reps, seed)
%CTMC generates sample paths for a Continuous Time Markov Chain
% tt is the vector of transition times
% yy is the vector of states
% Q is the transition rate matrix of the Markov Chain
% y0 is the initial state of the Markov Chain
% T is the length of time to generate
% reps is the number of sample paths to generate
% seed is the rng seed
% Credit to: Handbook of Monte Carlo Methods (Kroese, Taimre, Botev)
% 06/01/17

rng(seed)
S=size(Q,2);
q=-diag(Q);
K=diag(1./q)*Q + eye(S);
for j=1:reps
n=1;
t=0; y=y0;
yy(j,1)=y;
 tt(j,1)=t;
while t < T
    A = -log(rand)/q(y);
    y = find(cumsum(K(y,:))>rand, 1 );
    t = t + A;
end

yy(j,:)=y;
tt(j,:)=t;
end
yy=[yy; tt];

MATLAB Code 4: MMEulerMaruyama.m

function [t, y] = MMEulerMaruyama(Q,reps,a,b,x0,T,N,seed)
% MMEulerMaruyama uses Euler Maruyama method to solve a Markov Modulated SDE
% Q is the transition rate matrix of the underlying MC
% reps is number of simulations to repeat
% a is deterministic term - function of t, y and Z
% b is random term - function of t, y and Z
% x0 is starting pos
% T is final time
% N is number of discrete time intervals to use
% seed is rng seed
% t is the 1xN vector of time steps that are used
% y is the (reps)xN vector of solutions at time steps
% Lachlan Bridges
% 06/01/17

randn('seed',seed)

dt=T/N;
t=0:dt:T;

[tt, yy]=CTMC(Q,1,T,reps,seed);
for j=1:reps
    for i=1:length(t)
        J(j,i)=yy(j,find(tt(j,:)>t(i),1)-1);
    end
end

y=zeros(reps,N+1);
y(:,1)=repmat(x0,reps,1);
Z=randn(reps,N);
for i=2:N+1
    y(:,i)=y(:,i-1)+a(y(:,i-1),i*dt,J(i))*dt + b(y(:,i-1),i*dt,J(i))*sqrt(dt).*Z(:,i-1);
end
function [t, y, J, tt, yy] = MMMilstein(Q, reps, a, b, dbdy, x0, T, N, seed)

% MMMilstein uses the Milstein method to solve a Markov Modulated SDE
% % Q is the transition rate matrix of the underlying MC
% % reps is number of simulations to repeat
% % a is deterministic term - function of t, y and Z
% % b is random term - function of t, y and Z
% % dbdy is the derivative of b w.r.t y - function of t, y, Z
% % x0 is starting pos
% % T is a 2 element vector, specifying the start and end time
% % N is number of discrete time intervals to use between each transition
% % seed is rng seed
% %
% % t is the (reps)xN vector of time steps that are used
% % y is the (reps)xN vector of solutions at time steps
% %
% % Lachlan Bridges
% % 06/01/17

rng(seed)

[tt, yy] = CTMC(Q, 1, T(2), reps, seed);
maxjumps = (size(yy, 2));
intervals = (maxjumps-1)*(N);
t = zeros(reps, intervals);
y = t;
J = y;
for i = 1:reps
    Jrep = [];
    trep = [];
yrep = [];
    ttt = tt(i, :);
    yyy = yy(i, :);
    jumps = find(isnan(yyy), 1);
    if isempty(jumps);
        jumps = maxjumps;
    end
    for j = 1:jumps-1
        newT = [ttt(j), ttt(j+1)];
        state = yyy(j);
        if j == 1
            y0 = x0;
        else
            y0 = yrep(end);
        end
        [tempt, tempy] = Milstein(1, @(y, t) a(y, t, state), @(y, t) b(y, t, state), ... 
            @(y, t) dbdy(y, t, state), y0, newT(N, seed);
        tempt(1) = [];
        tempy(1) = [];
    end
end
`tempJ = repmat(state,size(tempt));
trep = [trep tempt];
yrep = [yrep tempy];
Jrep = [Jrep tempJ];
end

trep = [trep zeros(1,intervals-length(trep))];
yrep = [yrep zeros(1,intervals-length(yrep))];
Jrep = [Jrep zeros(1,intervals-length(Jrep))];
t(i,:) = trep;
y(i,:) = yrep;
J(i,:) = Jrep;
end

y(t==0 | t>T(2)) = nan;
J(t==0 | t>T(2)) = nan;
t(t==0 | t>T(2)) = nan;

