Homework 1

PSTAT 222C

Alex Bernstein

April 29, 2023

Problem 1: Simulating the Heston Model

(a) The Milstein Scheme for the Heston Model

Proof. Using the notation from the book, we let:

$$dV_t = \kappa(\theta - V_t)dt + \eta\sqrt{V_t}dW_t$$
$$L^1(x) = \eta\sqrt{x}\frac{\partial}{\partial x}$$

We therefore have that the Milstein Scheme for V_t as given is:

$$\begin{aligned} V_{n+1} &= V_n + \kappa(\theta - V_n)\Delta t + \eta\sqrt{V_n}\sqrt{t}Z + L^1(\eta\sqrt{(V_n)})\frac{\Delta t Z^2 - \Delta t}{2} \\ &= V_n + \kappa(\theta - V_n)\Delta t + \eta\sqrt{V_n}\sqrt{t}Z + \frac{\eta^2}{2}\left(\frac{\Delta t Z^2 - \Delta t}{2}\right) \end{aligned}$$

□

(b) Implement the following

Schema:

- Euler-Maruyama for both S_t and V_t
- Euler-Maruyama for S_t and Milstein for V_t

for the following contracts:

- A Put Option with payoff $\Phi(S_T) = e^{-rT}(K - S_T)_+$
- A discretely monitored Asian Option with payoff

$$\Phi(S_{[0,T]}) = (\frac{1}{N_T}\sum_{n=1}^{N_T} S_{T_n} - S_T)_+$$

Use parameters $r = 0.05$; $\kappa = 1$; $\theta = 0.2$; $\eta = 0.5$; $\rho = -0.4$ with initial conditions $S_0 = 100$ and $V_0 = 0.25$ and option parameters $T = 1, N_T = 52, T = 1$.

