Parallelization Geometric Brownian Motion(GBM) in predicting Gold Spot Prices

Amir Ghaderi and Mezbah Uddin

December 28, 2016

1 Background

The Efficient-Market Hypothesis states that stocks prices are reflections of all of the current information available about a certain company/commodity. This would imply that past historical information i.e. price history couldn't be used to accurately predict future changes in stock prices. However, in recent years there has been a rise of an industry that proposes that some analytical methods predict stock prices better than others. Under this assumption we are applying the Monte Carlo method in an attempt to accurately predict future gold stock prices. The Monte Carlo Method is the idea of repeating an algorithm a certain number of times in order to produce a distribution of outcomes that can analyzed. Essentially, the algorithm uses randomness in an attempt to mimic a deterministic system. The model that we have selected for our Monte Carlo Simulation is the Geometric Brownian Motion (GBM). The GBM is one of the most common financial models currently being used to predict stock prices. The model assumes that stock prices are affected by some constant upwards drift and by a daily random shock. The daily random shock is a product of the markets volatility and a randomly distributed variable. One of the limitations of the Monte Carlo Method is the amount of time it takes to compute all of the iterations. Often times a small number of iterations are used in order to be able to produce a solution in a reasonable amount of time. The goal of this project is to implement a parallelized version of this algorithm, which will allow us to compute a high number of iterations in a reasonable amount of time. It is expected that a greater number of iterations would result in a more accurate final solution.

Current trends in data analytics are calling for more advances in parallel processing of commonly used algorithms. What we are doing is unique to the scientific community as we are the first to compute the Monte Carlo Method with a GBM model in parallel in order to predict future gold prices.

2 Methodology

2.1 Brownian Motion

A geometric Brownian motion (GBM) (also known as exponential Brownian motion) is a continuous-time stochastic process in which the logarithm of the randomly varying quantity follows a Brownian motion (also called a Wiener process) with drift.[1]. Stochastic process S_t is said to follow a GBM if it satisfies the following stochastic differential equation (SDE): $dS_t = \mu S_t dt + \sigma S_t dW_t$ where W_t is a brownian motion, μ and σ are drift and volatility constants. W_t is a weiner process that follows a standard normal distribution with mean 0 and standard deviation 1. With the normality the equation becomes $dS_t = S_t + \mu$ dt $+ S_t\ \sigma\ z_t\sqrt{dt}$. With some more algebric manipulation and taking logarithmic change as overtime change in the price of a security. The equation simplifies to $\ln(S_t + 1/S_t) = \mu + z_t\ \sigma$.

2.2 Model set-up and Data

Research uses daily gold data (in US dollars) from 2016 January 4th to December 20th 2016 for μ and σ calcualtions. Linear model is been fitted to calculate the daily drift(μ and standard deviation of logarithmic return as σ. Furthermore, to get the mean drift the following transformation has been applied: $\mu - 0.5 * \sigma^2$. Using the linear fit values with transformation, the equation used to predict gold prices comes down to $ln((S_t + 1)/S_t) = 0.00015 + 0.01037\ *\ z$. For the purpose of this paper, the model expected to predict 15 day gold spot prices.

2.3 Non-Parallel code

In applying the above GBM equation to predict, the stochastic factor is z, which takes a random value with a mean 0 and standard deviation 1. For each iteration, z is going to generate 15 values and plug in to the gbm equation to get the predictions. And after all the iterations, each day prediction would be an average of the values generated by all iterations.

2.4 Parallelization

For parallelization, each processor core handles equal proportion of total iterations and runs it parallely. For this pupose, the paper uses R's 'for each' and 'doparallel' packages. Below is the pseudo code:

For each processor core(Do Parallel):
  For i=1 to (total iterations/3)[2]:
    generate z
    plug z into gbm and get predictions
    Save each iteration result to DataFrame
Average all the predictions from iterations to get the final series

[1] Ross, Sheldon M. (2014). "Variations on Brownian Motion". Introduction to Probability Models (11th ed.). Amsterdam: Elsevier. pp. 612–14. ISBN 978-0-12-407948-9

[2] Used 3 cores out of 4 for parallelization.

