American Style Options - Regression Methods on Pricing

Supervisor: Prof. Mihalis Markakis

Team DUNY Dom, Uwe, Niti, Yiqun

April 2016

Abstract

In this report we introduce and analyze a simulation-based approximate dynamic programming method for pricing complex American-style options. The usual starting point is the binomial option pricing model, but we extend our analysis further by using regression methods. Our methods involve the evaluation of value functions at a finite set, consisting of "representative" elements of the state space. We show that with an arbitrary choice of this set, the approximation error can grow exponentially with the time horizon (time to expiration).

Introduction

What is an 'American-style' Option?

American style option contracts are traded extensively over several exchanges. It differs from the European options in that it gives the holder the right to exercise at any time during the contract period. Between the start date and a prescribed expiry date in the future, the holder may choose to buy or sell some prescribed underlying asset (S) at a prescribed price called the strike price (K) at any time. The exercise time τ can be represented as a stopping time. Assuming that the exercise decision is made to maximize the payoff, option price can be determined by computing the discounted expectation of the payoff of the option under a risk-neutral measure.

However, this large range of possible stopping times makes the valuation of American option enormously difficult. The holder of an American option is thus faced with the dilemma of deciding when, if at all, to exercise. If, at time t, the option is out-of-the-money then it is clearly best not to exercise. However, if the option is in-the-money it may be beneficial to wait until a later time where the payoff might be even bigger.

Define the problem

The price of the option is given by:

$$\sup_{\tau \in [0,\mathcal{T}]} \mathbb{E}[e^{-r\tau} g(x_\tau)]$$

where

- $\{x_\tau \in \Re^d | 0 \leq \tau \leq \mathcal{T}\}$
- risk-neutral process, assumed to be Markov
- r - risk-free interest rate, assumed to be a known constant
- $g(x)$ - intrinsic value of the option at state x
- $\mathcal{T}$ - expiration time - the supremum is taken over all possible stopping times in $[0, \mathcal{T}]$

Without loss of generality it is assumed that $\mathcal{T}$ is equal to an integer N and that allowable exercise times are separated by a time interval of unit length.

The price of this option is then:

$$\sup_{\tau} \mathbb{E}[\alpha^{\tau} g(x_{\tau})]$$

where $\alpha = e^{-r}$ and $g(x_n) = \max(0, K - S)$ for a put option. In this discrete-time and Markovian formulation, the dynamics of the risk-neutral process can be described by a transition operator P, defined by:

$$(PJ)(x) = \mathbb{E}[J(x_{n+1})|x_n = x]$$

The above expression does not depend on n, since the process is assumed time-homogeneous. A primary motivation for this discretization is that it facilitates exposition of computational procedures, which typically entail discretization. The algorithm generates a sequence $J_N, J_{N-1}, J_{N-2}...J_0$ of value functions, where J_n is the price of the option at time n, if x_n is equal to x. The value functions are generated iteratively according to

$$J_N = g$$

and

$$J_n = \max(g, \alpha P J_{n+1}) \qquad n = (N-1, N-2, \ldots, 0),$$

where the optimal is $J_N(x_0)$. In principle, value iteration can be used to price any option. However, the algorithm suffers from the curse of dimensionality; the computation time grows exponentially in the number d of state variables. This difficulty arises because computations involve discretization of the state space, which leads to a grid whose size grows exponentially with multiple sources of uncertainty. Also, one value is computed and stored for each point in the grid leading to extensive storage space as well as exponential growth in the computation time.

Approximations

$\tilde{J} : \Re^d \times \Re^K \mapsto \Re$ which assigns values $\tilde{J}(x, r)$ to states x where $r \in \Re^K$ is a vector of free parameters. The objective then becomes to choose, for each n, a parameter vector r_n so that:

$$\tilde{J}(x, r) = f(\phi(x), r)$$

$$\tilde{J}(x, r_n) \approx J_n(x)$$

One may define several features $\phi_1, ..., \phi_k$. Then to each state $x \in \Re^d$ we associate a feature vector $\phi(x) = (\phi_1(x), ..., \phi_k(x))^T$ which represents the most salient properties of the given state.

The choice of the features and parameter requires theoretical analysis, human experience and that some computation of appropriate values is possible. In this paper we restrict our analysis to the least square method.

