Q3_main

2024-10-30

```r
library(knitr)
library(kableExtra)

simulate_S_paths <- function(S0,r,q,sigma,n,m,delta_t) {
  #Function for simulating discrete-time stock paths of m steps with n paths based on the
  # BSM risk neutral measure
  S = rep(S0,n)
  S_df = data.frame(matrix(ncol = 0, nrow = n))
  for (i in 1:m) {

  delta_S =  S * (r-q) * delta_t + sigma * S * sqrt(delta_t) * rnorm(n)
  S = S + delta_S
  S_df[[paste0("S",i)]] = S
  }
  return(S_df)
}

laguerre_poly <- function(X, i) {
  # Function to calculate the Laguerre Polynomials. Used for creating regressors
  # Args:
  #   X (vector): Underlying prices
  #   i (int): order of the polynomial

  # Ensure X is a vector
  if (!is.vector(X)) stop("X must be a vector")

  # Calculate Laguerre polynomial L_i(X)
  result <- 0
  for (n in 0:i) {
    result <- result + ((-1)^n * choose(i, n) * X^n / factorial(n))
  }

  return(result * exp(-X / 2))  # Apply exponential decay for the generalized form
}

# Closed form BSM solution for European put
european_put_BS <- function(S,K,r,q,sigma,t) {

  d1 = (log(S / K) + (r - q + 0.5 * sigma^2) * t) / (sigma * sqrt(t))
  d2 = d1 - sigma * sqrt(t)

  put_price = K * exp(-r * t) * pnorm(-d2) - S0 * exp(-q * t) * pnorm(-d1)
  return(put_price)
}
```

```
LSM_put <- function(S0,K,r,q,sigma,t,n,m, k_regressors) {
  # Function to run the Least-Squares Monte Carlo
  delta_t = t/m # time steps
  S_df = simulate_S_paths(S0,r,q,sigma,n,m,delta_t) # simulate stock paths
  exercise_times = rep(m,n) # intialize stopping times at expiration
  # create payoff dataframe for all discrete times
  payoff_df = data.frame(apply(K - S_df, c(1, 2), function(x) max(x, 0)))

  # scaling underlying stock prices to prevent numerical issues with polynomials
  S_scaled_df = S_df/K
  # Recursively loop backwards to apply LSM
  for (i in (m-1):1) {
    itm_idx = payoff_df[,i] > 0 # find ITM idx
    # get future payoffs according to current stopping times
    future_cashflows = mapply(function(row, col) payoff_df[row, col], row = 1:nrow(payoff_df), col = exe
    # get times to discount according to current stopping times
    discount_times = delta_t * (exercise_times - i)
    # define target as present value of future payoffs
    Y = future_cashflows * exp(-r*discount_times[itm_idx])
    # filter only ITM underlying stock prices
    S_itm = S_scaled_df[,i][itm_idx]
    # Create Laguerre polynomial regressors matrix
    X = data.frame(matrix(ncol = 0, nrow = length(S_itm)))
    for (o_i in 0:(k_regressors-1)) {

      X[[paste0("L",o_i)]] = laguerre_poly(S_itm, o_i)
      # X[[paste0("X",o_i + 1)]] = S_itm**(o_i + 1)
    }
    # Run OLS and calculate conditional expectation of Y/X
    model = lm(Y ~ ., data=X)
    cond_exp_Y = predict(model, newdata = data.frame(X))
    names(cond_exp_Y) = NULL
    # If current payoff exceeds E[Y|X], then exercise now, if not in the future
    # To implement this logic, we update our stopping times
    current_itm_payoff = payoff_df[,i][itm_idx]
    exercise_times[itm_idx] = ifelse(current_itm_payoff > cond_exp_Y, i, exercise_times[itm_idx])
  }
  # get future payoffs according to final stopping times, and discount them
  payoff_decisions = mapply(function(row, col) payoff_df[row, col], row = 1:nrow(payoff_df), col = exerc
  discount_times = delta_t * (exercise_times - i)
  option_path_values = payoff_decisions * exp(-r*discount_times)
  # option value is the mean of all option present values from each path
  option_value = mean(option_path_values)
  se = sd(option_path_values) / sqrt(n)
  # % of paths that are early exercised
  early_exercise_portion = mean(exercise_times < m)
  return(list(value = option_value, se=se, early_portion=early_exercise_portion, ee_times=exercise_time
}
```