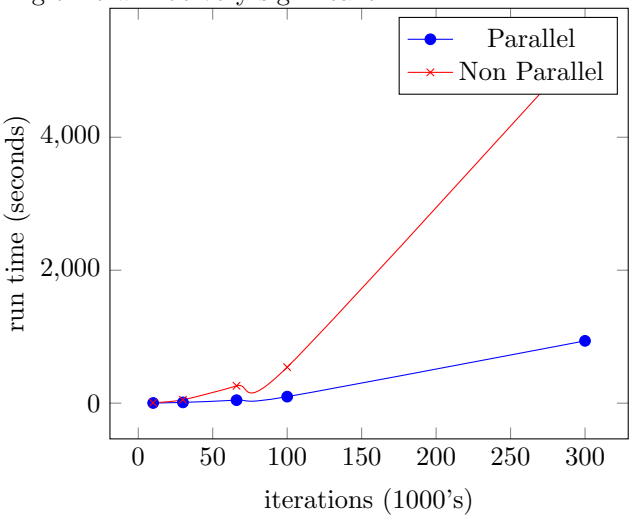
Table 1: Running time

iterations	**Parallelization Time**	**Non - Parallel Time**
10k	2.45 sec	6.53 sec
30k	11.35 sec	51.27 sec
66k	45 sec	261 sec
100k	98.4 sec	541.2 sec
300k	937	5400 sec

3 Results

Different iterations were chosen to see the speed comparison with the above parallelization code. Table 1 above gives the summary of run time for each iterations.

Below graph depicts a visual representation of the time difference. It can be seen that, parallel code runs much faster and the noticable difference starts to occur after 30,000 iterations. Also, run time for non parallel code behaves much like an exponential function, whereas parallel behaves similar to log function. That indicates that, with very high number of iterations, the difference in running time will be very significant.



4 Summary

In conclusion our goal was to implement a parallelized version of a Monte Carlo Method using a Geometric Brownian Motion Model in order to accurately and efficiently predict future gold prices. Overall our implementation was able to significantly outperform the serialized version of the algorithm in terms of processing time. This implies that out parallelized version of the Monte Carlo Method is able to compute iteration at a faster rate and therefore able to produce more accurate results.

References

[1] Jeffrey M. Woolridge, *Introductory Econometrics: A Modern Approach, 5th Edition.*

[2] http://investexcel.net/geometric-brownian-motion-excel/

```
#Install and load packages

install.packages("Quandl")
install.packages('parallel')
install.packages('foreach')
install.packages('doParallel')

library(foreach)
library(doParallel)
library(parallel)
library(Quandl)

#Load Data
gold <- Quandl("LBMA/GOLD")

#Visualize the data
head(gold)
nrow(gold)
class(gold)

#Shaping the data
gold$"EURO (PM)" <- NULL
gold$"EURO (AM)" <- NULL
gold$"GBP (PM)" <- NULL
gold$"GBP (AM)" <- NULL
gold$"USD (AM)" <- NULL

#Creating a subset of the data
gold2 <- gold[1:248,]

#Calculating daily drift
line <- lm(log(gold2$`USD (PM)`) ~ gold2$Date, data=gold2)
drift_daily <- 0.0002069

#Creating log returns column
log_returns <- c()

for (i in 2:length(gold2$`USD (PM)`)){
  x <- gold2$`USD (PM)`[i]/gold2$`USD (PM)`[i-1]
  y<- log(x)
  log_returns <- c(log_returns,y)
}

gold2$log_return <- c("NA",log_returns)
gold2$log_return <- as.numeric(gold2$log_return)
```

```
#Calculating daily volatility
vol_daily <- sd(gold2$log_return, na.rm=TRUE)

#Calculating drift mean
drift_mean <- drift_daily-0.5*(vol_daily^2)

#Calculating predicted log returns for the next 15 days
z <- rnorm(15,mean=0,sd=1)
log_ret <- drift_mean+z*vol_daily

#Non-Parallelized algorithm
a <- Sys.time()
table <- data.frame(row.names=1:15)

for (i in 1:300000){
  z <- rnorm(15,mean=0,sd=1)
  log_ret <- drift_mean+z*vol_daily
  price_pred <- c()
  price_i <- 1125.70
  for (j in 1:15){
    price_i = price_i*exp(log_ret[j])
    price_pred <- c(price_pred,price_i)
  }
  table <- cbind(table,data.frame(price_pred))
}

table$average <- apply(table,1,mean)

b <- Sys.time()
b-a

plot(table$average)
```

```
#Parallelized algorithm

no_cores <- detectCores() - 1
cl<-makeCluster(no_cores)
registerDoParallel(cl)

table <- data.frame(row.names=1:15)

result <- foreach(i = 1:3) %dopar% {
  for (i in 1:3000){
    z <- rnorm(15,mean=0,sd=1)
    log_ret <- drift_mean+z*vol_daily
    price_pred <- c()
    price_i <- 1185.35
    for (j in 1:15){
      price_i = price_i*exp(log_ret[j])
      price_pred <- c(price_pred,price_i)
    }
    table <- cbind(table,data.frame(price_pred))
  }
  table$average <- apply(table,1,mean)
}


table2 <- data.frame(row.names=1:15)

table2 <- cbind(table2,data.frame(result[1]))
table2 <- cbind(table2,data.frame(result[2]))
table2 <- cbind(table2,data.frame(result[3]))

table2$final <- apply(table2,1,mean)

plot(table2$final)

stopCluster(cl)
```