---

MATLAB Code 6: MMMilsteinExample.m

```matlab
%MMMilsteinExample tests the MMMilstein function with an example Markov-modulated
% diffusion process with 100 steps per interval
%
% % Lachlan Bridges
% % 06/01/17

Q = [-5,5;1,-1];
mu = [5, -3];
sigma = [1, 3];

a = @(y,t,J) mu(J);
b = @(y,t,J) sigma(J);
dbdy = @(y,t,J) 0;

x0 = 1;
T = [0,5];
seed = 31;
r = 3; % number of realisations to plot
N=100;
reps=100;

[t,y,J,tt,yy]=MMMilstein(Q,reps,a,b,dbdy,x0,T,N,seed);
plotmm(t(1,:),y(1,:),J(1,:),[1,0,0;0,0,1]), hold on
title(sprintf('%Single realisation of a Markov-modulated diffusion\nprocess, using N=%i
time steps per interval',r,reps,N));
xlabel('t','FontSize',14)
ylabel('X_t','FontSize',14)
saveas(gcf,sprintf('../figures/examplediff.png'))
close all

plotmm(t(1:r,:),y(1:r,:)), hold on
```

```matlab
[tm, ym] = meanmm(t, y, 10000);
plot(tm, ym)

title(sprintf('%i realisations of an MMDP and the mean across %i realisations, using N=%i time steps per interval', r, reps, N));
xlabel('t', 'FontSize', 14)
ylabel('X_t', 'FontSize', 14)
saveas(gcf, sprintf('..\figures\examplediffe.png'));
close all

MATLAB Code 7: OccTime.m

function [f1, f2] = OccTime(lambda, mu)

%OccTime calculates the conditional (generalised) densities of the times
% spent in the first state of a two state Markov chain, and returns them
% as functions of x (time spent in state 1) and t (length of the time
% interval)
%
% f1 is the time spent in state 1, given the process in state 1 at time t
% f2 is the time spent in state 1, given the process in state 2 at time t
%
% lambda and mu are the values of the Q transition matrix
%
% Lachlan Bridges
% 09/01/17

f1 = @(x,t) exp(-lambda*x-mu.*(t-x)) .* ( dirac(t-x)+sqrt(lambda*mu.*x./(t-x)).*besseli(1,2*sqrt(lambda*mu.*x.*(t-x))) +
lambda*besseli(0,2*sqrt(lambda*mu.*x.*(t-x))) );

f2 = @(x,t) exp(-lambda*x-mu.*(t-x)) .* ( dirac(x)+sqrt(lambda*mu.*(t-x)./x).*besseli(1,2*sqrt(lambda*mu.*x.*(t-x))) +
lambda*besseli(0,2*sqrt(lambda*mu.*x.*(t-x))) );
end

MATLAB Code 8: EmpOccTime.m

function f = EmpOccTime(d, Q, y0, T, N, seed)

%EmpOccTime uses the CTMC function to randomly generate Markov chains with
% which it calculates the probability density function of the empirical
% occupation time
%
% f1 is the probability density function, given the process started in
% state y0, function of time
%
% d is the state for which the occupation time shall be found
% Q is the transition rate matrix for the Markov chain
% y0 is the state that the Markov chain started in
% T is the time interval for which the occupation time will be found
```
% N is the number of Markov chains to generate. larger n -> more accurate
% seed is the seed to be used with the random number generator
%
% Lachlan Bridges
% 16/01/17

[tt,yy] = CTMC(Q,y0,T,N,seed);
tt(tt>T) = T;
tt(tt==0) = T;
tt(isnan(tt)) = T;
tt(:,1) = 0;
tdifs = tt(:,2:end)-tt(:,1:end-1);

idt = (yy==1);
x = sum(tdifs.*idt(:,1:(end-1)),2);
f = sort(x);
end

MATLAB Code 9: KSTestOccTime.m

% KSTestOccTime uses the Kolmogorov-Smirnov goodness-of-fit test to test
% whether or not the empirical CDF of the occupation time, found from
% the Monte Carlo generated Markov chains, match the theoretical
% distribution of the occupation time
%
% Lachlan Bridges
% 02/01/2017

lambda = 10;
nu = 5;
Q=[-lambda, lambda; nu, -nu]; % transition rate matrix

y0 = 1; % initial state
T = 50; % length of time interval
N = 1000; % number of distinct sample paths to generate

x1=EmpOccTime(1,Q,y0,T,N,100);

% function for theoretical pdf of occupation time in state 1
[f1, f2] = OccTime(lambda, nu);
func = @(x) f1(x,T);

x = linspace(0,T,100000); % data points to generate cdf at
x = x(1:end-1); % remove last point since NaN
f = func(x); % theoretical pdf
F = cumsum(f);
F = F./F(end); % theoretical cdf

cdf = [x' F']; % <- THEORETICAL OCC TIME
\[ [H, P, KSTAT] = \text{kstest}(x1, 'CDF', \text{cdf}) \]

\% PLOTTING THEORETICAL

\s = \text{linspace}(0, T, 1000);
\text{f1plot} = \text{f1}(s, T);
\text{f2plot} = \text{f1}(T-s, T);

\text{plot}(s, \text{f1plot}, '-b') \% state 1 occtime
\text{hold on}
\text{plot}(s, \text{f2plot}, '-r')
\text{title('Probability distribution of occupation times for states 1 and 2')}
\text{xlabel('Occupation time')}
\text{ylabel('Probability')}
\text{fname} = \text{sprintf}('../\text{figures/tocctime1_lam%i_nu%i.png}', \lambda, \nu);
\text{saveas(gcf, fname)}
\text{hold off}

\% PLOTTING THEORETICAL VS EMPIRICAL
\text{plot}(s, \text{f1plot}), \text{hold on}
\text{histogram(x1, 'Normalization', 'probability', 'BinWidth', 1)}
\text{title(sprintf('Probability distribution of occupation times for state 1
vs. empirical
occupation times'))}
\text{xlabel('Occupation time')}
\text{ylabel('Probability')}
\text{fname} = \text{sprintf('..//\text{figures/occtime_lam%i_nu%i.png}', \lambda, \nu)};
\text{saveas(gcf, fname)}
\text{hold off}
\text{close all}

MATLAB Code 10: BMAbsorbing.m

\textbf{function} \text{X} = \text{BMAbsorbing}(B, lb, ub)
\% BMAbsorbing generates a diffusion process with absorbing boundary
\% conditions, given a normal diffusion process as an input
\% \% X is the (reps)xN vector of solutions to the diffusion process with jump
\% \% B is the original diffusion process
\% \% lb is the value of the lower boundary at which the process will be
\% absorbed (put [] if there is no lower bound)
\% \% ub is the value of the upper boundary at which the process will be
\% absorbed (put [] if there is no lower bound)
% Lachlan Bridges
% 06/01/17

% case when no upper/lower boundary
if isempty(lb)
    lb = -inf;
end
if isempty(ub)
    ub = inf;
end

N = size(B,2);
reps = size(B,1);

X = B;

for r=1:reps
    for k=2:N
        if X(r,k) >= ub
            X(r,k:end) = ub;
        end
        if X(r,k) <= lb
            X(r,k:end) = lb;
        end
    end
end

function X = BMInstantJump(B,lb,ljump,ub,ujump)
%BMInstantJump generates a diffusion process with instantaneous jump
%boundary conditions, given a normal diffusion process as input
%
% X is the (reps)xN vector of solutions to the diffusion process with jump
%
% B is the original diffusion process
%
% lb is the value of the lower boundary at which the jump will occur
% ljump is the value that is jumped to upon hitting the lower boundary
% ub is the value of the upper boundary at which the jump will occur
% ujump is the value that is jumped to upon hitting the upper boundary
%
% Lachlan Bridges
% 06/01/17

% case when no upper/lower boundary
if isempty(lb)
    lb = -inf;
end
if isempty(ub)
ub = inf;
end

N = size(B,2);
reps = size(B,1);

X=B;

ujs = ub - ujump;
ljs = lb - ljum;p;

for r=1:reps
    for k=2:N
        if X(r,k) >= ub
            X(r,k:end) = X(r,k:end)-ujs;
        end
        if X(r,k) <= lb
            X(r,k:end) = X(r,k:end)-ljs;
        end
    end
end

MATLAB Code 12: BMReflect.m

function X = BMReflect(B,lb)

%BMReflect generates reflected brownian motions given a non-reflected
%brownian motion as input
%
% X is the (reps)xN vector of solutions to the REFLECTED SDE at t
% B is the NON-REFLECTED SDE
% lb is the value of the lower bound for the REFLECTED SDE
%
% Lachlan Bridges
% 06/01/17

N = size(B,2);
reps = size(B,1);

X=zeros(reps,N);

%initial condition
X(:,1)=B(1,1);

for r=1:reps
    for k=2:N
        infm = -1*(min(B(r,1:k))-lb);
        X(r,k) = B(r,k) + max(0,infm);
    end
end
MATLAB Code 13: CompareBoundaries.m

% CompareBoundaries models and plots diffusion processes generated with
% MMMilstein, and applies a range of boundary conditions to these processes
% 
% Lachlan Bridges
% 14/01/17

seed=10113;
N=100;
r = 3; %number of realisations to plot
reps=10;

T=[0,30];
Q=[-3,3;1,-1];
x0=0;
mu = [5,-3];
sigma = [1,10];
drift = @(y,t,J) mu(J);
diff = @(y,t,J) sigma(J);
dbdy = @(y,t,J) 0;

% boundary conditions
lb=-20;
ub=[];
ljump = 10;
ujump = -10;

% cell of boundary condition functions and there names
boundtype=cell(2);
boundtype{1,1}='absorbing';
boundtype{1,2}='instantaneousjump';
boundtype{1,3}='reflected';
boundtype{1,4}='reflected';
boundtype{1,5}='sticky';

boundtype{2,1}=@(x,lb,ljump,ub,ujump) BMAbsorbing(x,lb,ub);
boundtype{2,2}=@(x,lb,ljump,ub,ujump) BMInstantJump(x,lb,ub);
boundtype{2,3}=@(x,lb,ljump,ub,ujump) BMReflect(x,lb);

% generating original diffusion process
[t, x, ~, ~] = MMMilstein(Q,reps,drift,diff,dbdy,x0,T,N,seed);

% used in titles and saving of plot
bounds=[lb;ub];
if numel(bounds)==1
str = sprintf('%i',bounds(1));
end
elseif numel(bounds)==2  
    str = sprintf('%i and %i', bounds(1), bounds(2));
end

% iterates through each boundary condition
for i=1:size(boundtype,2)
    y = boundtype{1,i}(x,lb,ljump,ub,ujump); % applying boundary condition
    plot(t(1,:),x(1,:), 'b-'), hold on
    plot(t(1,:),y(1,:), 'r-')
    xlabel('t', 'FontSize', 10)
    ylabel('X_t', 'FontSize', 10)
    title(sprintf('No boundary condition (blue)\n vs \n %s boundary condition
 (red)', boundtype{1,i}),...
         'FontSize', 8)
    saveas(gcf, sprintf('../figures/bound_%s_mu_%1.0f_%1.0f_sig_%1.0f_%1.0f_lbound_%i_ubound_%i.png'...
                      , boundtype{1,i}, mu(1), mu(2), sigma(1), sigma(2), lb, ub));
    hold off
end

% plotting, labelling and saving figures
figure;
ax1 = subplot(2,2,1);
plotmm(t(1:r,:),x(1:r,:))
ylabel('X_t', 'FontSize', 10)
ax2 = subplot(2,2,2);
[tm,xm] = meanmm(t,x,10000);
plot(tm,xm, 'b-')
ax3 = subplot(2,2,3);
plotmm(t(1:r,:),y(1:r,:))
ylabel('Y_t', 'FontSize', 10)
ax4 = subplot(2,2,4);
[tm,xm] = meanmm(t,y,10000);
plot(tm,xm, 'r-')
linkaxes([ax2,ax3,ax1, ax4], 'xy');
saveas(gcf, sprintf('../figures/bound_%s_comp_mu_%1.0f_%1.0f_sig_%1.0f_%1.0f_lbound_%i_ubound_%i.png'...
                      , boundtype{1,i}, mu(1), mu(2), sigma(1), sigma(2), lb, ub));
close all
function C = MMEuroCall( S, K, r, T, sigma, Q)
% MMEuroCall calculates the current price of a European call option
% MMEuroCall calculates the current price of a European call option
% based on the assumption that the underlying asset follows a
% Markov-modulated diffusion process
% C is the current price of the call option
% S is the initial price of the stock
% K is the strike price
% r is the risk free rate, in years
% T is the time until expiry, in years
% sigma is a vector of states that the volatility can be in (e.g. low and
% high volatility)
% Q is the transition rate matrix for the underlying Markov chain
% Lachlan Bridges
% (using McKinlay’s model from thesis 'Markov-Modulated Models for Derivatives Pricing')
% 16/01/17

lambda = Q(1,2);
u = Q(2,1);
v1 = sigma(1);
v2 = sigma(2);
[f1, ~] = OccTime(lambda,u);

int = @(s) blsprice(S,K,r,T,sqrt(((s*(v1^2)+(T-s)*(v2^2))/T))).*f1(s,T);
C = integral(int,0,T)+exp(-lambda*T)*blsprice(S,K,r,T,v1);

MATLAB Code 15: TestEuroCall.m

% TestEuroCall uses our implementation (MMEuroCall) of McKinlay’s model and
% compares the resulting price with the prices calculated by McKinlay to
% ensure our model is correct
% Lachlan Bridges
% 16/01/17

S0 = 50; % initial stock price
r = 0.1; % risk free interest rate
sigma = [0.25, 0.55]; % volatility
T = 1; % expiry

% strike prices to test:
K = [29.943175, 39.952443, 49.828298, 59.899056, 69.829658, 79.921838];
% correct prices (from McKinlay's thesis):
\[ m1 = [23.229596, 15.354187, 9.3150276, 5.2848860, 2.9956081, 1.7274812]; \]
\[ m2 = [23.568903, 16.353291, 10.919362, 7.0734063, 4.5946202, 2.9876057]; \]
\[ m3 = [23.104774, 15.020033, 8.8040807, 4.7166456, 2.4739770, 1.3000675]; \]
oprices = [m1 ; m2 ; m3];

% model paramaters:
lambda = [1, 3, 1];
nu = [1, 1, 3];

for i=1:3
    fprintf('MODEL %i:
',i)
    Q = [-lambda(i), lambda(i); nu(i), -nu(i)]; % transition matrix
    for j=1:length(K)
        C=MMeurocall(S0,K(j),r,T,sigma,Q);
        fprintf('McKinlay''s price: %f, Our Price: %f, Difference:%f
',...
            oprices(i,j),C,abs(C-oprices(i,j))
        )
    end
    fprintf('
')
end

fprintf('\n')