```r
# main code to run LSM for list of m steps, n paths, and k regressors
# output two dataframes: option value and se, early exercise value and
# % of early exercise paths.
# Early exercise value = American (LSM) value - European value
set.seed(3)
S0 = K = 100
t = 1/12
r = 0.04
q = 0.02
sigma = 0.2

european_put_value = european_put_BS(S0,K,r,q,sigma,t)

m_list = c(10,20,30,40,50)
n_list = c(1000,1000*4, 1000*4**2, 1000*4**3, 1000*4**4)
k_regressors_list = c(1,2,3)

put_LSM_results = data.frame()
put_LSM_results2 = data.frame()

for (k_regressors in k_regressors_list) {
  temp_results = data.frame(matrix(ncol = 0, nrow = length(n_list)))
  temp_results2 = data.frame(matrix(ncol = 0, nrow = length(n_list)))

  for (m in m_list) {
    value_list = c()
    se_list = c()

    early_portion_list = c()
    early_value_list = c()
    for (n in n_list) {
      option_LSM_res = LSM_put(S0,K,r,q,sigma,t,n,m, k_regressors)
      value_list = c(value_list, option_LSM_res$value)
      se_list = c(se_list, option_LSM_res$se)

      early_value_list = c(early_value_list, option_LSM_res$value - european_put_value)
      early_portion_list = c(early_portion_list, option_LSM_res$early_portion)

    }
    temp_results[[paste0("value_m",m)]] = value_list
    temp_results[[paste0("se_m",m)]] = se_list

    temp_results2[[paste0("EE_value_m",m)]] = early_value_list
    temp_results2[[paste0("Pct_EE_m",m)]] = early_portion_list
  }
  n_names = c()
  for (n in n_list) {n_names = c(n_names, paste0("k",k_regressors,",","n",n))}
  rownames(temp_results) = n_names
  rownames(temp_results2) = n_names
  put_LSM_results = rbind(put_LSM_results, temp_results)
  put_LSM_results2 = rbind(put_LSM_results2, temp_results2)
}
```

	value_m10	se_m10	value_m20	se_m20	value_m30	se_m30	value_m40	se_m40	value_m50	se_m50
k1,n1000	2.204171	0.0890545	2.159549	0.0862710	2.291037	0.0922878	2.226129	0.0887805	2.183315	0.0876205
k1,n4000	2.244613	0.0468933	2.265402	0.0451826	2.225166	0.0436517	2.225083	0.0461742	2.212701	0.0440833
k1,n16000	2.233609	0.0232139	2.226619	0.0229183	2.194444	0.0226021	2.184176	0.0221193	2.224474	0.0222317
k1,n64000	2.225113	0.0115267	2.215055	0.0112761	2.228521	0.0112392	2.197249	0.0109480	2.222790	0.0110832
k1,n256000	2.211166	0.0057492	2.228068	0.0056760	2.217719	0.0055785	2.212058	0.0055333	2.211259	0.0055212
k2,n1000	2.324024	0.1036054	2.238239	0.0857458	2.363621	0.0840589	2.204242	0.0962105	2.319409	0.0868074
k2,n4000	2.193251	0.0484682	2.223317	0.0445525	2.150080	0.0442447	2.192207	0.0420381	2.200488	0.0437335