Use $\Delta t = \frac{1}{52 \cdot 2^r}$ for $r = 1, 2, 3, 4$ and $M = 10^5$ Monte Carlo simulations to estimate the price of the above two contracts. Report: 1. 95% Confidence Interval 2. 99% Confidence Interval 3. Running Time of your scheme.

```r
# Parameters
T <- 1                    # Time horizon
K <- 100                  # Number of paths
N_T <- 52                 # Number of time steps
M <- 10 ^ 5
r_vec <- c(1, 2, 3, 4)
M <- 10000

put_euler <- c()
asian_euler <- c()
running_time <- c()
sd_put <- c()
sd_asian <- c()

# Heston model parameters
S0 <- 100
V0 <- 0.25
r <- 0.05
kappa <- 1
theta <- 0.2
eta <- 0.5
rho <- -0.4

# Random seed for reproducibility
set.seed(123)

for (r_val in r_vec) {
  writeLines(" \n ")
  writeLines(paste0("l = ", r_val))
  dt <- 1 / (52 * 2 ^ r_val)            # Time step size
  periods <- 52 * 2 ^ r_val
  time_sim <- Sys.time()           # Starting Time
  # Initialize matrices for S_t and V_t
  S <- matrix(0, nrow = M, ncol = periods + 1)
  V <- matrix(0, nrow = M, ncol = periods + 1)
  S[, 1] <- S0
  V[, 1] <- V0
  asian <- c()
  pb = txtProgressBar(min = 0, M, initial = 0)
  for (m in 1:M) {
    sum <- 0
    for (i in 1:periods) {
      # Generate correlated random variables
      Z1 <- rnorm(n = 1,
                  mean = 0,
                  sd = sqrt(dt))
      Z2 <-
        rho * Z1 + sqrt(1 - rho ^ 2) * rnorm(n = 1,
                                             mean = 0,
                                             sd = sqrt(dt))

      # Update V_t (Euler scheme)
      V[m, i + 1] <-
```

```r
        V[m, i] + kappa * (theta - V[m, i]) * dt + eta * sqrt(V[m, i]) * Z2
      V[m, i + 1] <-
        max(V[m, i + 1], 0)  # Ensure positivity of V[k, i + 1]

      # Update S_t (Euler scheme)
      S[m, i + 1] <-
        S[m, i] + r * S[m, i] * dt + S[m, i] * sqrt(V[m, i]) * Z1
      S[m, i + 1] <-
        max(S[m, i + 1], 0)  # Ensure positivity of S[k, i + 1]
      if (i %% 2 ^ r_val == 0) {
        sum <- sum + S[m, i + 1]
      }
    }
    asian <- c(asian, max(sum / N_T - S[m, i + 1], 0))
    setTxtProgressBar(pb, m)
  }
  # Put payoff
  put_euler <-
    c(put_euler, mean(exp(-r * T) * pmax(K - S[, periods + 1], 0)))
  asian_euler <- c(asian_euler, mean(exp(-r * T) * asian))
  end_time <- Sys.time()
  running_time <- c(running_time, end_time - time_sim)
  sd_put <-
    c(sd_put, sd(exp(-r * T) * pmax(K - S[, periods + 1], 0)) / sqrt(M))
  sd_asian <- c(sd_asian, sd(exp(-r * T) * asian) / sqrt(M))
}
running_time <- rep(running_time, 2)
put_ci_95 <-
  t(matrix(
    c(
      put_euler - qnorm(0.975) * sd_put,
      put_euler + qnorm(0.975) * sd_put
    ),
    nrow = 2,
    byrow = T
  ))
put_ci_99 <-
  t(matrix(
    c(
      put_euler - qnorm(0.995) * sd_put,
      put_euler + qnorm(0.995) * sd_put
    ),
    nrow = 2,
    byrow = T
  ))
asian_ci_95 <-
  t(matrix(
    c(
      asian_euler - qnorm(0.975) * sd_put,
      asian_euler + qnorm(0.975) * sd_put
    ),
    nrow = 2,
    byrow = T
```

```r
  ))
asian_ci_99 <-
  t(matrix(
    c(
      asian_euler - qnorm(0.995) * sd_put,
      asian_euler + qnorm(0.995) * sd_put
    ),
    nrow = 2,
    byrow = T
  ))

data_out_euler <-
  cbind(rbind(put_ci_95, asian_ci_95), rbind(put_ci_99, asian_ci_99))
prices_euler <- c(put_euler, asian_euler)
data_out_euler <- cbind(prices_euler, data_out_euler, running_time)
colname_set <-
  c("prices",
    "95% lower CI",
    "95% upper CI",
    "99% lower CI",
    "99% upper CI",
    "Running Time")
rowname_set <-
  c(
    "EuroPut, r=1",
    "EuroPut, r=2",
    "EuroPut, r=3",
    "EuroPut, r=4",
    "Asian, r=1",
    "Asian, r=2",
    "Asian, r=3",
    "Asian, r=4"
  )
colnames(data_out_euler) <- colname_set
rownames(data_out_euler) <- rowname_set
```

```r
stargazer(data_out_euler,type='latex',title="Euler-Maruyama Scheme")
```

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac at gmail.com % Date and time: Sat, Apr 29, 2023 - 16:19:29

Table 1: Euler-Maruyama Scheme

	prices	95% lower CI	95% upper CI	99% lower CI	99% upper CI	Running Time
EuroPut, r=1	15.811	15.406	16.216	15.279	16.344	3.965
EuroPut, r=2	15.849	15.439	16.258	15.311	16.386	7.628
EuroPut, r=3	15.503	15.101	15.905	14.975	16.031	14.929
EuroPut, r=4	15.412	15.009	15.816	14.882	15.942	29.612
Asian, r=1	8.838	8.433	9.243	8.305	9.371	3.965
Asian, r=2	8.800	8.391	9.210	8.263	9.338	7.628
Asian, r=3	8.768	8.366	9.170	8.240	9.296	14.929
Asian, r=4	8.516	8.113	8.919	7.986	9.046	29.612

