# RCode For Monte Carlo Least Squares

2025-04-17

## (Appendix A) Estimation of Parameters

Estimation of Parameters Notebook Github

## (Appendix B) Binomial Model Code

### Function to price an American put using a Binomial Tree

```
binomial_tree_american_put <- function(S0, K, T, r, sigma, M) {
  dt <- T / M  #Time Step Size
  u <- exp(sigma * sqrt(dt)) # Upturn factor
  d <- exp(-sigma * sqrt(dt)) # Downward factor
  p <- (exp(r * dt) - d) / (u - d) # Risk-neutral probability

  # Stock price tree
  stock_tree <- matrix(0, nrow = M + 1, ncol = M + 1) # Create matrix to store stock prices
  for (j in 0:M) {
    for (i in 0:j) {
      stock_tree[i + 1, j + 1] <- S0 * u^i * d^(j - i)
      # Number of steps multiplied by upturn and downturn factors
    }
  }
  # Option value tree
  option_tree <- matrix(0, nrow = M + 1, ncol = M + 1) # Matrix to store option values
  option_tree[, M + 1] <- pmax(K - stock_tree[, M + 1], 0)
  # Payoff is max(K - S, 0) at maturity, applied to entire column M+1

  # Work backward through the tree
  for (j in (M - 1):0) { # Go through each column in reverse order starting at M-1
    for (i in 0:j) { # Go though each row in the column
      continuation <- exp(-r*dt)*(p*option_tree[i+2,j+2]+(1-p)*option_tree[i+1,j+2])
      exercise <- pmax(K - stock_tree[i + 1, j + 1], 0)
      option_tree[i + 1, j + 1] <- pmax(continuation, exercise)
    }
  }

  return(option_tree[1, 1])  # Option price at root
}
```

# (Appendix C) Monte Carlo Algorithm

### Function to price an American put using Monte Carlo simulations

```r
price_american_put_longstaff_schwartz_MC <- function(K, M, N, r, S0,sigma, polynomial) {

  dt <- 1/M
  discount <- exp(-r * dt)
  set.seed(123)
  Z <- matrix(rnorm(N * M), nrow = N, ncol = M) # Vectorize Brownian Motion Simulation
  S <- S0 * exp(sigma * sqrt(dt) * t(apply(Z, 1, cumsum)))

  Cash_flow <- matrix(0, nrow = N, ncol = M)
  Cash_flow[, M] <- pmax(K - S[, M], 0)

  # Cash Flows at each time step
  for (m in M:2) {
    X <- S[, m-1]
    Y <- Cash_flow[, m] * discount
    XY <- cbind(X, Y)
    XY[X > K, ] <- NA

    if (all(is.na(XY))) {
        Cash_flow[, m-1] <- 0  # Skip regression if no in the money paths
        next
    }
    regression <- lm(polynomial, data = as.data.frame(XY))
    immediate_exercise <- pmax(K - S[, m-1], 0)
    continuation <- predict(regression, newdata = as.data.frame(X))
    Cash_flow[, m-1] <- ifelse(continuation < immediate_exercise, immediate_exercise, 0)
  }

  # Discounting cash flows
  for (i in 1:nrow(Cash_flow)) {
    for (j in 1:ncol(Cash_flow)) {
      if (Cash_flow[i, j] != 0) {
        Cash_flow[i, j] <- Cash_flow[i, j] * round(exp(-r * j), 5)
        if (j < ncol(Cash_flow)) {
          Cash_flow[i, (j+1):ncol(Cash_flow)] <- 0
        }
        break
      }
    }
  }

  option_price <- mean(rowSums(Cash_flow))
  return(option_price)
}
```

# (Appendix D) Effect of Time Steps M

Effect of Time Steps M Notebook Github

# (Appendix E) Monte Carlo Algorithm using Laguerre and Hermite polynomials

## Function to price an American put using Laguerre polynomials

```
price_american_put_Laguerre <- function(K, M, N, r, S0, sigma, degree) {

  dt <- 1/M
  discount <- exp(-r * dt)
  set.seed(123)
  Z <- matrix(rnorm(N * M), nrow = N, ncol = M)
  S <- S0 * exp(sigma * sqrt(dt) * t(apply(Z, 1, cumsum)))

  Cash_flow <- matrix(0, nrow = N, ncol = M)
  Cash_flow[, M] <- pmax(K - S[, M], 0)

  laguerre_basis <- function(x, d) {
    L <- list()
    L[[1]] <- rep(1, length(x))                                 # L0
    if (d >= 1) L[[2]] <- -x + 1                                 # L1
    if (d >= 2) L[[3]] <- 0.5 * (x^2 - 4*x + 2)                  # L2
    if (d >= 3) L[[4]] <- ( -x^3 + 9*x^2 - 18*x + 6 ) / 6        # L3
    if (d >= 4) L[[5]] <- (x^4 - 16*x^3 + 72*x^2 - 96*x + 24) / 24 # L4
    if (d >= 5) L[[6]] <- (-x^5 + 25*x^4 - 200*x^3 + 600*x^2 - 600*x + 120) / 120  # L5
    do.call(cbind, L[1:(d+1)])
  }

  for (m in M:2) {
    X <- S[, m-1]
    Y <- Cash_flow[, m] * discount

    in_the_money <- X < K
    if (sum(in_the_money) == 0) {# Skip regression if no in the money paths
      Cash_flow[, m-1] <- 0
      next
    }

    X_in <- X[in_the_money]
    Y_in <- Y[in_the_money]

    laguerre_features <- laguerre_basis(X_in, degree) # Create matrix with laguerre degree
    regression_df <- data.frame(Y = Y_in, laguerre_features)
    colnames(regression_df) <- c("Y", paste0("L", 0:degree))

    regression_formula <- as.formula(paste("Y ~", paste(colnames(regression_df)[-1], collapse = " + ")))
    regression <- lm(regression_formula, data = regression_df)

    all_features <- laguerre_basis(X, degree)
    colnames(all_features) <- paste0("L", 0:degree)
    continuation <- predict(regression, newdata = as.data.frame(all_features))

    immediate_exercise <- pmax(K - X, 0)
    Cash_flow[, m-1] <- ifelse(continuation < immediate_exercise, immediate_exercise, 0)
```

```r
  }

  # Discounting cash flows
  for (i in 1:N) {
    for (j in 1:M) {
      if (Cash_flow[i, j] != 0) {
        Cash_flow[i, j] <- Cash_flow[i, j] * round(exp(-r * j), 5)
        if (j < M) Cash_flow[i, (j+1):M] <- 0
        break
      }
    }
  }

  return(mean(rowSums(Cash_flow)))
}
```

### Function to price an American put using Hermite Polynomials

```r
price_american_put_Hermite <- function(K, M, N, r, S0, sigma, degree) {

  dt <- 1/M
  discount <- exp(-r * dt)
  set.seed(123)
  Z <- matrix(rnorm(N * M), nrow = N, ncol = M)
  S <- S0 * exp(sigma * sqrt(dt) * t(apply(Z, 1, cumsum)))

  Cash_flow <- matrix(0, nrow = N, ncol = M)
  Cash_flow[, M] <- pmax(K - S[, M], 0)

  hermite_basis <- function(x, d) {
    H <- list()
    H[[1]] <- rep(1, length(x))                                    # H0
    if (d >= 1) H[[2]] <- x                                         # H1
    if (d >= 2) H[[3]] <- x^2 - 1                                   # H2
    if (d >= 3) H[[4]] <- x^3 - 3*x                                 # H3
    if (d >= 4) H[[5]] <- x^4 - 6*x^2 + 3                          # H4
    if (d >= 5) H[[6]] <- x^5 - 10*x^3 + 15*x                      # H5
    do.call(cbind, H[1:(d+1)])
  }

  for (m in M:2) {
    X <- S[, m-1]
    Y <- Cash_flow[, m] * discount

    in_the_money <- X < K
    if (sum(in_the_money) == 0) {
      Cash_flow[, m-1] <- 0
      next
    }

    X_in <- X[in_the_money]
    Y_in <- Y[in_the_money]
```

```
    hermite_features <- hermite_basis(X_in, degree)# Create matrix with Hermite degree
    regression_df <- data.frame(Y = Y_in, hermite_features)
    colnames(regression_df) <- c("Y", paste0("H", 0:degree))

    regression_formula <- as.formula(paste("Y ~", paste(colnames(regression_df)[-1], collapse = " + ")))
    regression <- lm(regression_formula, data = regression_df)

    all_features <- hermite_basis(X, degree)
    colnames(all_features) <- paste0("H", 0:degree)
    continuation <- predict(regression, newdata = as.data.frame(all_features))

    immediate_exercise <- pmax(K - X, 0)
    Cash_flow[, m-1] <- ifelse(continuation < immediate_exercise, immediate_exercise, 0)
  }

  # Discounting cash flows
  for (i in 1:N) {
    for (j in 1:M) {
      if (Cash_flow[i, j] != 0) {
        Cash_flow[i, j] <- Cash_flow[i, j] * round(exp(-r * j), 5)
        if (j < M) Cash_flow[i, (j+1):M] <- 0
        break
      }
    }
  }

  return(mean(rowSums(Cash_flow)))
}
```