k2,n16000	2.215614	0.0226425	2.206705	0.0220807	2.234868	0.0217439	2.246475	0.0213929	2.236706	0.0214626
k2,n64000	2.211044	0.0113288	2.229560	0.0112938	2.200900	0.0108737	2.231524	0.0108957	2.212395	0.0107106
k2,n256000	2.222746	0.0056378	2.213741	0.0054861	2.220810	0.0054394	2.221647	0.0053826	2.219348	0.0053481
k3,n1000	2.141900	0.0977352	2.460723	0.1009352	2.250335	0.0939327	2.195842	0.0856802	2.189706	0.0839213
k3,n4000	2.241029	0.0459363	2.220507	0.0456327	2.260753	0.0468733	2.248462	0.0425968	2.234902	0.0478318
k3,n16000	2.241842	0.0234925	2.210562	0.0231888	2.255139	0.0235969	2.209144	0.0222908	2.193897	0.0212058
k3,n64000	2.208245	0.0111802	2.224249	0.0111573	2.206882	0.0109662	2.241031	0.0109252	2.226777	0.0107753
k3,n256000	2.213651	0.0056771	2.226842	0.0056405	2.213046	0.0055344	2.218747	0.0054628	2.223026	0.0054843

	EE_value_m10	Pct_EE_m10	EE_value_m20	Pct_EE_m20	EE_value_m30	Pct_EE_m30	EE_value_m40	Pct_EE_m40	EE_value_m50	Pct_EE_m50
k1,n1000	-0.0108856	0.1700000	-0.0555072	0.1930000	0.0759800	0.2870000	0.0110722	0.2250000	-0.0317411	0.4760000
k1,n4000	0.0295560	0.1502500	0.0503459	0.2300000	0.0101096	0.2820000	0.0100265	0.2092500	-0.0023555	0.2627500
k1,n16000	0.0185528	0.1761250	0.0115630	0.2150625	-0.0206125	0.2023750	-0.0308809	0.2389375	0.0094178	0.2969375
k1,n64000	0.0100563	0.1828594	-0.0000018	0.2087031	0.0134649	0.2585000	-0.0178077	0.2711250	0.0077329	0.2703594
k1,n256000	-0.0038908	0.1707539	0.0130114	0.2140156	0.0026621	0.2430273	-0.0029984	0.2655508	-0.0037980	0.2704414
k2,n1000	0.1089672	0.0240000	0.0231828	0.2910000	0.1485645	0.3970000	-0.0108147	0.3110000	0.1043521	0.5270000
k2,n4000	-0.0218053	0.2062500	0.0082602	0.2792500	-0.0649768	0.2090000	-0.0228497	0.3245000	-0.0145687	0.3070000
k2,n16000	0.0005575	0.2411250	-0.0083516	0.2860000	0.0198119	0.2850625	0.0314183	0.3133750	0.0216500	0.3338750
k2,n64000	-0.0040127	0.2345938	0.0145039	0.2647031	-0.0141569	0.2811094	0.0164674	0.3060312	-0.0026620	0.3195156
k2,n256000	0.0076899	0.2373828	-0.0013161	0.2826602	0.0057536	0.2831641	0.0065901	0.3057930	0.0042911	0.3115078
k3,n1000	-0.0731566	0.2320000	0.2456665	0.1620000	0.0352782	0.4030000	-0.0192144	0.3570000	-0.0253507	0.3930000
k3,n4000	0.0259725	0.2990000	0.0054505	0.3137500	0.0456968	0.1607500	0.0334055	0.3470000	0.0198451	0.3460000
k3,n16000	0.0267854	0.2581875	-0.0044945	0.3021875	0.0400827	0.3316875	-0.0059124	0.3615625	-0.0211595	0.3591250
k3,n64000	-0.0068119	0.2765781	0.0091924	0.3073906	-0.0081742	0.3290469	0.0259747	0.3504531	0.0117206	0.3594062
k3,n256000	-0.0014053	0.2728398	0.0117858	0.3055664	-0.0020110	0.3276445	0.0036900	0.3375273	0.0079697	0.3541289