Note that the the European Put and Asian options with corresponding r values have the same runtime because both used the same sample paths of S. This is true of the following Euler-Milstein scheme as well.

```
# Parameters
T <- 1                    # Time horizon
K <- 100                  # Number of paths
N_T <- 52                 # Number of time steps
M <- 10 ^ 5
r_vec <- c(1, 2, 3, 4)
M <- 10000

put_milstein <- c()
asian_milstein <- c()
running_time <- c()
sd_put <- c()
sd_asian <- c()

# Heston model parameters
S0 <- 100
V0 <- 0.25
r <- 0.05
kappa <- 1
theta <- 0.2
eta <- 0.5
rho <- -0.4

# Random seed for reproducibility
set.seed(123)

for (r_val in r_vec) {
  writeLines(" \n ")
  writeLines(paste0("l = ", r_val))
  dt <- 1 / (52 * 2 ^ r_val)             # Time step size
  periods <- 52 * 2 ^ r_val
  time_sim <- Sys.time()           # Starting Time
  # Initialize matrices for S_t and V_t
  S <- matrix(0, nrow = M, ncol = periods + 1)
  V <- matrix(0, nrow = M, ncol = periods + 1)
  S[, 1] <- S0
  V[, 1] <- V0
  asian <- c()
  pb = txtProgressBar(min = 0, M, initial = 0)
  for (m in 1:M) {
    sum <- 0
    for (i in 1:periods) {
      # Generate correlated random variables
      Z1 <- rnorm(n = 1,
                  mean = 0,
                  sd = sqrt(dt))
      Z2 <-
        rho * Z1 + sqrt(1 - rho ^ 2) * rnorm(n = 1,
                                             mean = 0,
                                             sd = sqrt(dt))
```

```
      # Update V_t (Milstein scheme)
      V[m, i + 1] <-
        V[m, i] + kappa * (theta - V[m, i]) * dt + eta * sqrt(V[m, i]) * Z2 +
        eta ^ 2 / 4 * (Z2 ^ 2 - dt)
      V[m, i + 1] <-
        max(V[m, i + 1], 0)  # Ensure positivity of V[k, i + 1]

      # Update S_t (Euler scheme)
      S[m, i + 1] <-
        S[m, i] + r * S[m, i] * dt + S[m, i] * sqrt(V[m, i]) * Z1
      S[m, i + 1] <-
        max(S[m, i + 1], 0)  # Ensure positivity of S[k, i + 1]
      if (i %% 2 ^ r_val == 0) {
        sum <- sum + S[m, i + 1]
      }
    }
    asian <- c(asian, max(sum / N_T - S[m, i + 1], 0))
    setTxtProgressBar(pb, m)
  }
  # Put payoff
  put_milstein <-
    c(put_milstein, mean(exp(-r * T) * pmax(K - S[, periods + 1], 0)))
  asian_milstein <- c(asian_milstein, mean(exp(-r * T) * asian))
  end_time <- Sys.time()
  running_time <- c(running_time, end_time - time_sim)
  sd_put <-
    c(sd_put, sd(exp(-r * T) * pmax(K - S[, periods + 1], 0)) / sqrt(M))
  sd_asian <- c(sd_asian, sd(exp(-r * T) * asian) / sqrt(M))
}
running_time <- rep(running_time, 2)
put_ci_95 <-
  t(matrix(
    c(
      put_milstein - qnorm(0.975) * sd_put,
      put_milstein + qnorm(0.975) * sd_put
    ),
    nrow = 2,
    byrow = T
  ))
put_ci_99 <-
  t(matrix(
    c(
      put_milstein - qnorm(0.995) * sd_put,
      put_milstein + qnorm(0.995) * sd_put
    ),
    nrow = 2,
    byrow = T
  ))
asian_ci_95 <-
  t(matrix(
    c(
      asian_milstein - qnorm(0.975) * sd_put,
      asian_milstein + qnorm(0.975) * sd_put
```

```
    ),
    nrow = 2,
    byrow = T
  ))
asian_ci_99 <-
  t(matrix(
    c(
      asian_milstein - qnorm(0.995) * sd_put,
      asian_milstein + qnorm(0.995) * sd_put
    ),
    nrow = 2,
    byrow = T
  ))
data_out_milstein <-
  cbind(rbind(put_ci_95, asian_ci_95),
        rbind(put_ci_99, asian_ci_99),
        running_time)
prices_milstein <- c(put_milstein, asian_milstein)
data_out_milstein <- cbind(prices_milstein, data_out_milstein)
colnames(data_out_milstein) <- colname_set
rownames(data_out_milstein) <- rowname_set
```

```
stargazer(data_out_milstein,type='latex',title="Euler-Milstein Scheme")
```