This simplest form of approximate value iteration involves a single projection matrix Π that projects onto the feature space with respect to a weighted quadratic norm. We extend our analysis to sample-based time dependent projection. Generally exact computation of projection is not viable. However, one can approximate effectively by sampling a collection of states $y_1, ..., y_m \in \Re^d$ according to the probability measure π and then defining an approximate projection operator

$$\tilde{\Pi} J = argmin_{\Phi r} \sum_{i=1}^{m} (J(y_i) - (\Phi r)(y_i))^2$$

.

The difference between the exact and the approximate value converges to 0 w.p. 1 as our sample size m grows. Now one can define an approximate value iteration version as

$$\tilde{J}(\cdot, r_{N-1}) = \tilde{\Pi} \max(g, \alpha P g)$$

and

$$\tilde{J}(\cdot, r_n) = \tilde{\Pi} \max(g, \alpha P \tilde{J}(\cdot, r_{n+1}))$$

The drawback with this method, however, is that for each sample y_i and any function J we need to compute the expectation $(PJ)(y_i) = \mathbb{E}[J(x_{k+1})|x_k = y_i]$ which is over a potentially high dimensional space. This poses a computational challenge which can be resolved by Monte Carlo simulation.

For each sample y_i we can simulate independent samples $z_{i,1}, ..., z_{i,l}$ from the transition distribution conditioned on the state being y_i.

Approximation using Q-Values and single samples estimates

A more convenient approach relies on single sample estimates of the desired expectation. Define for each $n = 0, ..., N-1$ a Q-function

$$Q_n = \alpha P J_{n+1}$$

where $Q_n(x)$ represents the expected discount payoff at time n conditioned on a decision not to exercise. This modification gives a new version of our dynamic pricing algorithm.

$$\tilde{Q}(\cdot, r_{N-1}) = \alpha \tilde{\Pi} \tilde{P} g$$
$$\tilde{Q}(\cdot, r_n) = \alpha \tilde{\Pi} \tilde{P} \max(g, \tilde{Q}(\cdot, r_{n+1}))$$

where $n = N-2, N-1, ..., 0$

Here we base our approximation of the expectation of only a single sample. We select m independent random samples of the state, $y_1, ..., y_m$, according to the probability measure π, and for each y_i, we simulate a successor state z_i. Then our parameter vector r_n is found by minimizing

$$\sum_{i=1}^{m} \left(\alpha \max \left\{ g(z_i), \sum_{k=1}^{K} r_{n+1}(k) \phi_k(z_i) \right\} - \sum_{k=1}^{K} r(k) \phi_k(y_i) \right)^2$$

with respect to $r_1, ..., r_K$.

Given a sample state y_i, the expected value (with respect to the random next state z_i) of $(\tilde{P}J)(y_i) = (PJ)(y_i)$ for any function J, making our approximation an unbiased estimate the true value. Note that $\tilde{P}$ enters linearly and effectively allows for the noise to be averaged out. This was not possible in the original version of approximation where the dependence of $\tilde{P}$ was nonlinear.

In a more general version of the algorithm we introduce probability measures $\pi_0, ..., \pi_{N-1}$ such that for each time n we generate m random states $y_{ni}, ..., y_{nm}$ sampled independently. This leads to an approximate projection and our dynamic pricing algorithm becomes

$$\tilde{Q}(\cdot, r_{N-1}) = \alpha \tilde{\Pi}_{N-1} \tilde{P} g$$
$$\tilde{Q}(\cdot, r_n) = \alpha \tilde{\Pi}_n \tilde{P} \max(g, \tilde{Q}(\cdot, r_{n+1}))$$

where $n = N-2, N-1, ..., 0$

If the sample states used in the approximate projections are obtained by simulating trajectories of the process x_n then the approximation error does not grow exponentially but rather of the order $\sqrt{N}\epsilon$. The approximation error of the idealized algorithm is the best possible error under the chosen approximation architecture. $\epsilon_n^* = \min_r \|\tilde{Q}(\cdot, r_n) - Q_n\|_{\pi_n} = \|\Pi_n Q_n - Q_n\|_{\pi_n}$.

Numerical Example and Code

Now we consider a somewhat more realistic example.

In this table we have listed the input values for pricing the American Put

strike	$\bar{x}$	1
high return	u	3/2
low return	d	2/3
probability of high return	p	0.4121
discount factor	α	0.99

We will approximate the value function using a quadratic. For each n, the weights are given by a three-dimensional vector $r_n \in \Re^3$ and the approximation is defined by

$$\tilde{Q}(x, r_n) = r_n(1) + r_n(2)x + r_n(3)x^2$$

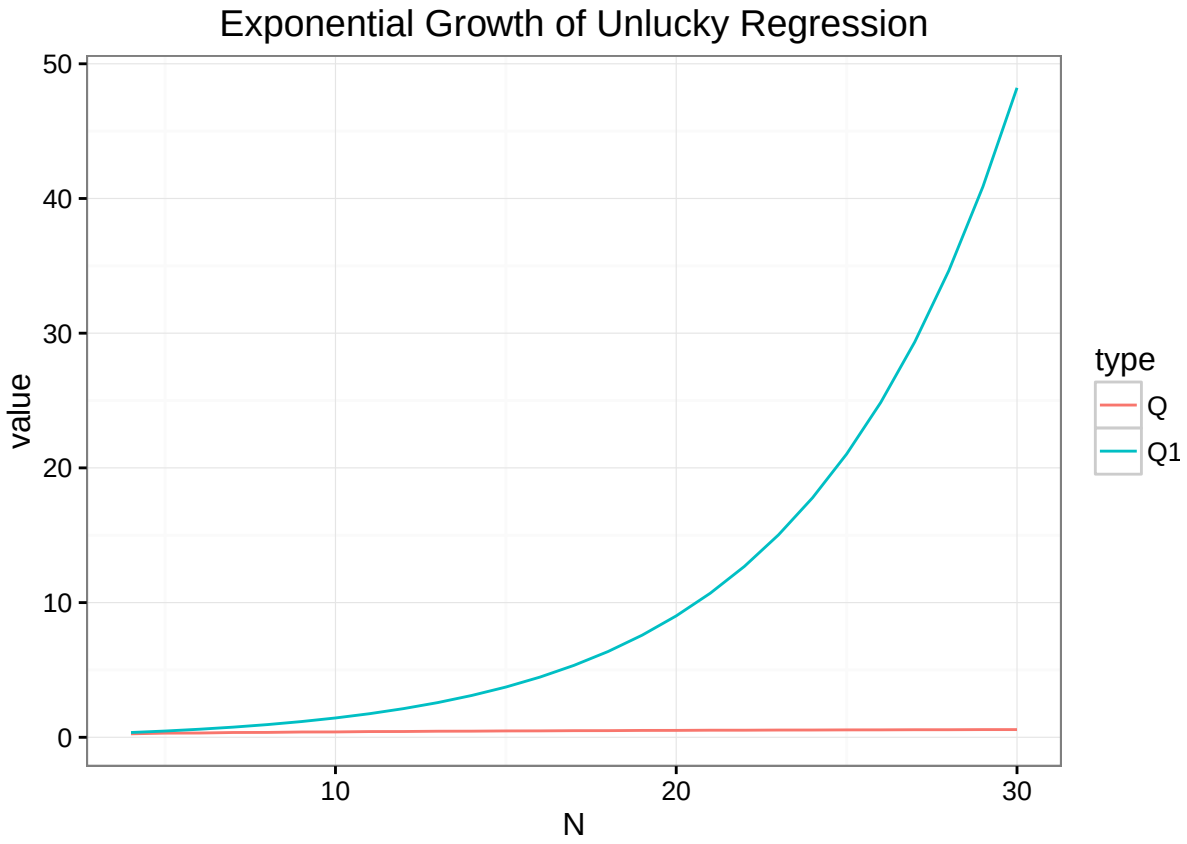
.