# (Appendix F) Convergence plots for Regular, Laguerre, and Hermite Polynomials

Convergence plots for Regular, Laguerre, and Hermite Polynomials Notebook Github

# (Appendix G) Monte Carlo Algorithm using Custom Polynomials

## Black-Scholes formula for European call and put options

```
# Black-Scholes formula for European call
bs_call <- function(S, K, r, T, sigma) {
  set.seed(123)
  d1 <- (log(S / K) + (r + 0.5 * sigma^2) * T) / (sigma * sqrt(T))
  d2 <- d1 - sigma * sqrt(T)
  S * pnorm(d1) - K * exp(-r * T) * pnorm(d2)
}

# Black-Scholes formula for European put
bs_put <- function(S, K, r, T, sigma) {
  set.seed(123)
  d1 <- (log(S / K) + (r + 0.5 * sigma^2) * T) / (sigma * sqrt(T))
  d2 <- d1 - sigma * sqrt(T)
```

```r
  K * exp(-r * T) * pnorm(-d2) - S * pnorm(-d1)
}
```

### Function to price an American put using European call and put options

```r
price_american_put_longstaff_schwartz_MC_euro <- function(K, M, N, r, S0, sigma, polynomial) {
  dt <- 1 / M
  discount <- exp(-r * dt)
  set.seed(123)
  Z <- matrix(rnorm(N * M), nrow = N, ncol = M)
  S <- S0 * exp(sigma * sqrt(dt) * t(apply(Z, 1, cumsum)))

  Cash_flow <- matrix(0, nrow = N, ncol = M)
  Cash_flow[, M] <- pmax(K - S[, M], 0)

  for (m in M:2) {
    X <- S[, m - 1]
    T_remaining <- (M - m + 1) * dt

    call_bs <- bs_call(X, K, r, T_remaining, sigma)
    put_bs <- bs_put(X, K, r, T_remaining, sigma)

    df_reg <- data.frame( # Create regressors using custom functions
      S = X,
      call = call_bs,
      put = put_bs,
      Y = Cash_flow[, m] * discount
    )

    df_reg[X > K, ] <- NA

    if (all(is.na(df_reg))) {
      Cash_flow[, m - 1] <- 0
      next
    }

    regression <- lm(polynomial, data = df_reg)
    immediate_exercise <- pmax(K - X, 0)

    df_pred <- data.frame(
      S = X,
      call = call_bs,
      put = put_bs
    )
    continuation <- predict(regression, newdata = df_pred)
    Cash_flow[, m-1] <- ifelse(continuation < immediate_exercise, immediate_exercise, 0)
  }

  # Discounting cash flows
  for (i in 1:nrow(Cash_flow)) {
    for (j in 1:ncol(Cash_flow)) {
      if (Cash_flow[i, j] != 0) {
        Cash_flow[i, j] <- Cash_flow[i, j] * round(exp(-r * j * dt), 5)
```

```
        if (j < ncol(Cash_flow)) {
          Cash_flow[i, (j + 1):ncol(Cash_flow)] <- 0
        }
        break
      }
    }
  }

  return(mean(rowSums(Cash_flow)))
}
```

## (Appendix H) Covergence Plots using European Options as Regressors

Convergence Plots using European Options as Regressors Notebook Github

## (Appendix I) Convergence Error with Degree In-The-Money

Convergence Error with Degree In-The-Money Notebook Github