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac at gmail.com % Date and time: Sat, Apr 29, 2023 - 16:19:29

Table 2: Euler-Milstein Scheme

	prices	95% lower CI	95% upper CI	99% lower CI	99% upper CI	Running Time
EuroPut, r=1	15.808	15.403	16.213	15.275	16.341	3.855
EuroPut, r=2	15.848	15.439	16.257	15.310	16.386	7.586
EuroPut, r=3	15.502	15.100	15.904	14.974	16.031	15.184
EuroPut, r=4	15.414	15.011	15.817	14.884	15.944	29.886
Asian, r=1	8.838	8.433	9.243	8.306	9.371	3.855
Asian, r=2	8.801	8.392	9.210	8.263	9.339	7.586
Asian, r=3	8.768	8.366	9.170	8.240	9.296	15.184
Asian, r=4	8.517	8.114	8.921	7.987	9.047	29.886

We find that the Milstein scheme and the Euler-Maruyama scheme have very comparable variances and runtimes. Both Euler-Maruyama and Milstein schemes get more precise as $ dt$ goes to 0, as seen by the narrowing of the confidence interval (although this does not necessarily account for bias in the estimator). Furthermore, both narrow in a similar manners, suggesting that in the case of the Heston Model, sampling according to the Milstein scheme does not appreciably improve the result compared to Euler-Maruyama.

Problem 2: Richardson-Romberg Extrapolation

Use the CEV model

$$dX_t = rX_t dt_\sigma X_t^\gamma dW_t$$

for a corridor option that has payoff:

$$\Phi(X_T) = \mathbf{1}_{\{L_1 < X_t < L_2\}} \; t \leq T$$

with parameters $\gamma = .8, X_0 = 20, \sigma = 00.4, r = 0.05, T = \frac{1}{2}, L_1 = 15$ and $L_2 = 25$ Implement: a. Euler scheme for X_t with $h = 0.01$ and estimate via Monte Carlo for discrete times t_k with $M = 10^5$ the 95% confidence interval for the price of the corridor option.

b. Richardson Romberg extrapolation for $h = 0.01$. How much variance reduction is obtained? duplicate for $h = 0.005$ and $h = 0.0025$.

c. Antithetic sampling for a plain Euler scheme. How much variance reduction is obtained?

(a) Euler Monte Carlo

```r
# parameters
h <-  0.01
gam <- 0.8
s <- 0.4
r <- 0.05
T <- 0.5
periods <- T / h
x0 = 20
set.seed(123)
M <- 10 ^ 5
L1 <- 15
L2 <- 25
X <- matrix(0, nrow <- M, ncol <- periods + 1)
X[, 1] <- x0
payoffs <- rep(1, M)
start_time<-Sys.time()
for (m in 1:M) {
  for (i in 1:periods) {
    X[m, i + 1] <- X[m, i] + r * X[m, i] * h + s * X[m, i] ^ gam * sqrt(h) *
      rnorm(1)
  }
  if (max(X[m, ] > L2) || min(X[m, ]) < L1) {
    payoffs[m] <- 0
  }
}
end_time<-Sys.time()

em_runtime<-end_time-start_time
price_em <- exp(-r * T) * mean(payoffs)
se_em_corridor <- sd(exp(-r * T) * (payoffs)) / sqrt(M)
se_em_ci <-
  c(price_em - se_em_corridor * qnorm(.975),
    price_em + se_em_corridor * qnorm(.975))
em_results <- c(price_em, se_em_corridor, se_em_ci,em_runtime)
```

We find that with a basic Euler-Maruyama sample, we have an mean price of 0.7865192 and a confidence interval of 0.7841308, 0.7889075.

(b) Richardson-Romberg Extrapolation

```
# parameters
h_vec <-  c(0.01, 0.005, 0.0025)
gam <- 0.8
s <- 0.4
r <- 0.05
T <- 0.5

x0 = 20
set.seed(123)
M <- 10 ^ 5
L1 <- 15
L2 <- 25


price_rr <- c()
se_rr_corridor <- c()
rr_runtime<-c()

for (h in h_vec) {
  periods <- T / h
  X_coarse <- matrix(0, nrow <- M, ncol <- periods + 1)
  payoff_coarse <- rep(1, M)
  X_coarse[, 1] <- x0
  X_fine <-  matrix(0, nrow <- M, ncol <- 2 * periods + 1)
  payoff_fine <- rep(1, M)
  X_fine[, 1] <- x0
  start_time<-Sys.time()
  for (m in 1:M) {
    dW_fine <- rnorm(2 * periods)
    dW_coarse <-
      (dW_fine[1:length(dW_fine) %% 2 == 1] + dW_fine[1:length(dW_fine) %% 2 == 0]) /
      sqrt(2)

    for (i in 1:(2 * periods)) {
      X_fine[m, i + 1] <-
        X_fine[m, i] + r * X_fine[m, i] * (h / 2) + s * X_fine[m, i] ^ gam * sqrt(h /
                                                                                  2) * dW_fine[i]
      if (i %% 2 == 0) {
        X_coarse[m, i / 2 + 1] <-
          X_coarse[m, i / 2] + r * X_coarse[m, i / 2] * h + s * X_coarse[m, i / 2] ^
          gam * sqrt(h) * dW_coarse[i / 2]
      }
    }
    if (max(X_fine[m,]) > L2 || min(X_fine[m,]) < L1) {
      payoff_fine[m] <- 0
    }
    if (max(X_coarse[m,]) > L2 || min(X_coarse[m,]) < L1) {
      payoff_coarse[m] <- 0
    }
```

```
  }
  payoffs <- 2 * payoff_fine - payoff_coarse
  price_rr <- c(price_rr, exp(-r * T) * mean(payoffs))
  se_rr_corridor <-
    c(se_rr_corridor, sd(exp(-r * T) * (payoffs)) / sqrt(M))
  end_time<-Sys.time()
  rr_runtime<-c(rr_runtime,end_time-start_time)

}

se_rr_ci <-
  cbind(price_rr - qnorm(.975) * se_rr_corridor,
        price_rr + qnorm(.975) * se_rr_corridor)
rr_results <- cbind(price_rr, se_rr_corridor, se_rr_ci,rr_runtime)
colnames(rr_results) <-
  c("mean price", 'standard error', "lower 95%", "upper 95%","runtime")
RR_rows <-
  c(paste("RR, h=", h_vec[1]),
    paste("RR, h=", h_vec[2]),
    paste("RR, h=", h_vec[3]))
rownames(rr_results) <- RR_rows
```

We find (as summarized in the table after the next section) that the Richardson-Romberg actually *slightly increases* variance compared to Euler-Maruyama.

(c) Antithetic Sampling

```
# parameters
h <-  0.01
gam <- 0.8
s <- 0.4
r <- 0.05
T <- 0.5
periods <- T/h
x0 = 20
set.seed(123)
M<-10^5
L1 <- 15
L2 <- 25
X_1 <- matrix(0, nrow <- M, ncol <- periods + 1)
X_1[,1]<- x0
X_2 <- matrix(0, nrow <- M, ncol <- periods + 1)
X_2[,1]<- x0
payoffs_1<-rep(1,M)
payoffs_2<-rep(1,M)
start_time<-Sys.time()
for(m in 1:M){
  for(i in 1:periods){
    dW<-rnorm(1)
    X_1[m,i+1] <- X_1[m,i] + r*X_1[m,i]*h + s*X_1[m,i]^gam*sqrt(h)*dW
    X_2[m,i+1] <- X_2[m,i] + r*X_2[m,i]*h + s*X_2[m,i]^gam*sqrt(h)*(-dW)
  }
  if(max(X_1[m,]>L2) || min(X_1[m,])<L1){
```

```
      payoffs_1[m]<-0
  }
  if(max(X_2[m,]>L2) || min(X_2[m,])<L1){
      payoffs_2[m]<-0
  }
}
end_time<-Sys.time()
as_runtime<-end_time-start_time
payoffs<-c(payoffs_1,payoffs_2)
price_as<-exp(-r*T)*mean(payoffs)
se_as_corridor <- sd(exp(-r*T)*(payoffs))/sqrt(2*M)
se_as_ci<-c(price_as-se_as_corridor*qnorm(.975),price_as+se_as_corridor*qnorm(.975))
as_results<-c(price_as,se_as_corridor,se_as_ci,as_runtime)
```

We find that with Antithetic Sampling, we have an mean price of 0.7858121 and a confidence interval of 0.7841209, 0.7875033, a noticable reduction in variance compared to both Richardson-Romberg and Euler-Maruyama.

Combined Results

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac at gmail.com % Date and time: Sat, Apr 29, 2023 - 13:52:51

Table 3:

	mean price	standard error	lower 95%	upper 95%	runtime
Simple EM	0.786519	0.001219	0.784131	0.788908	7.41975
RR, h= 0.01	0.768895	0.001333	0.766282	0.771509	4.197478
RR, h= 0.005	0.766486	0.001321	0.763898	0.769075	7.64564
RR, h= 0.0025	0.762741	0.001311	0.760171	0.765311	14.627825
Antithetic Sampling	0.785812	0.000863	0.784121	0.787503	8.326304

Problem 3: Wealth Maximization by Monte Carlo

(a)

The initial algorithm for maximizing wealth is as follows:

```
# other parameters
r <- 0.05                 # risk-free interest rate
M <- 10000                # number of paths
pi_vec <- seq(0, 1, .1)     # wealth proportion pi
x0 <- 100                  # initial wealth
gam <- -1.5
# GBM Parameters
mu <- 0.1                           # GBM drift
s <- 0.2                            # GBM volatility
lam <- 0                           # Poisson Process Rate
eta <-
  50                             # exponential jump avg rate (exponential distn rate param)
s0 <- 100                           # initial condition for S0
T <- 2                                # Time Window (years)
n_steps <- 52 * 2                     # number of steps
dt <- 1 / 52                          # regular grid
optimal = (1 / (1 - gam)) * (mu - r) / s ^ 2


set.seed(51234)

expected_wealth <- rep(0, length(pi_vec))
expected_utility <- rep(0, length(pi_vec))
pi_id <- 0
for (pi_prop in pi_vec) {
  pi_id <- pi_id + 1
  set.seed(12345)
  grid_times_orig <- seq(from = 0, to = T, by = dt)
  sum <- 0
  util_sum <- 0
  for (m in 1:M) {
    n_jumps <- rpois(1, lam * T)
    jump_times <- runif(n_jumps, 0, T)
    grid_times <- sort(c(jump_times, grid_times_orig))
    S <- rep(0, length(grid_times))
    X <- rep(0, length(grid_times))
    jump_indices <- match(jump_times, grid_times)
    S[1] <- s0
    X[1] <- x0
    for (i in 2:length(grid_times)) {
      dt_mod = grid_times[i] - grid_times[i - 1]
      S[i] <-
        max(S[i - 1] + mu * S[i - 1] * dt_mod + s * S[i - 1]  * rnorm(1, sd = sqrt(dt_mod)), 0)
      if (i %in% jump_indices) {
        S[i] <- S[i] * (2 - exp(rexp(n = 1, rate = eta)))
      }
      dS_s <- (S[i] - S[i - 1]) / S[i]
      X[i] <-
        X[i - 1] + pi_prop * X[i - 1] * dS_s + (1 - pi_prop) * X[i - 1] * r * dt_mod
```

```r
    }
    sum <- sum + X[length(grid_times)]
    util_sum_old <- util_sum
    util_sum <- util_sum + (X[length(grid_times)] ^ gam / gam)
    if (is.na(util_sum)) {
      browser()
    }
  }
  expected_wealth[pi_id] <- sum / M
  expected_utility[pi_id] <- util_sum / M

}
plot(pi_vec, expected_utility, type = "l", col = "red")
abline(v = optimal, col = 'blue', )
```

(b) Maximization over π

Empirically, we construct a mesh of 0.05-increment steps between 0 and 1. The algorithm was slightly modified so that paths are identical for each value of h used.

```
# other parameters
r <- 0.05               # risk-free interest rate
M <- 10 ^ 4              # number of paths
pi_vec <- seq(0, 1, .05)    # wealth proportion pi
x0 <- 100                # initial wealth
gam <- -1.5
# GBM Parameters
mu <- 0.1                         # GBM drift
s <- 0.2                          # GBM volatility
lam <- 12                         # Poisson Process Rate
eta <-
  50                          # exponential jump avg rate (exponential distn rate param)
s0 <- 100                         # initial condition for S0
T <- 2                              # Time Window (years)
n_steps <- 52 * 2                   # number of steps
dt <- 1 / 52                        # regular grid
optimal = (1 / (1 - gam)) * (mu - r) / (s ^ 2)
set.seed(123456)
expected_wealth <- rep(0, length(pi_vec))
expected_utility <- rep(0, length(pi_vec))
pi_id <- 0
pb <- txtProgressBar(min = 0, length(pi_vec), initial = 0)
pi_id <- pi_id + 1
set.seed(12345)
grid_times_orig <- seq(from = 0, to = T, by = dt)
sum <- rep(0, length(pi_vec))
util_sum <- rep(0, length(pi_vec))
X <- rep(x0, length(pi_vec))

for (m in 1:M) {
  X <- rep(x0, length(pi_vec))
  n_jumps <- rpois(1, lam * T)
  jump_times <- runif(n_jumps, 0, 2)
  grid_times <- sort(c(jump_times, grid_times_orig))
  S <- rep(0, length(grid_times))
  jump_indices <- match(jump_times, grid_times)
  S[1] <- s0
  for (i in 2:length(grid_times)) {
    dt_mod = grid_times[i] - grid_times[i - 1]
    S[i] <-
      max(S[i - 1] + mu * S[i - 1] * dt_mod + s * S[i - 1]  * rnorm(1, sd = sqrt(dt_mod)), 0)
    if (i %in% jump_indices) {
      S[i] <- S[i] * (2 - exp(rexp(n = 1, rate = eta)))
    }
    dSt_S <- (S[i] - S[i - 1]) / S[i - 1]
    for (pi_id in 1:length(pi_vec)) {
      X_old <- X[pi_id]     #store t-1 value
      X[pi_id] <-
        X_old + pi_vec[pi_id] * X_old * dSt_S + (1 -  pi_vec[pi_id]) * X_old * r * dt_mod
    }
```

```r
  }
  for (pi_id in 1:length(pi_vec)) {
    sum[pi_id] <- sum[pi_id] + X[pi_id]
    util_sum[pi_id] <- util_sum[pi_id] + (X[pi_id] ^ gam / gam)
  }
  setTxtProgressBar(pb, m)
}

for (pi_id in 1:length(pi_vec)) {
  expected_wealth[pi_id] <- sum[pi_id] / M
  expected_utility[pi_id] <- util_sum[pi_id] / M
}

plot(
  pi_vec,
  expected_utility,
  type = "l",
  col = "red",
  main = expression(paste(
    "Expected Utility of Terminal Wealth, ", lambda, "=12"
  )),
  xlab = expression(pi),
  ylab = "Expected Utility"
)
abline(v = optimal, col = 'blue', lty = 3)
legend(
  "bottomleft",
  legend = c("Empirical Avg. Utility", expression(paste(lambda, "=0 Optimal"))),
  lty = c(1, 3)
)
```

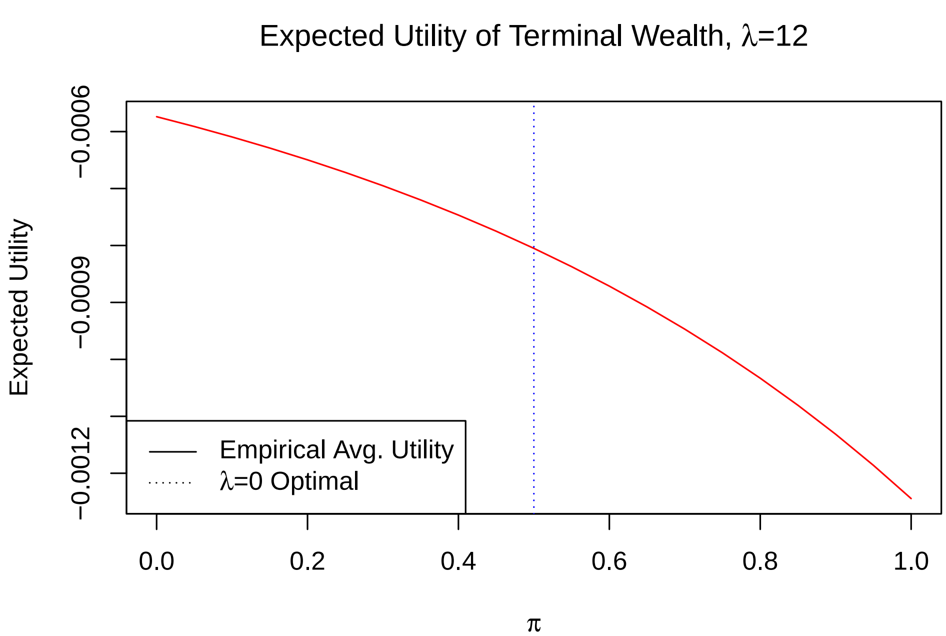

Figure 1: Utility with $\lambda = 12$

Empirically, we find that the best value for π is 0. This seems somewhat counterintuitive to me. When we rerun the same experiment for $\lambda = 0$ (i.e. the classical Merton problem), we find:

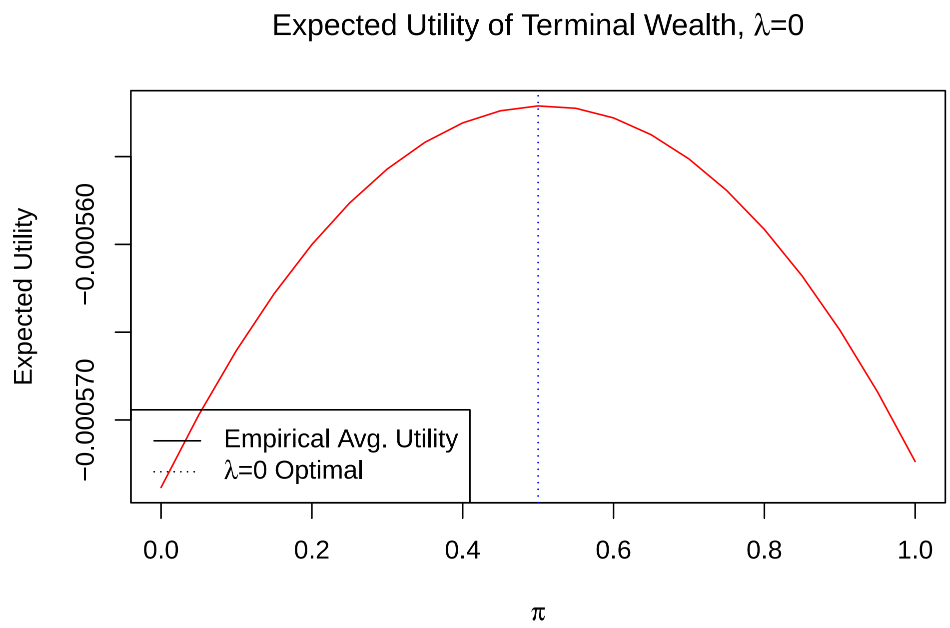

Figure 2: Utility with $\lambda = 0$

This does match theory about the solution to the Merton. This suggests that with these jumps, we would want to choose π as small as possible, i.e. we would ignore the risky asset. Intuitively, this means that jumps downward for the risky asset mean that, if using power utility, it is unwise to invest in this asset.

Citations

Higham, Desmond J. 2021. *Introduction to the Numerical Simulation of Stochastic Differential Equations*. Other Titles in Applied Mathematics. Philadelphia: Society for Industrial; Applied Mathematics.

Hlavac, Marek. 2022. *Stargazer: Well-Formatted Regression and Summary Statistics Tables*. Bratislava, Slovakia: Social Policy Institute. https://CRAN.R-project.org/package=stargazer.