Further plots

Exponential Growth Rate

```
tbl <- data.frame(N=4:30, Q=Q_collect, Q1=Q1_collect)
tbl <- melt(tbl, id.vars = "N", variable.name = "type")


ggplot(aes(x = N, y = value, color = type), data = tbl)+
  geom_line()+
  theme_bw()+
  ggtitle("Exponential Growth of Unlucky Regression")
```

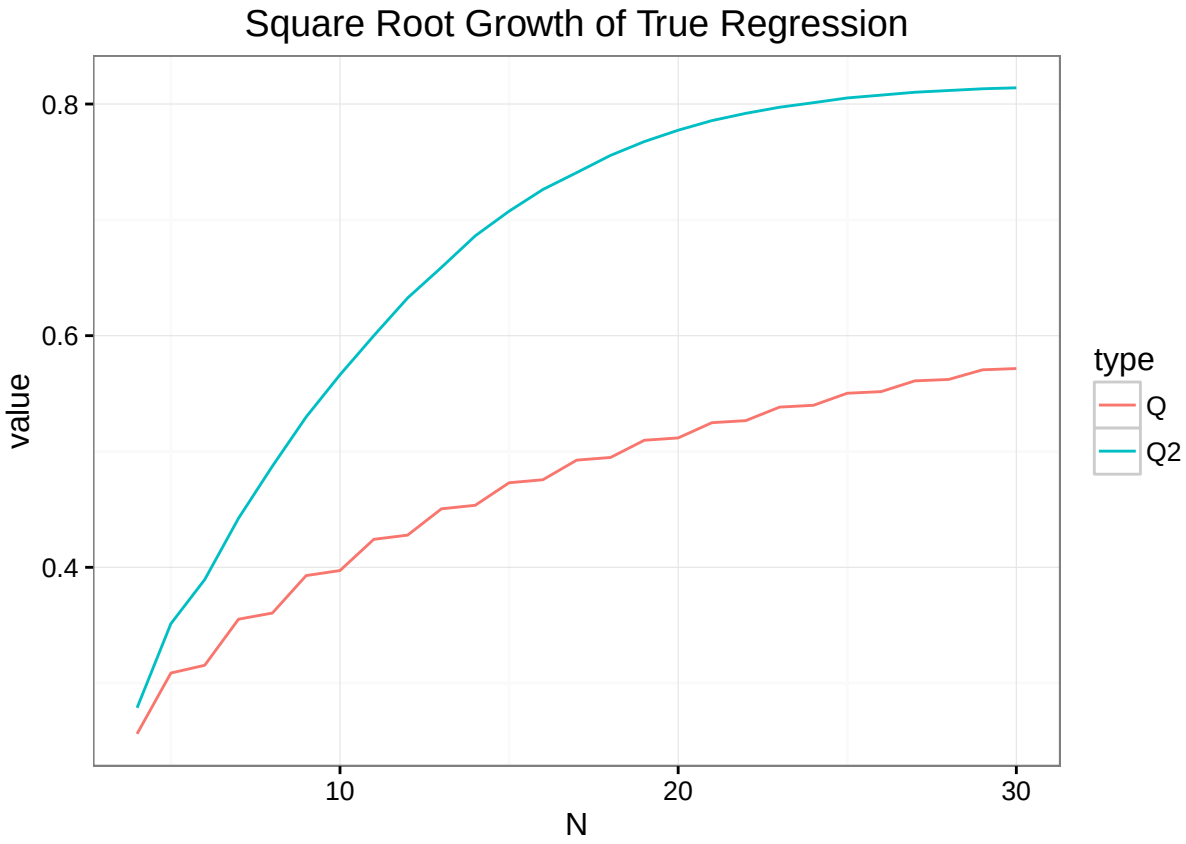


We can see that if we let the distribution assign probabilities uniformly to points in a discrete grid spread over a segment of the state space which is far from the true data generating process π_t at each state, approximate value iteration algorithms lead to errors that grow exponentially in the problem horizon.

Square Root Growth Rate

```
tbl2 <- data.frame(N=4:30, Q=Q_collect, Q2=Q2_collect)
tbl2 <- melt(tbl2, id.vars = "N", variable.name = "type")


ggplot(aes(x = N, y = value, color = type), data = tbl2)+
  geom_line()+
  theme_bw()+
  ggtitle("Square Root Growth of True Regression")
```

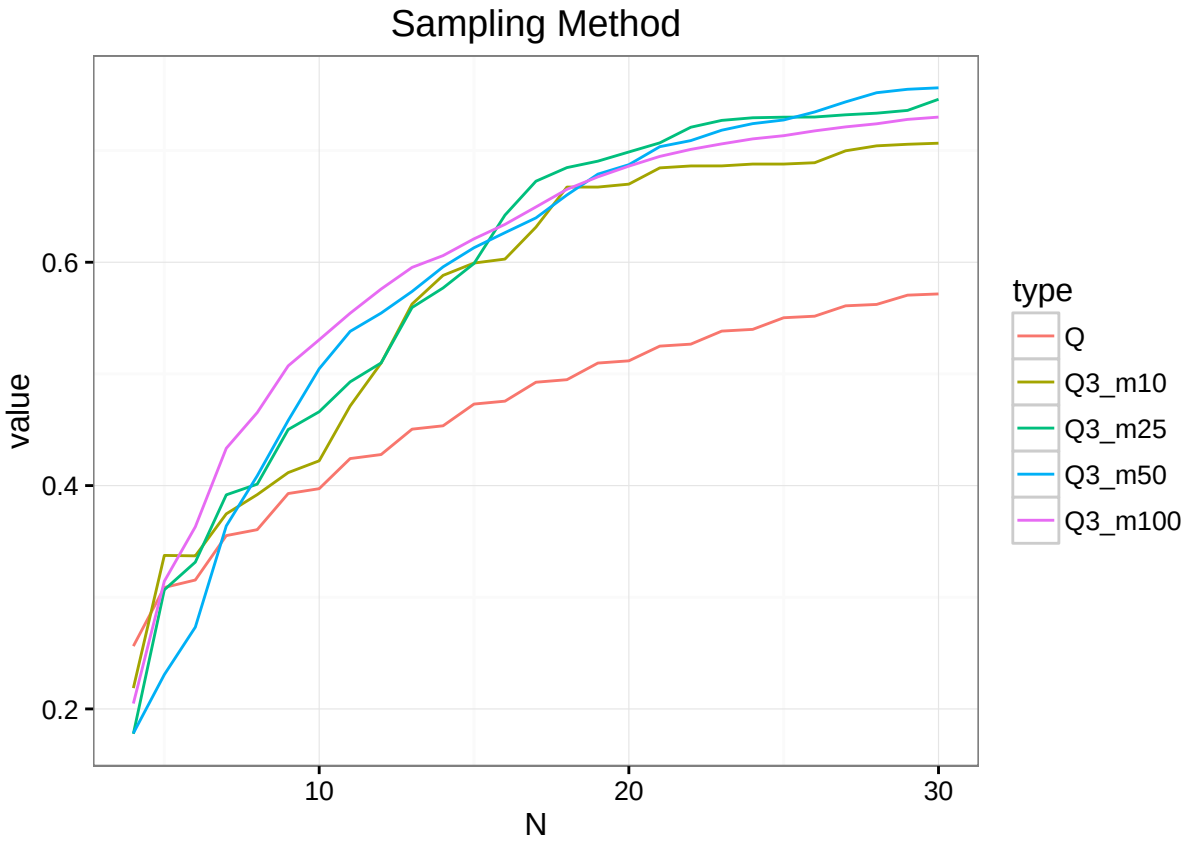


However, if the sample states used in approximate projection are obtained according to true probability π_t, then the approximation error grows no faster than $\sqrt{N}$. The figure above uses exact calculation of expectations and projections.

Sample-based Method

```
tbl3 <- data.frame(N=4:30, Q=Q_collect,
                   Q3_m10 = Q3_collect[,1],
                   Q3_m25 = Q3_collect[,2],
                   Q3_m50 = Q3_collect[,3],
                   Q3_m100 = Q3_collect[,4])
tbl3 <- melt(tbl3, id.vars = "N", variable.name = "type")


ggplot(aes(x = N, y = value, color = type), data = tbl3)+
  geom_line()+
  theme_bw() +
  ggtitle("Sampling Method")
```



So starting from the given initial state, we can simulate independent trajectories of the process to run the approximate projection. The above figure shows that the additional errors introduced by random sampling of the state space are not the main issue of our toy example.

Remarks: Even the paper gives an asymptotic result on the dependence of sampling error and sample size m and suggests drawing samples as many as possible. Of course the sample size should not be too small, since it will cause sampling bias or even make the the input matrix of regression singular. But for our toy example(or other applications), the variance of some feature mapping on state variable is quite large especially in later stages. In other words, more sample draws means higher possibility of getting outliers. This is a bad thing for linear regressions because they might not be robust and sensitive to outliers. So it will cause extimation errors of weights accumulated during the dynamic programming. The most serious situation is to create **computational singular** input matrix when extimating the regression coefficients and then stop the iterations in an unexpected way.

Conclusion

For simple contracts, including "vanilla options" such as American puts and calls, the relevant optimal stopping problems can be solved efficiently by traditional numerical methods. However, the computational requirements become prohibitive as the number of uncertainties of a contract grows.

- We introduced certain simulation-based methods of the value iteration type for pricing complex American-style options.
- provided convergence results and error bounds that establish that such methods are viable, as long as state sampling is carried out by simulating the natural distribution of the underlying state process.
- This provides theoretical support for the apparent effectiveness of this particular form of state sampling.

```r
#generate the process sample space(merged binomial tree)
sim.Price_Bin <- function(N=50, u=3/2, d=2/3, x0 =1){
  x <- matrix(0, nrow = N+1, ncol = N+1)

  for(i in 1: (N+1)){
    for(j in 1:i){
      x[i, j]=x0*u^(j-1)*d^(i-j)

    }

  }
  return(x)

}


#generate the corresponding probability
sim.Prob_Bin <- function(N=50, p = 0.4121){
  pi <- matrix(0, nrow = N+1, ncol = N+1)

  for(i in 1: (N+1)){
    for(j in 1:i){
      pi[i,j] <- dbinom(x = j-1, size = i-1, prob = p )

    }

  }
  return(pi)

}


#payoff function
g <- function(x, strike = 1){
  return(pmax(0, strike - x))

}


#Expectation operator
P <- function(state, p = 0.4121){
  return(sum(state*c(1-p, p)))

}


#some parameters
x_strike <- 1
alpha <- 0.99
p = 0.4121
x0=1.1
u = 3/2
d = 2/3
```

```r
#exact value of J
J.exact <- function(N=25,...){
  J <- matrix(0, nrow = N+1, ncol = N+1)

  J[N+1,] <- g(x[N+1,],x_strike)

  for(i in N:1){
    for(j in 1:i){
      J[i,j] <- pmax(g(x[i,j],x_strike), alpha * P(J[i+1,c(j,j+1)],p))

    }

  }
  return(J[1,1])

}


#exact Q function
Q.exact <- function(N=25,...){
  Q <- matrix(0, nrow = N+1, ncol = N+1)

  Q[N+1,] <- g(x[N+1,],x_strike)

  for(j in 1:N){
    Q[N, j] <- alpha * P(Q[N+1,c(j,j+1)], p)

  }

  for(i in (N-1):1){
    for(j in 1:i){
      Q[i,j] <- alpha * P(pmax(g(x[i+1,c(j, j+1)],x_strike), Q[i+1,c(j,j+1)]), p)

    }

  }
  return(Q[1,1])

}


##Regression with fictitious pi
dim <- 3
S <- seq(from = 0.1, to = 2, by = 0.1)


#feature mapping
phi1 <- function(x){
  return(rep(1, length(x)))

}
```

```r
phi2 <- function(x){
  return(x)

}


phi3 <- function(x){
  return(x^2)

}


#conbind features to get input matrix
phi <- function(x, dim = 3, base = list(phi1, phi2, phi3)){
  base_Mat <- matrix(0, nrow = length(x), ncol = dim)

  for( j in 1:dim){
    base_Mat[,j] <- base[[j]](x)

  }
  return(base_Mat)

}


#Regression with gussing probability
Q1.approx <- function(N=25,...){
  r <- matrix(0, ncol = N, nrow = dim )
  Q1 <- matrix(0, nrow = N, ncol = length(S))

  for(j in 1:length(S)){
    Q1[N, j] <- alpha * P(g(c(S[j]*d, S[j]*u)),p)

  }

  r[, N] <- solve((t(phi(S)) %*% phi(S)), t(phi(S)) %*% Q1[N,])

  for (i in (N-1):1){
    for(j in 1:length(S)){

      Q1[i, j] <- alpha*P(pmax(g(c(S[j]*d, S[j]*u)),
                               c(phi(S[j]*d) %*% r[,i+1], phi(S[j]*u) %*% r[,i+1])))

    }

    r[, i] <- solve((t(phi(S)) %*% phi(S)), t(phi(S)) %*% Q1[i,])

  }
  Q1_0 <- phi(S[11]) %*% r[,1]
  return(Q1_0)

}
```

```r
#Regression with true probability
Q2.approx.pop <- function(N=25,...){
  r2 <- matrix(0, ncol = N, nrow = dim )
  Q2<- matrix(0, nrow = N, ncol =  N)

  for(j in 1:N){
    Q2[N, j] <- alpha * P(g(x[N+1,])[c(j,j+1)], p)

  }

  r2[,N] <- solve(t(phi(x[N,1:N])) %*% (phi(x[N, 1:N])*pi[N,1:N]),
                  t(phi(x[N, 1:N]))%*% (Q2[N,1:N] *pi[N,1:N]))

  Q2[N,] <- phi(x[N,1:N]) %*% r2[,N]

  for(i in (N-1):dim){
    for(j in 1:i){
      Q2[i,j] <- alpha * P(pmax(g(x[i+1,c(j, j+1)]), Q2[i+1,c(j,j+1)]), p)

    }

    r2[,i] <- solve(t(phi(x[i,1:i])) %*% (phi(x[i,1:i]) * pi[i,1:i]),
                    t(phi(x[i, 1:i])) %*% (Q2[i, 1:i]* pi[i,1:i]))

    Q2[i, 1:i ] <- phi(x[i,1:i]) %*% r2[,i]

  }

  for(i in (dim-1):1){
    for(j in 1:i){
      Q2[i,j] <- alpha * P(pmax(g(x[i+1,c(j, j+1)]), Q2[i+1,c(j,j+1)]), p)

    }

  }
  return(Q2[1,1])

}


#simulate trajectories
sim.Trajectory <- function(m = 10, N=25, x0 = 1.1,...){
  traj <- matrix(0, nrow = m, ncol = N+1)
  traj[,1] <- x0

  for(j in 2:ncol(traj)){
    traj[,j] <- traj[,j-1]*ifelse(rbinom(m,1,prob = p), u, d)

  }
  return(traj)

}
```

```r
#Regression with sampling from true probability
Q3.approx.sim <- function(N=25,m=10,...){
  r3 <- matrix(0, ncol = N, nrow = dim )

  Q3<- matrix(0, nrow = N, ncol =  m)

  for(j in 1:m){
    Q3[N,j] <- alpha * g(traj[j,N+1])

  }

  r3[,N] <- solve(t(phi(traj[,N])) %*% phi(traj[,N]),
                  t(phi(traj[,N])) %*% Q3[N,])

  Q3[N,] <- phi(traj[,N]) %*% r3[,N]

  for(i in (N-1):dim){
    for(j in 1:m){
      Q3[i,j] <- alpha * pmax(g(traj[j,i+1]), Q3[i+1, j])

    }

    r3[,i] <- solve(t(phi(traj[,i])) %*% phi(traj[,i]),
                    t(phi(traj[,i])) %*% Q3[i,])

    Q3[i,] <- phi(traj[,i]) %*% r3[,i]

  }

  for(i in (dim-1):1){
    Q3[i,] <- phi(traj[,i]) %*% r3[,dim]

  }
  return(Q3[1,1])

}


#Caculate
Q_collect <-numeric(30-dim)
Q1_collect <-numeric(30-dim)
Q2_collect <- numeric(30-dim)
Q3_collect <- matrix(0, nrow = 30-dim, ncol = 4 )


#simulation sample size
m <- c(10, 25, 50, 100)

for(N in 4:30){
  x <- sim.Price_Bin(N=N, x0 = x0)

  pi <- sim.Prob_Bin(N=N, p = p)
```

```
  Q_collect[N-dim]<- Q.exact(N=N)

  Q1_collect[N-dim] <- Q1.approx(N=N)

  Q2_collect[N-dim] <- Q2.approx.pop(N=N)

  for(j in 1:length(m)){
    set.seed(1111)
    traj <- sim.Trajectory(N=N, m=m[j])

    Q3_collect[N-dim,j] <- Q3.approx.sim(N=N,m = m[j])

  }

}

library(ggplot2)
library(reshape2)
```

References

- Regression Methods for Pricing Complex American-Style Options by John N.Tsitsiklis, Benjamin Van Roy
- Pricing American Options by Simulation, Cem Coskan (2006)