rlsm: R PACKAGE FOR LEAST SQUARES MONTE CARLO

JEREMY YEE

ABSTRACT. This short paper briefly describes the implementation of the least squares Monte Carlo method in the *rlsm* package. This package provides users with an easy manner to experiment with the large amount of R regression tools on any regression basis and reward functions. This package also computes lower and upper bounds for the true value function via duality methods.

Keywords. Dynamic programming, Least squares Monte Carlo, Markov decision processes

1. INTRODUCTION

The popularity of the least squares Monte Carlo method [4, 13, 9] has been largely spurred on by its applications in finance and real options valuation. This method uses statistical regression to represent the continuation value functions in the Bellman recursion as a linear combination of selected feature functions. Unlike tradition approaches such as finite differences and tree methods, least squares Monte Carlo is largely independent of the dimension of the state space and so mostly avoids the so-called curse of dimensionality that is common in dynamic programming problems. A rigiourous treatment of this method is beyond the scope of this paper and so the reader is referred to the work done by [2] and the references contained within for a more comprehensive analysis of this approach and its convergence properties. In its typical form, least squares Monte Carlo employs linear least squares regression. However, as shown by [12], other regression approaches may be more appropriate such as Ridge regression or Least Absolute Shrinkage and Selection Operator (LASSO). Numerous regression approaches have been honed by statisticians and coded into the R statistical language [10]. The aim of the *rlsm* package is to allow least squares Monte Carlo users to access the large amount of regression tools developed by the R community. To the author's knowledge, this has not been done before in an R package. Note that this package only focuses on global regression methods (as opposed to local methods such as nearest neighbours). The *rlsm* [14] package is able to handle any specification of the regression basis and reward functions. In addition, this package also implements the dual approach studied by [1, 8, 11, 3] to construct lower and upper bound for the unknown value function using a pathwise dynamic

E-mail address: jeremyyee@outlook.com.au.

programming approach. The computational effort is done at $C++$ level via *Rcpp* [7]. Before proceeding, let us make a key point. This paper neglects some mathematical rigour in exchange for brevity. However, references are provided for the interested reader. The paper is structured as follows. The next section gives the problem setting. Section 3 gives a description of the least squares Monte Carlo algorithm. Section 4 details the construction of the lower and upper bounds. Section 5 demonstrates the usage of the *rlsm* package on a simple optimal stopping problem. Section 6 concludes this paper.

2. Markov decision process

Let $\mathbf{X} = \mathbf{P} \times \mathbf{Z}$ represent our state space and is the product of a finite set $\mathbf{P}$ and a subset of the Euclidean space $\mathbf{Z} \subseteq \mathbb{R}^d$. At time $t = 0, 1, \ldots, T$, an action $a \in \mathbf{A}$ is chosen and these actions influences the evolution of the stochastic process $(X_t)_{t=0}^T := (P_t, Z_t)_{t=0}^T$. The discrete component $(P_t)_{t=0}^T$ is a controlled Markov chain with transition probabilities $(\alpha_{p,p'}^a)_{p,p' \in \mathbf{P}, a \in \mathbf{A}}$, where $\alpha_{p,p'}^a$ is the probability of moving from p to p' after applying action a. The continuous component $(Z_t)_{t=0}^T$ evolves according to $Z_{t+1} = f_{t+1}(W_{t+1}, Z_t)$ where $(W_{t+1})_{t=0}^{T-1}$ are indepedent random variables and f_{t+1} is a measurable function. At each time $t = 0, \ldots, T$ the decision rule π_t is given by a mapping $\pi_t : \mathbf{X} \to \mathbf{A}$, prescribing at time t an action $\pi_t(p, z) \in \mathbf{A}$ for a given state $(p, z) \in \mathbf{X}$. A sequence $\pi = (\pi_t)_{t=0}^T$ of decision rules is called a policy. For each policy $\pi = (\pi_t)_{t=0}^T$, associate it with a so-called policy value $v_0^\pi(p_0, z_0)$ defined as the total expected cumulative reward

$$(1) \qquad v_0^\pi(p_0, z_0) = \mathbb{E}\left[\sum_{t=0}^{T-1} r_t(P_t, Z_t, \pi_t(X_t)) + r_T(P_t, Z_t)\right]$$

where r_t and r_T are the reward and scrap functions, respectively. A policy $\pi^* = (\pi_t^*)_{t=0}^T$ is called optimal if it maximizes the above expectation over all policies $\pi \mapsto v_0^\pi(p, z)$. If an optimal policy exists, it satisfies the Bellman recursion via

$$(2) \qquad \pi_t^*(p, z) = \arg\max_{a \in \mathbf{A}}\left\{r_t(p, z, a) + \sum_{p' \in \mathbf{P}} \alpha_{p,p'}^a \mathbb{E}[v_{t+1}^*(p', f_{t+1}(W_{t+1}, z))]\right\}$$

for $t = T - 1, \ldots, 0$.

Note that the assumption that $(Z_t)_{t=0}^T$ is uncontrolled is a simplyfing one since the least square Monte Carlo methods simulates a number of scenarious from (P_0, Z_0). Therefore, if the process $(Z_t)_{t=0}^T$ is controlled, the use of a reference probability measure and the corresponding densities is required to adjust the conditional expectaions in (2) (see problem formulation in Section 2 in [2]). This is difficult to implement and not typically used in practice and so is not considered in this paper and package.

3. Least squares Monte Carlo (LSM)

The goal of least squares Monte Carlo (LSM) is to express the conditional expectations in (2) as a linear combination of basis functions using values held by simulated paths. Suppose $(Z_t(\omega_i))_{t=0}^T$ represents simulated trajectory i. At terminal time $t = T$ and position p, the scrap $\widetilde{v}_T(p, \omega_i) := r_T(p, Z_T(\omega_i))$ is realized for each of the n sample paths. Now at $t = T-1$, the values $(\widetilde{v}_T(p, \omega_i))_{i=1}^n$ are then regressed on a chosen regression basis constructed using $(Z_{T-1}(\omega_i))_{i=1}^n$ to give an approximation of the conditional expectations which we will denote by $\widetilde{c}_T^a(p', z)$ for $p' \in \mathbf{P}$. Note that the regression is performed using all the simulated paths. Now for each path ω_i, determine fitted decision rule

$$\widetilde{\pi}_{T-1}(p, \omega_i) := \arg\max_{a \in \mathbf{A}}\{r_{T-1}(p, Z_{T-1}(\omega_i), a) + \sum_{p' \in \mathbf{P}} \alpha_{p,p'}^a \widetilde{c}_T^a(p', Z_{T-1}(\omega_i))\}$$

and resulting value obtained by each sample path

$$\widetilde{v}_{T-1}(p, \omega_i) := r_{T-1}(p, Z_{T-1}(\omega_i), \pi_{T-1}(p, \omega_i)) + \sum_{p' \in \mathbf{P}} \alpha_{p,p'}^{\widetilde{\pi}_{T-1}(p,\omega_i)} \widetilde{v}_T(p', \omega_i)$$

and proceed inductively for $t = T-2, \ldots, 1, 0$ until sample $(\widetilde{v}_0(p, \omega_i))_{i=1}^n$ is obtained. Many authors (e.g. [9]) has shown that the mean of $(\widetilde{v}_0(p, \omega_i))_{i=1}^n$ converges in probability to $v_0^{\pi^*}(p, z_0)$ as the number of sample paths and size of the regression basis grows to infinity. If the true value function can be expressed exactly as a linear combination of the selected basis functions, then the convergence is almost sure when $n \to \infty$ [5].

4. Lower and upper bounds

Now it is clear that a lower bound for $v_0^{\pi^*}(p_0, z_0)$ is given by

$$(3) \qquad \mathbb{E}\left[\sum_{t=0}^{T-1} r_t(P_t, Z_t, \widetilde{\pi}_t(P_t, Z_t)) + \varphi_{t+1}(P_t, Z_t, \widetilde{\pi}_t(P_t, Z_t)) + r_T(P_T, Z_T)\right]$$

where $(P_0, Z_0) = (p_0, z_0)$ a.s., $\widetilde{\pi}$ is some decision policy, and $(\varphi_t)_{t=1}^T$ are zero mean and independent random variables. Similarly, an upper bound is given by the expectation of the following pathwise maximum

$$(4) \qquad \max_{\pi} \sum_{t=0}^{T-1} r_t(P_t, Z_t, \pi_t) + \varphi_{t+1}(P_t, Z_t, \pi_t) + r_T(P_T, Z_T).$$

When $(\varphi_t)_{t=1}^T$ are zero mean and independently distributed, the upper bound represents the case where the controller has perfect foresight into the future. It turns out that the careful choice of $(\varphi_t)_{t=1}^T$ affects the location of these bounds. It is not hard to see that when $\widetilde{\pi}_t(x) = \pi_t^*(x)$ and $\varphi_t(p, z, a)$ is given by

$$\sum_{p' \in \mathbf{P}} \alpha_{p,p'}^a \left(\mathbb{E}[v_{t+1}^{\pi^*}(p', f_{t+1}(W_{t+1}, z))] - v_{t+1}^{\pi^*}(p', f_{t+1}(W_{t+1}, z)])\right)$$

for $t = 0, \ldots, T - 1$, both the lower and upper bounds coincide and gives the value function $v_0^{\pi^*}(p_0, z_0)$. This can be verified by substitution into (3) and (4). Please see Section 5 in [2] for the rigorous details.

In practice, the true value functions $v_t^{\pi^*}$ are unknown since its knowledge vitiates the need to perform numerical work in the first place. However, the function approximations from the least squares Monte Carlo can be used in their place instead i.e.

$$(5) \qquad \sum_{p' \in \mathbf{P}} \alpha_{p,p'}^a \left(\frac{1}{I} \sum_{i=1}^{I} \widetilde{v}_{t+1}(p', f_{t+1}(W_{t+1}^{(i)}, z)) - \widetilde{v}_{t+1}(p', f_{t+1}(W_{t+1}, z)) \right)$$

for some number I and where $\widetilde{v}_T(p, z) = r_T(p, z)$ and

$$\widetilde{v}_{t+1}(p, z) = \max_{a \in \mathbf{A}} r_{t+1}(p, z, a) + \sum_{p' \in \mathbf{P}} \alpha_{p,p'}^a \widetilde{c}_{t+1}^a(p', z), \quad t = T - 2, \ldots, 0.$$

With this substitution, the closer our regression approximations are to their true counterparts, the tighter the bound estimates and the smaller their standard errors. In this manner, these bound estimates allow us to partially gauge the quality of our function approximations as well as proving bounds for $v_0^{\pi^*}(p_0, z_0)$.

5. Demonstration: Bermudan put

The following numerical experiment was run on a Linux Ubuntu 16.04 machine with Intel i5-5300U CPU @2.30GHz and 16GB of RAM using the author's R package *rlsm* which can be found at: `https://github.com/YeeJeremy/rlsm`, and the package manual can be found at `https://github.com/YeeJeremy/RPackageManuals/blob/master/rlsm-manual.pdf`. In what follows, a Bermudan put option is considered. A Bermudan put option gives the owner the right but not the obligation to sell the underlying asset for a contracted strike price K at prespecified time points. In this setting, $\mathbf{P} = \{\text{excerised}, \text{unexercised}\}$ and $\mathbf{A} = \{\text{exercise}, \text{don't exercise}\}$. At $P_t = $ "unexercised", applying $a = $ "exercise" and $a = $ "don't exercise" leads to $P_{t+1} = $ "exercised" and $P_{t+1} = $ "unexercised", respectively with probability one. If $P_t = $ "exercised", then $P_{t+1} = $ "exercised" with probability one. Now represent the interest rate by κ and underlying asset price by z, the reward and scrap functions are given by

$$r_t(\text{unexercised}, z, \text{exercise}) = e^{-\kappa t}(K - z)^+,$$
$$r_T(\text{unexercised}, z) = e^{-\kappa T}(K - z)^+,$$

for all $z \in \mathbb{R}_+$ and zero for other $p \in \mathbf{P}$ and $a \in \mathbf{A}$. In the above $(K - z)^+ := \max(K - z, 0)$. The fair price of the option is given by

$$v_0^{\pi^*}(\text{unexercised}, z_0) = \max\left\{\mathbb{E}(\max(e^{-\kappa \tau}(K - Z_\tau), 0)) : \tau = 0, 1, \ldots, T\right\}.$$

The option is assumed to reside in the Black-Scholes world where the asset price process $(Z_t)_{t=0}^{T}$ follows geometric Brownian motion i.e.

$$Z_{t+1} = e^{(\kappa - \frac{\text{vol}2}{2})\Delta_{t+1} + \text{vol}\sqrt{\Delta_{t+1}}W_{t+1}} Z_t$$

where $(W_t)_{t=1}^{T}$ are independent standard normal random variables, Δ_{t+1} is the time step and vol is the volatility of stock returns.

Let us set up our model in the below code listing. In this example, the package *StochasticProcess*[15] was used to generate our paths. However, the user is free to do so in which ever manner they wish to. The simulated paths are represented by object `path` which gives a 3 dimensional array where entry $[i, j, k]$ gives the j-th component of $Z_{k-1}(\omega_i)$. For the case that P_t is governed deterministically by the actions, users can specify a control matrix (Line 10) instead of the more tedious transition probabilities (Lines 11-15).

LISTING 1. Set up

```
1   library(StochasticProcess)
2   ## Parameters
3   set.seed(123)
4   step <- 0.02 # Step size
5   kappa <- 0.06 * step ## Adjust interest according to step size
6   vol <- 0.2 * sqrt(step) ## Adjust vol according to step size
7   n_dec <- 51 # Number of decision times T + 1
8   strike <- 40 # Strike price
9   ## The transition for P_t. See manual for more information.
10  control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
11  ## control <- array(data = 0, dim = c(2,2,2))
12  ## control[2,1,2] <- 1
13  ## control[2,2,1] <- 1
14  ## control[1,1,1] <- 1
15  ## control[1,2,1] <- 1
16  ## Reward and scrap functions
17  Reward <- function(state, time) {
18      output <- array(data = 0, dim = c(nrow(state), 2, 2))
19      output[, 2, 2] <- exp(-kappa * (time - 1)) * pmax(strike - state, 0)
20      return(output)
21  }
22  Scrap <- function(state) {
23      output <- matrix(data = 0, nrow = nrow(state), ncol = 2)
24      output[, 2] <- exp(-kappa * (n_dec - 1)) * pmax(strike - state, 0)
25      return(output)
26  }
27  ## Simulate paths to do regression on
28  n_path <- 10000 # Number of paths
29  start <- 36 # Starting state Z_0
30  path <- GBM(start, kappa, vol, n_dec, n_path, TRUE) # Generated paths
```

5.1. **Choice of basis functions.** It is well known that the quality of the LSM results depend on an appropriate choice of the regression basis. With this in mind, the *rlsm* package aims to allow users to specify any possible set of basis functions using a combination of the following

six objects. