Statistical Model Checking for Algorithmic Trading Strategies via Simulation of Geometric Brownian Motion

Abhishek Bhargava
abharga2@andrew.cmu.edu

Michael You
myou@andrew.cmu.edu

December 6th, 2018

Abstract

Financial markets are becoming increasingly complex due to the rise of big data and the creation of more financial assets such as derivative securities, which are difficult to accurately model. We propose a hybrid systems based model that formally defines how a trading algorithm interacts with the market, and verifies the potential of the strategy using a variant of Statistical Model Checking. The goal in this formulation is to verify that a proposed allocation strategy has probability at most θ of losing money. Other properties can also be addressed in a similar fashion. An adversarial world state controller, or scheduler, is trained optimally using reinforcement learning to maximize the probability of the trader losing money. We then used statistical model checking to verify probability bounds on the success of various strategies. Finally, we implemented a trading algorithm based on the results of our simulations, and back tested it on historical data to determine its performance in the real world. The model-creation, formal verification, and algorithm implementation presented in our paper is a process that can be adopted by investment banks and other financial institutions as a more robust and efficient mechanism for managing risk and modeling evolution of securities under rapidly changing world states. It can be especially useful to gain quick feedback about trading strategies, learn where the trading strategies are losing money, and fix the issues, resulting in a better strategy overall. In the future, we hope to model the interaction between a trading strategy and a scheduler as a two-player zero-sum game, where both are competing against each other, in a generative adversarial network-style of modeling.

Introduction

Statistical Model Checking

Standard model checking encompasses a set of techniques that can be used to formally verify certain properties of models that are specified using state transitions, i.e. a deterministic finite automaton (DFA). Probabilistic Model Checking (PMC) [8] is a more general approach, where the goal is to find the probability that some property holds in a system. The advantage of PMC is that it accounts for nondeterminism. Unfortunately, when the number of states is very high, finding exact probability bounds becomes extremely difficult because it requires reasoning about the entire state space. A similar, and much more feasible approach, is Statistical Model Checking [6]. SMC is an approximation of PMC where simulation of the state space is combined with statistical methods to provide probabilistic guarantees of a particular property. SMC has the drawback of not being able to handle systems with nondeterminism, such as Markov Decision Processes (MDPs). Henriques et. al. [7] addressed this problem and applied SMC in MDPs by introducing the concept of an optimal adversarial "scheduler" that is allowed to resolve the nondeterminism. It can be shown that an optimal adversarial deterministic scheduler is at least as good as any probabilistic or nondeterministic scheduler, and it is therefore sufficient to only consider deterministic schedulers when choosing such an adversarial scheduler.

Exact methods in PMC find the optimal scheduler by propagating information through the state space. Here, we will follow Henriques et al.'s approach of using a guided search via reinforcement learning to find the optimal scheduler. Given the optimal scheduler, we no longer have any nondeterminism in the system since the scheduler resolves the nondeterminism, so we can then use standard SMC to find the probability that the system satisfies desired properties.

Finance

There are many models we could use for the evolution of a stock price. The downfall of most of these models and most trading strategies in general is not necessarily that the strategy or model is bad; it is that the model only holds under certain conditions, which we define here as a "world state." Different models may be more effective in different world states, and there is currently no robust and scalable technique to manage a variety of different models under different world states. SMC provides an opportunity to do just that by using a series of state transitions (including but not limited to MDPs). Here, we assume that there are some different world states, and we transition from one world state to another nondeterministically. Each world state can be described by the models and parameters that apply in the world state and the specific characteristics that define the world state. In our model, we also assume we can trade a fixed number of stocks and we can hold either a long or short position in any of them. Furthermore, there exists a risk-free asset that we can trade at any time, similar to a bank. These are standard assumptions of most financial models.

Motivation of Approach

Below, we discuss the approach we use for SMC. To motivate this, we provide relevant definitions in this section. First, in finance, a trading strategy is characterized by an initial capital X_0, and a process of allocations of wealth to various assets. This allocation of wealth at time t is often represented by a vector $\alpha_t \in \mathbb{R}^{p\times 1}$ such that $|\alpha_t|_1 = 1$. Here, p is the number of risky assets and the dimension of α_t is $p \times 1$ because there is an additional risk-free asset that can be thought of as the "bank account." This process of allocation vectors comprises the foundation of our discrete control, discussed in more detail below. Furthermore, in finance it is usually unclear what the state of the world is, in terms of which stocks are going up and which stocks are going down. We can discretize the world into different states that capture such relationships.

SMC Approach

There are three main components to the general scheme we will use for SMC. We formulate these components as a hybrid program in the formal problem description:

1. A discrete control where we choose the proportion of our wealth we want to allocate to each of the stocks and risk-free asset in our model. This represents how a trader would make trades in real life.

   (I'm not sure we need this, as we can explain it later) Notice that we require the proportions to sum to 1. That is, we cannot hide money "under the mattress," a standard assumption in portfolio theory.

2. A nondeterministic choice between world state transitions. World states correspond to what behavior stocks will have in the market. For example, there can be a state where tech stocks are dropping. Therefore, this component models behavior that markets can potentially have.

3. Continuous evolution of the stock prices as correlated geometric Brownian motions (GBMs), described by stochastic ordinary differential equations. In finance, GBMs are used extensively to model stock price behavior [1]. We will describe GBMs in further detail in the following section.

Geometric Brownian Motion

Brownian Motion

A Brownian motion is a continuous stochastic process that is widely used in physics and finance to describe random evolution of a system over a period of time [1]. A Brownian motion $(B_t)_{t\geq 0}$ is defined as follows:

1) $B_0 = 0$, and B_t is defined for all $t \geq 0$.
2) All realizations of a Brownian motion are continuous in t. More formally, with probability 1 the function $t \to B_t$ is continuous in t.
3) The process $(B_t)_{t\geq 0}$ has stationary and independent increments.
4) The increment $B_{t+s} - B_s$ is such that $B_{t+s} - B_s \sim N(0, t)$, i.e. normally distributed.

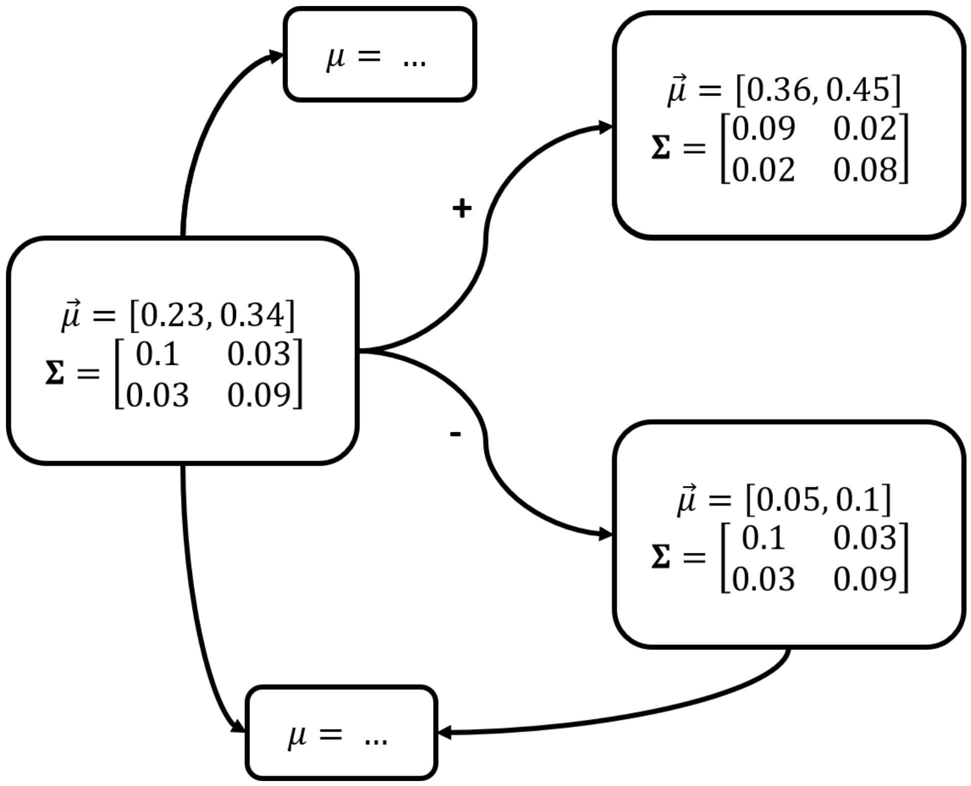
Geometric Brownian Motion

In GBMs, the logarithm of the time-varying quantity follows a Brownian motion by definition. The most widely used model in finance is the Black-Scholes model for options pricing [2]. It makes many assumptions, one of which is that stock prices follow a geometric Brownian motion. We will use that assumption in the concrete models we discuss in this paper, but this assumption is *not* necessary for the general premise of our work to hold. In particular, we can specify models within this framework that allow the stock price to evolve in arbitrary ways. If we let S_t denote the price of a stock at time t, stock price evolving as a geometric Brownian motion can be described by the following stochastic differential equation, where μ represents the expected log returns for the stock, and σ represents the volatility, or standard deviation, of the log returns:

$$\mathrm{d}S_t = S_t(\mu\,\mathrm{d}t + \sigma\,\mathrm{d}B_t). \tag{1}$$

This stochastic differential equation models the local movement of a stock price. It has been shown that geometric Brownian motions [13] are fairly accurate models for stock price evolution. Furthermore, this is an underlying assumption of the Black-Scholes model, which is extremely popular in industry.
Notice that this model makes the assumption of a constant σ, which means it assumes that volatility is constant. This is not the case in practice, and various so-called stochastic volatility models, such as the GARCH model [3], have attempted to address this flaw in the Black-Scholes model.

Our methodology is another way to address this flaw. We can specify different world states and different values for σ in each world state. Therefore, this is analogous to the benefit we gain from using differential equations, as opposed to a global solution to a differential equation, to describe system evolution. We can think of a general stochastic volatility model as a global solution, while our world-state-based model is more like a local description of how volatility evolves. It also allows us to capture complex, inexplicable nonlinear relationships that stochastic volatility models cannot generally capture. For example, if for some unknown reason volatility of Apple stock spikes when the S&P500 has low correlation with the technology industry, a world-state-based model can capture this anomaly. A stochastic volatility model, however, cannot. This is because stochastic volatility models assume global patterns and cannot capture local anomalies.



$\mu = ...$
$\vec{\mu} = [0.36, 0.45]$
$\mathbf{\Sigma} = \begin{bmatrix} 0.09 & 0.02 \\ 0.02 & 0.08 \end{bmatrix}$
+
$\vec{\mu} = [0.23, 0.34]$
$\mathbf{\Sigma} = \begin{bmatrix} 0.1 & 0.03 \\ 0.03 & 0.09 \end{bmatrix}$
-
$\vec{\mu} = [0.05, 0.1]$
$\mathbf{\Sigma} = \begin{bmatrix} 0.1 & 0.03 \\ 0.03 & 0.09 \end{bmatrix}$
$\mu = ...$

Figure 1: Stochastic Transitions

Problem Definition

Motivation

In finance, extensive work is done by hedge funds and investment banks to estimate the risk they are taking on in their portfolios [4, 9]. In general, firms that manage risk more efficiently tend to have more effective trading strategies under changing market conditions, and therefore make more money overall. Here, we propose a robust method to manage risk and various other properties of portfolios using statistical model checking (SMC). These properties can range from guaranteeing certain types of behavior under different market conditions, to verifying loading (i.e. market beta) to various factors, to making sure the probability of a particular strategy losing money is less than θ.

To handle nondeterminism of price evolution, which exists in finance, we will use an adversarial scheduler that resolves the nondeterminism by choosing the worst possible world state transitions for the trader. In this paper, when we talk about a "realization" of nondeterminism, we really mean a schedule, or sequence of world states that the system goes through. The motivation behind our modified SMC approach is that PMC requires properties to hold in all realizations of nondeterminism — this is analogous to proving properties about the box operator in differential dynamic logic when the hybrid system contains nondeterministic choice. Because there are far too many realizations of nondeterminism to consider for our model, we have to consider another option. SMC does not have this rigid requirement and is therefore a better choice for finance applications due to the high degree of uncertainty and number of states we have to consider to obtain a realistic model. To use SMC, we need a way to remove nondeterminism from the model, which is what the scheduler in our model does; the scheduler makes the choice whenever the model faces a nondeterministic step.

Our model also has the advantage that it allows us to localize the randomness of the world by considering separate world states. This is analogous to the difference in complexity between describing a global solution to an ordinary differential equation and describing it locally using a differential equation. Another nice feature is that we do not require a model for how the world evolves from one state to another, because this leads to additional sources of error and uncertainty. Maybe we know things about certain world states, and not others, like an evolution domain constraint, and so we can only trust a model in certain states. Such notions are easily described in this framework.

As a concrete example, one world state for the stock market might be when the S&P500 is losing value quickly, and another might be when the S&P500 is gaining value quickly. We can specify different models for how Apple stock evolves in each of these market conditions, that goes beyond traditional geometric Brownian motions or even stochastic volatility models.

Assumptions

In order to choose a model we can practically implement and simulate, we will make the following assumptions in our problem:

1. **We have one risk-free asset**, which we will refer to as S_0. This asset can be thought of as the "bank", and is an asset that never decreases in value. The risk-free asset increases in value at a growth rate of r. More formally, if we invest X_0 dollars in this risk-free asset at time 0, it will be worth $X_0 e^{rt}$ dollars at time t.

2. **We have seven stocks**, These stocks have prices that evolve over time in ways that can depend on the world state $\mathcal{J}$.

3. **There are four world states** that correspond to whether the stock prices are increasing or decreasing from one period to the next:

World State	S&P500	S&P500 50-Day Moving Average
$\mathcal{J}_1$	Increasing	Above
$\mathcal{J}_2$	Increasing	Below
$\mathcal{J}_3$	Decreasing	Above
$\mathcal{J}_4$	Decreasing	Below

   The other case is that we have **eight world states**, where we define them similarly to the four world states above. The only difference is that instead of just having an increasing and ecreasing label on the behavior of the S&P500, we looked at many S&P500 behaviors, sorted them from most decreasing to most increasing behavior, and then divided these behaviors into eight buckets. These buckets were then used to define our eight world states. The reason for having more world states is to give us a more granular model, which can give us more complex strategies.

4. **There is a trader** that tries to minimize the probability that $X_T < X_0$, where X_T is the portfolio value at time T and X_0 is the initial portfolio value, where T is the duration that the trading strategy will be used.

5. **All of the portfolio value is allocated** between the risk-free asset and the stocks (i.e. no money is sitting anywhere else).

Overview of Problem

In this paper, we will first lay out definitions of variables used in our problem. Then, we will construct a stochastic hybrid program that describes how the trader interacts with the market, as well as how the stock prices and portfolio values evolve over time.

In particular, the hybrid program can be described by 3 major components:

1. The trader chooses his allocation strategy. This involves the trader looking at the world state, and the transition matrix of the world state, and determining how much of his portfolio to allocate into each asset.
2. The parameters for stock evolution are chosen. This step involves choosing the next world state to evolve to, setting μ, r, σ, ρ and other parameters used in the differential equation based on that world state. These parameters are chosen based on historical data. In particular, we obtained data from Yahoo Finance for various stocks from January 1st, 2005 to December 31st, 2017, segmented it by our defined world states, and computed the mean return vectors and covariance matrices of returns for each world state.
3. Evolution of the Stochastic Differential Equations (SDEs). The SDEs determine the stock price behavior over time, and will evolve for some time T_0, the amount of time the world stays in each world state. This interval can be any amount of time depending on our view of how long the world stays in each state.

Finally, we can use our hybrid program syntax and semantics from class to formulate our ultimate question of whether or not the trader has an allocation strategy that guarantees success with at least some probability $1 - \theta$.

Model

Here we describe the general model used for our market simulator. We will define each component of the model formally, and then put all the parts together in a hybrid program at the end.
In our concrete model presented here, we only have two risky assets and a risk-free asset. This is a simplification for the sake of explaining our model completely. However, our model can be extended to have an arbitrary N_S number of stocks. In our actual modeling, the number of stocks was a parameter, so we present a general model here rather than a separate model for each of the hundreds of strategies we tried.

Definitions

Here we formally define the variables and notation used throughout the paper.
First, we add the $[\alpha]_\theta P$ operator to the syntax of stochastic differential dynamic logic (Sd$\mathcal{L}$) [12]. Assuming a finite state space, which we have in our models, We define the semantics as follows:

$$[\![[\alpha]_\theta P]\!] = \{\omega \mid \nu \in [\![P]\!] \text{ for at most } \theta \text{ proportion of all } \nu \text{ such that } (\omega, \nu) \in [\![\alpha]\!]\} \tag{2}$$

Then, we have the following definitions:

Definition 1. The **allocation** $\vec{\alpha}$ is a vector that represents the distribution of a portfolio's value among the risk-free asset, and N_s stocks. For all $i \in [0, N_s]$, $\vec{\alpha}_i$ is the proportion of the portfolio's value that is invested in asset i, where asset 0 is the risk-free asset. Therefore, $\vec{\alpha}$ is a $(N_s + 1) \times 1$ vector, and

$$\sum_{i=1}^{N_s+1} \alpha_i = 1, \tag{3}$$

where $\alpha_i \in \mathbb{R}$.
Note that α_i can be negative, which corresponds to *shorting* an asset in financial terms.

Definition 2. The risk-free asset has a **continuous risk-free rate** represented by $r \in \mathbb{R}$, which is the rate at which money in the risk-free asset accumulates interest over time.

Definition 3. $\mu \in \mathbb{R}$ is the **annualized drift term** in the stochastic differential equation for geometric Brownian motion. This can be approximated from daily stock price data by finding the mean of log returns and then multiplying by the number of trading days in one year, usually taken to be 252 days. This is the expected annual return.

Definition 4. $\sigma \in \mathbb{R}$ is the **annualized volatility term** in the stochastic differential equation for geometric Brownian motion. This can be approximated from stock price data by finding the standard deviation of daily log returns and then multiplying by the square root of the number of trading days in one year, usually taken to be 252 days. This is the annual standard deviation, or volatility, of returns.

Definition 5. The **transition matrix C** is a $N_{\mathcal{J}} \times N_{\mathcal{J}}$ matrix, where $N_{\mathcal{J}}$ is the number of world states. In our concrete model $N_{\mathcal{J}}$ is 4. The transition matrix entries are as follows:

$$\mathbf{C}_{i,j} = \begin{cases} 1, & \text{if transition } \mathcal{J}_i \to \mathcal{J}_j \text{ is possible} \\ 0, & \text{if transition } \mathcal{J}_i \to \mathcal{J}_j \text{ is not possible} \end{cases} \tag{4}$$

Specifically, for our choice of world states, we have

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{5}$$

The intuition behind this transition matrix comes from finance. Recall that world state 0 corresponds to a period where the S&P500 increased in value and had a price above its 50-day moving average. World state 1 corresponds to the S&P500 increasing in value but at a price below its 50-day moving average. World state 2 corresponds to the S&P500 decreasing in value at a price above its 50-day moving average. Finally, world state 3 corresponds to the S&P500 decreasing in value at a price below its 50-day moving average. From state 0, since the stock price increases from a point above its 50-day moving average, it is unlikely for the S&P500 to suddenly drop below its 50-day moving average, so it cannot go to states 1 or 3. From states 1 and 2, virtually all transitions are possible. From state 3, since the S&P500 decreases in value from a point below its 50-day moving average, it is unlikely for the price to suddenly jump above its 50-day moving average.

Geometric Brownian Motion

Using geometric Brownian motion to model the stock price evolution is an important part of our model, although arbitrary evolution models are allowed. Here, we will define the variables related to geometric Brownian motion.

Definition 6. We assume the stock prices follow geometric Brownian motions that are correlated, since in practice stock prices are correlated. This evolution can be described using the following differential equations for $i \in [N_s]$:

$$\mathrm{d}S_t^i = S_t^i(\mu_i \,\mathrm{d}t + \sigma_i \,\mathrm{d}B_t^i) \tag{6}$$

We represent the Brownian motion associated with stock S^i as $(B_t^i)_{t\geq 0}$.

Collectively, the Brownian motions for the different stocks may be correlated, and can be defined as

$$\vec{B} \sim \mathsf{BM}(0, \Sigma), \tag{7}$$

where BM is Brownian Motion, and

$$\Sigma = \mathbf{A}\mathbf{A}^T$$

is the Cholesky Decomposition, which is generally used to find a numerical solution of linear equations, but is helpful in this case to express the solution of the differential equations that describe correlated geometric Brownian motions.

Definition 7. We define ρ to be the correlation of the Brownian motions. That is,

$$dB_t^1 dB_t^2 = \rho \,\mathrm{d}t \tag{8}$$

Hybrid Program for Problem Description

```
/* Assumptions and Initial Conditions */
PRE ::=
(
    $T_0$ > 0 & /* Amount of time evolution for ODEs in each iteration */
    $X$ > 0 & /* Portfolio value */
    $X_0$ = $X$ & /* Starting portfolio value */
    $\mathcal{J} \in \{\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \mathcal{J}_4\}$ & /* Starting world state */
    $S_1$ > 0 & $S_2$ > 0 /* Stock prices */
)

SHP ::=
(
    /*** Allocation Strategy by Trader ***/
    $\alpha_0$ := $f_0(\mathcal{S}, \mathbf{A})$;
    $\alpha_1$ := $f_1(\mathcal{S}, \mathbf{A})$;
    $\alpha_2$ := $f_2(\mathcal{S}, \mathbf{A})$;
    ?($\alpha_0 + \alpha_1 + \alpha_2 = 1$); /* Valid allocation */

    /*** Set up world state and associated parameters ***/
    {$\mathcal{J} := \mathcal{J}_1$ ++ $\mathcal{J} := \mathcal{J}_2$ ++ $\mathcal{J} := \mathcal{J}_3$ ++ $\mathcal{J} := \mathcal{J}_4$};

    /* choose parameters based on world state */
    {
        {
            ?($\mathcal{J} = \mathcal{J}_1$);
            $\mu^1$ := $\mu_1^1$; $\mu^2$ := $\mu_1^2$;
            $\sigma^1$ := $\sigma_1^1$; $\sigma^2$ := $\mu_1^2$;
            $r$ := $r_1$;
            $\rho$ := $\rho_1$;
        } ++
        /* Similarly for other world states $\mathcal{J}_i$ */
    }

    /*** SDEs ***/
    {
        /* Portfolio value */
        $\mathrm{d}X = \alpha_0\,\mathrm{d}t + \alpha_1\,\mathrm{d}S_t^1 + \alpha_2\,\mathrm{d}S_t^2$,

        /* Stock values */
        $\mathrm{d}S_t^1 = S_t^1(\mu_1\,\mathrm{d}t + \sigma_1\,\mathrm{d}B_t^1)$,
        $\mathrm{d}S_t^2 = S_t^2(\mu_2\,\mathrm{d}t + \sigma_2\,\mathrm{d}B_t^2)$

        /* Maintains Brownian motion correlation */
        & $\mathrm{d}B_t^1\,\mathrm{d}B_t^2 = \rho\,\mathrm{d}t$
    }
)$^*$

/* Losing condition */
$\varphi$ ::= $X < X_0$ (with probability $\theta$)
```

Market Simulation

Identifying World State Parameters

The first step of the simulation was to identify the parameters that characterized each world state. We took the approach of using historical data to compute the parameters. In particular, we used data from January 1st, 2005 to December 31st, 2017, segmented it by which world state it falls into, and then computed the aggregate returns for each world state. We then used the aggregate returns to find μ_i and Σ_i for each stock i that we were considering.

Constructing Trading Strategies

We constructed many different types of trading strategies to test our simulation on. There are three main categories of strategies that we looked at.

Buy and Hold

In the buy and hold strategy, we simply bought equal proportions of each stock at time 0 and held each stock until the end of the simulation.

Mean-Variance Optimal

To compute mean-variance optimal strategies, we assumed a certain investment in the risk-free asset and then optimized the allocation of remaining portfolio wealth among the risky assets. This is a standard optimization problem in finance, and so the details of the optimization are omitted. For interested readers, we computed the portfolio that maximizes Sharpe ratio using Markowitz mean-variance portfolio optimization.

Proportional Allocation

This was perhaps the simplest strategy, where we simply allocated funds based on the proportion of returns each stock had in a particular world state. In world states where the stock prices were likely to go down, our strategy was to short the risky assets and invest more money in the bank, while in other world states, we simply normalized the mean returns vector and used that as our portfolio weights.

Simulating Correlated Geometric Brownian Motions

We simulated correlated geometric Brownian motions by simply simulating their solutions. In particular, we used the following update rule to construct a time series of simulated stock prices (one realization of the correlated GBM):
Let i denote the risky asset, and let t denote the time period for which we are generating prices. Here, since we generated prices for one day at a time, we took $t = \frac{1}{252}$ since there are 252 trading days in a year, and one year represents $t = 1$ by convention. Furthermore, A_i is the ith row of the Cholesky decomposition matrix A from earlier. z_{t+1} is a $p \times 1$ vector of independent standard normal variables.

$$S_{t+1}^i = S_t^i * \exp\left\{\sqrt{t} * (A_i^T * z_{t+1}) + t * (\mu_i - \frac{\Sigma_{ii}^2}{2}\right\} \tag{9}$$

There are very few available resources detailing this equation. It comes from the solution to the geometric Brownian motion differential equation. The Cholesky matrix A can be thought of intuitively as the square root of the covariance matrix of the Brownian motions.

Complete Overview

The steps in the simulation involve the following:

1. First, the scheduler gives the simulation a set of transitions that it decided on.

2. For each transition, the trader chooses how to allocate wealth to the risky assets and risk-free asset. Note that the trader does not know which world state the world is currently in.
3. Every stock in the model is simulated based on the world state for some length (this is a parameter, set to 20 days (about 1 trading month) in our model).
4. Log returns for the stocks are computed.
5. The dot product of the allocation vector and the log stock returns is computed to find the return for the portfolio.
6. At the end of the simulation, the mean portfolio returns and the standard deviation, or volatility of the portfolio is computed. Using this, the Sharpe ratio is computed.

Statistical Model Checking

After we have defined the model for the market and how the trader manages assets within the market, we want to verify properties about our model.

Specifically, since we want to place a probability bound on how likely the trader is to fail, Using our new semantics definition, we can then describe our problem as verifying the following formula,

$$\mathsf{PRE} \to [\mathsf{SHP}]_{\theta}\varphi \tag{10}$$

i.e. the trader's allocation strategy, or trading algorithm, loses money with probability $\leq \theta$.

In order to perform this verification, we will use SMC. Although SMC does not support nondeterminism, which is exhibited in our program, [7] describes a workaround for using SMC to verify a nondeterministic system. The idea is to use the optimal adversarial scheduler against the trader, since that will get rid of nondeterminism. It can be shown that an optimal deterministic adversarial scheduler is at least as good as any other optimal scheduler. Then, we can use Monte Carlo (MC) methods to verify probability bounds about the desired condition.

In the following sections, we will describe how the work in [7] can be adapted to our problem so we can leverage their methodology.

Scheduler Optimization

The goal of this step is to make our scheduler be optimally adversarial, i.e. maximize the probability θ of our trader losing money and satisfying $X < X_0$. We can do this by training the scheduler via reinforcement learning, which involves improves the scheduler by giving positive feedback on good moves that the scheduler makes, and negative feedback on bad moves.

The algorithm for optimizing the scheduler involves alternating between two procedures:

1. **Scheduler Evaluation:** In this step, we run the scheduler against a trading strategy many times in the simulation. Then, we can find quality estimate values that will be used in the next step to improve the scheduler.

   One slight modification that we madde to our reinforcement learning algorithm that is different from Henriques et al. is that instead of just incrementing $R^{+}(s,a), R^{-}(s,a)$ values by 1, we increment them by

$$\hat{R} = \frac{X - X_0}{X_0}, \tag{11}$$

or the percentage return the trader's portfolio received during the simulation.

The reason for this change is that even though we only care about the truthness of $X < X_0$, we want to give more reward for strategies that make the trader lose more money. This also has an analog to real life, where losing more money is less desirable than losing less.

```
def schedulerEvaluate(scheduler):
    for i in range(SCHEDULER_TRAIN_ITER):
        Generate moves with scheduler

        # Run simulation with moves that the scheduler has chosen
        success = sim(moves)

        Update $R^+(s,a), R^-(s,a)$ Reinforcement feedback values based on success

    Update scheduler quality estimates Q using $R^+(s,a), R^-(s,a)$

    return Q
```

2. **Scheduler Improvement:** Once we have our scheduler evaluation, we can improve our scheduler by greedily updating our scheduler so that it tries to choose world states that are most likely to satisfy its goal. The algorithm is described below:

```
def schedulerImprove(scheduler, Q):
    # Fill in newScheduler based on new quality estimates
    for s, a in (STATES, ACTIONS):
        # the probability we are assigning to the action
        p = EXPLORATION_RATE * (Q[s][a] / totalQ)

        if a == bestAction:
            p += 1 - EXPLORATION_RATE

        newScheduler[s][a] = scheduler[s][a] * (1 - LEARNING_RATE) + p * LEARNING_RATE

    return newScheduler
```

Finally, we have code that alternates between the scheduler evaluation and improvement

```
def main();
    Load scheduler file

    # train the scheduler
    for in in range(NUM_TRAIN_ITER):
        # Compute new quality factor
        Q = schedulerEvaluate(scheduler, moves, result)

        # Update scheduler with new quality estimates
        scheduler = schedulerImprove(scheduler, Q)

    Save scheduler file
```

SMC

As defined earlier in (10), we want to analyze the probability θ of $\varphi \equiv X < X_0$, after a run of the market simulation. In order to implement this, we will write a Python program that implements our market simulation, and allows the scheduler to choose which world state to go into. Then, we can write the SMC algorithm in [7] as a wrapper that integrates the optimization of the scheduler, choosing the world state, and testing if we satisfy φ.

Sampling

Our SMC algorithm runs the stock market simulation with the trained adversarial scheduler and a trading strategy. The evaluation algorithm we use is essentially a Monte Carlo method to figure out the probability we satisfy property φ.

```
def SMC(scheduler, strategy):
    win = 0

    for NUM_SAMPLES:
        if simulate(scheduler, strategy) == success:
            win += 1

    return win / NUM_SAMPLES
```

With this, we can get an estimate of our p value of the Monte Carlo Algorithm.
We ended up having to modify the SMC algorithm used by Henriques et. al. This is because our trained scheduler was extremely good at finding sequences of transitions that almost always made the trader lose. Intuitively, this makes sense because the trader has some amount of uncertainty about future states, so the trader cannot act optimally in all states. So after training the scheduler, we used Monte Carlo simulations to test the probability that the trader has a Sharpe ratio less than or equal to some value. Since the scheduler wins almost 100% of the time using a Sharpe ratio cutoff of 0, we instead set the probability bound to 40%, and determined the Sharpe ratio cutoff for which the trader would have a smaller Sharpe ratio with probability less than or equal to 40%. We then used this Sharpe ratio cutoff to evaluate various trading strategies. The higher the cutoff, the better the strategy.

Performance

Because the simulation of 100 actions by the scheduler takes around 2 seconds to run on an average laptop, training our model without making our code faster was unfeasible. In order to make our code faster, we employed parallelization techniques:

1. **Running Simulations in Parallel**: Because the scheduler does not change during all of its simulation runs within an epoch, these runs can be run in parallel. In order to do this, we employed the `multiprocessing` library, and spun up processes for each simulation iteration.

2. **Vectorizing code**: Because we were using the `numpy` library for Python, we wanted to take advantage of vectorized operations in the package as much as possible. Therefore, we tried techniques such as representing our data structures in `numpy` arrays, and avoided using for-loops when possible:

```
# R is a dictionary of (state, action) -> [int1, int2]
for state in NUM_WORLD_STATES:
    for action in NUM_WORLD_STATES:
        total = (R[(state, action)][0] + R[(state, action)][1])

        if total == 0:
            Q[state][action] = 0
        else:
            Q[state][action] = (R[(state, action)][1] / total)
```

Listing 1: Old code

```
# R is a 2xNUM_WORLD_STATESxNUM_WORLD_STATES array
Q = divide(R[1], R[0] + R[1], where=(R[0] + R[1])!=0)
```

Listing 2: New code

Vectorization techniques only made minor improvements to time performance, but significantly improved code concision and readability, and also increased potential for parallelism if sizes of data structures were to expand in the project.

Our changes significantly improved the speed of our training. For example, for 5 epochs of 50 simulation iterations, we were able to go from 15 minutes with sequential code to 3 minutes with parallelism.

Trading Strategy Improvement Pipeline

Now that we have described our market simulation, trader strategy and scheduler optimization, we want to put all these pieces together to make a process to help us develop and improve trader strategies. Below, is the basic flowchart for the trader strategy improvement process.

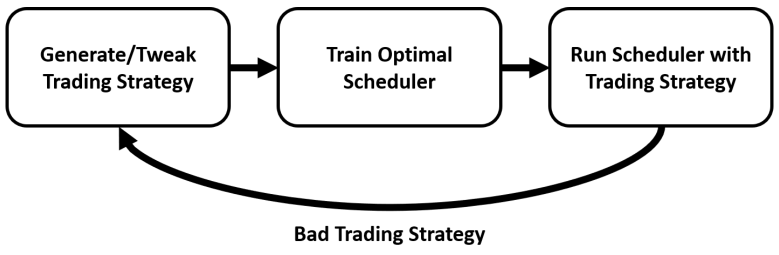

Generate/Tweak Trading Strategy
Train Optimal Scheduler
Run Scheduler with Trading Strategy
Bad Trading Strategy

Figure 2: Flowchart for improving trader strategy

The idea is to generate the strategy, then train the optimal scheduler so we can figure out where our strategy has gaps, i.e. if the scheduler is able to take advantage of weaknesses in our strategy. Then, after figuring out what is wrong, we can adjust the trader by hand to counteract the optimal scheduler strategy, and repeat the cycle. This process can be repeated many times to produce a very robust trading strategy.

Results: 4 World States

Using our market world state simulation code, we were able to construct 3 trading strategies:

1. **Buy and Hold:** As the name suggests, buying assets, and not selling them until the very end of the simulation.
2. **Mean Variance Optimal:** A simple strategy that looks at the risk associated with stocks.
3. **Short Down**: Performs shorts when it is in a world state of falling prices, and buys long when the world state indicates rising prices.

We will now present the parameters that were calculated, what the optimal adversarial scheduler was to each of these trading strategies and its success bounds, and what suggestions we can take from the scheduler to improve the strategy.

For all runs, we used the following parameters:

- `TIME_PERIOD`: 500, number of time periods to simulate
- `EXPLORATION_RATE`: 0.05, how likely we are to try out non-optimal actions
- `LEARNING_RATE`: 0.5, what proportion of newly calculated transition values should replace the old
- `OPTIMIZATION_ITER`: 100, the number of times we evaluate and improve our scheduler
- `MONTE_CARLO_ITER`: 1000, the number of samples to perform Monte Carlo analysis

In addition, because we were able to find the optimal scheduler for all of the strategies, in order to evaluate how good each strategy was, we define the following metric:

Definition 8. The **Trading Strategy Metric** m indicates how good a trading strategy is based on the optimal adversarial scheduler, where

$$\Pr(\text{Scheduler wins and } \texttt{sharpe} < -m) = 0.4 \tag{12}$$

Intuitively, it is harder for the Scheduler to win with a lower m value, since that means it has to make the trader lose more money.
We also used the following stocks

$$\texttt{AAPL}, \texttt{MSFT}, \texttt{GOOG}, \texttt{F}, \texttt{JNJ}, \texttt{JPM}, \texttt{XOM} \tag{13}$$

for our simulations.

Buy and Hold

Talk about the parameters that were generated and intuitively why

Optimal Adversarial Scheduler

One of the optimal adversarial schedulers found for the buy and hold strategy was

$$\mathbf{C}^*_{\mathsf{BH}} = \begin{bmatrix} 0.26202832 & 0 & 0.73797168 & 0 \\ 0.13087403 & 0.13099891 & 0.13144896 & 0.6066781 \\ 0.1308839 & 0.13100446 & 0.1314959 & 0.60661574 \\ 0 & 0.26198099 & 0 & 0.73801901 \end{bmatrix} \tag{14}$$

Running Monte Carlo sampling on the success of this scheduler with 1,000 iterations gives us

$$p_{\mathsf{BH}} = 1, \tag{15}$$

Comparatively, an untrained scheduler that chooses transitions at uniform has a success rate of 0.2994.
The metric for buy and hold is

$$m_{\mathsf{BH}} = -1.30. \tag{16}$$

Because the scheduler defined by (14) never loses, we conclude Buy and Hold is a bad trading strategy. Intuitively, the scheduler is what we would expect the optimal adversarial scheduler to be, since world states 2 and 3 mean falling stock prices, which buy and hold has no protection against, since the strategy relies on the fact that stock prices will go up in the long run.

Mean Variance Optimal

The Mean Variance Optimal (MVO) strategy considers the expected value of stocks and associated risks.

Optimal Adversarial Scheduler

One of the optimal adversarial schedulers found for the buy and hold strategy was

$$\mathbf{C}^*_{\mathsf{S1}} = \begin{bmatrix} 0.26221818 & 0 & 0.73778182 & 0 \\ 0.13105491 & 0.1310813 & 0.13136782 & 0.60649596 \\ 0.1310894 & 0.13114105 & 0.13128459 & 0.60648495 \\ 0 & 0.2621511 & 0. & 0.7378489 \end{bmatrix} \tag{17}$$

Running Monte Carlo sampling on the success of this scheduler with 1,000 iterations gives us

$$p_{\mathsf{MVO1}} = 1. \tag{18}$$

Comparatively, an untrained scheduler that chooses transitions at uniform has a success rate of 0.498.
The metric for MVO is

$$m_{\mathsf{MVO1}} = -1.33. \tag{19}$$

Because the scheduler defined by (17) never loses, we conclude MVO is a bad trading strategy. Intuitively, the scheduler is what we would expect the optimal adversarial scheduler to be, since world states 2 and 3 mean falling stock prices, which, like similar to buy and hold, MVO has no protection against, since the strategy relies on the fact that stock prices will go up in the long run.

Short Down

Short down is a trading strategy that utilizes both long and short trades. This strategy is well-adapted to changing market conditions, as it can work in both markets with rising or falling stock prices.

Optimal Adversarial Scheduler

The scheduler converged to the following:

$$\mathbf{C}^*_{\mathsf{SD}} = \begin{bmatrix} 0.26138116 & 0 & 0.73861884 & 0 \\ 0.13084581 & 0.13060336 & 0.13170011 & 0.60685072 \\ 0.13138537 & 0.60652355 & 0.13089041 & 0.13120066 \\ 0 & 0.73816687 & 0 & 0.26183313 \end{bmatrix} \tag{20}$$

Running Monte Carlo sampling on the success of this scheduler with 1,000 iterations gives us

$$p_{\mathsf{SD}} = 1. \tag{21}$$

Comparatively, an untrained scheduler that chooses transitions at uniform has a success rate of 0.0688.
The metric for Short Down is

$$m_{\mathsf{SD}} = -1.07. \tag{22}$$

In summary, the 4-world state cases had the following results:

	Buy and Hold	MVO	Short Down
m	-1.3	-1.33	-1.07

Results: 8 World States

After considering a world with 4 states, we now expand our model to have 8 states to give the scheduler more granularity in making its choices, and also provide more insight into a more complex system.

MVO

Our first trading strategy is the mean variance optimal, which is a strategy that we also tested in the 4 world-state case.
The following matrix is the optimal scheduler for the MVO 8-world state case:

$$\mathbf{C}^*_{\mathsf{MVO}} = \begin{bmatrix} 0.7375434 & 0.2624565 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.6500701 & 0.1750553 & 0.1748744 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.6501701 & 0.1749996 & 0.1748302 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.6501477 & 0.175019 & 0.1748328 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.6501338 & 0.1749762 & 0.1748899 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.6500916 & 0.1750035 & 0.1749047 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.6500665 & 0.1749957 & 0.174937 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.7375929 & 0.2624070 \end{bmatrix} \tag{23}$$

The metric for MVO is

$$m_{\mathsf{MVO}} = -2.05. \tag{24}$$

Long-Short Strategy

The long short strategy is similar to the short down strategy for the 4-world case. The strategy is very flexible in that it can potentially make money in markets that have both rising and falling prices.
The following is the optimal scheduler for the Long Short 8-world state case:

$$\mathbf{C}^*_{\mathsf{LS1}} = \begin{bmatrix} 0.14104432 & 0.85895568 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.09394089 & 0.33353759 & 0.57252152 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.80958791 & 0.09555559 & 0.0948565 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3342015 & 0.57360287 & 0.09219562 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.33448719 & 0.57206721 & 0.09344561 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.33499361 & 0.5777664 & 0.08723999 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.57333361 & 0.09823814 & 0.32842825 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.37488016 & 0.62511984 \end{bmatrix} \tag{25}$$

The metric for Long Short is

$$m_{\mathsf{LS1}} = -2.20. \tag{26}$$

Improved Long-Short Strategy

After training the Long Short Strategy, we wanted to test out our pipeline for improving our strategy by tweaking some parameters to counteract the scheduler. The tweaks that were made:

1. We gave the trader a transition matrix that gave the trader a hint of what the scheduler would more likely transition to, so the trader could potentially prepare for future world states.
2. We did not short when it was risky to do so, and opted to put the money in risk-free.

However, even though our parameter tweaks did do better at first, the reinforcement learning algorithm was still able to find the optimal scheduler for the Improved Long Short 8-world state case:

$$\mathbf{C}^*_{\mathsf{LS2}} = \begin{bmatrix} 0.0235401 & 0.9764598 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0912971 & 0.8921339 & 0.0165689 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5748540 & 0.1646162 & 0.2605296 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.6648248 & 0.3222536 & 0.01292157 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.6895103 & 0.0620642 & 0.2484254 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5964237 & 0.1387528 & 0.2648233 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.6410240 & 0.2864883 & 0.0724875 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.9167351 & 0.0832648 \end{bmatrix} \tag{27}$$

The metric for Improved Long Short is

$$m_{\mathsf{LS1}} = -1.95. \tag{28}$$

In summary, the 8-world state cases had the following results:

	MVO	Long Short	Improved Long Short
m	-2.05	-2.20	-1.95

Discussion

Here, we discuss the implications of our results, and how they apply to the real world.

One of the most elegant results of our project is that adopting our approach allows a trader to determine how robust a strategy is in the presence of changing world states and conditions. We found, for example, that momentum strategies tend to be fairly robust to changing world conditions. The "short down" strategy we implemented was similar to a momentum strategy because we go long when the market is doing well and short when the market is doing poorly. This strategy performed the best out of all of our other strategies because it had the highest Sharpe ratio cutoff, which is the metric we used.

Since we always train the scheduler with the same algorithm, our model also provides an excellent way to compare different strategies. For example, if we knew there was going to be high volatility or rapidly changing market conditions in the future, for example during a presidential election, we should choose a trading strategy that will perform best in such market conditions. Using an optimal scheduler exposes trading strategies' weaknesses under changing market conditions, and therefore provides insight into what needs to change in the algorithm, eventually leading to a better algorithm. A practitioner could look at the specific Q-values of the trained optimal scheduler and observe how the scheduler was exploiting the trading strategy. With this new information, the practitioner could easily construct a trading strategy that fixes the exploit. In this way, our approach can be used to quickly gain feedback about a trading strategy, including its most significant potential risks.

Application to Real Data

Our adversarial scheduler tries to come up with the worst possible set of world state transitions possible. Although such a scheduler will give us a lower bound on trading performance, in the real world, world state transitions are generally not too evil, and therefore, some strategies that were ruled out to be bad against our optimal adversarial scheduler turn out to be decent strategies in real life. For example, even though the adversarial scheduler easily beat the buy and hold strategy, people who buy and hold stocks generally do well in the market in the long run. Our back-testing resulted in the buy and hold strategy being one of the most successful, with a Sharpe ratio of approximately 0.75.

Conclusion

By building a model for simulating, interacting with and verifying properties about financial markets, our project relates to the modeling and verification techniques we learned in the course. We demonstrate an application of $\mathsf{S}\,\mathsf{d}\mathcal{L}$ to a complex real-world problem. We also combined CPS-style reasoning with a variety of techniques from different fields, such as machine learning and finance, to formally describe the problem, verify models, and explore the local behavior of stochastic processes. In addition, despite the overhead in formalizing our problem, we show how effective the results of verifying a formal model can be compared to existing methods used for managing risk and modeling financial phenomenon.

Finally, our final product shows a real-world implementation of a trading algorithm, which demonstrates the practicality and real-world applications of our project. The ultimate goal of CPS is to create controllers that can be used to perform tasks in the world that are guaranteed to be safe. Our project does exactly that, by bringing safety to the inherently risky task of using a fixed strategy to trade billions of dollars in a constantly changing world.

Future Work

Despite the results of our GBM simulator, generating truly random data is very difficult in practice. One way to improve our GBM simulator is using Brownian bridges to simulate the Brownian motions, rather than generating the data sequentially in time. Additional work should also focus on creating a generative adversarial network style of model for trading. In particular, it would be interesting to have the scheduler and the trader be playing a zero-sum game where each one is trying to get better and better, and in the process the trader ends up becoming a good trading strategy. This is similar to generative adversarial networks, in which two neural networks are interlocked in a two player zero-sum game.

References

[1] Ermogenous Angeliki. Brownian motion and its applications in the stock market. `http://ecommons.udayton.edu/mth_epumd/15`, 2006. Accessed: 2018-12-6.

[2] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[3] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307 – 327, 1986.

[4] Gavin Cassar and Joseph J. Gerakos. Do risk management practices work? evidence from hedge funds. `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1722250`, Dec 2010.

[5] Joe Chang. Brownian motion conditional distributions. `http://disi.unal.edu.co/~gjhernandezp/mathcomm/slides/bm.pdf`. Accessed: 2018-11-18.

[6] D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for markov decision processes. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 84–93, Sept 2012.

[7] D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for markov decision processes. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 84–93, Sept 2012.

[8] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker. In Tony Field, Peter G. Harrison, Jeremy Bradley, and Uli Harder, editors, *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 200–204, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[9] Andrew W. Lo. Risk management for hedge funds: Introduction and overview. `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=283308`, Sep 2001.

[10] Grant Olney Passmore and Denis Ignatovich. Formal verification of financial algorithms. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 26–41, Cham, 2017. Springer International Publishing.

[11] Simon Peyton Jones, Jean-Marc Eber, and Julian Seward. Composing contracts: An adventure in financial engineering (functional pearl). *SIGPLAN Not.*, 35(9):280–292, September 2000.

[12] André Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, pages 446–460, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[13] Dr Reddy and V Clinton. Simulating stock prices using geometric brownian motion: Evidence from australian companies. 10:23–47, 01 2016.

[14] Karl Sigman. Simulating normal (gaussian) rvs with applications to simulating brownian motion and geometric brownian motion in one and two dimensions. `http://www.columbia.edu/~ks20/4703-Sigman/4703-07-Notes-BM-GBM-I.pdf`. Accessed: 2018-11-18.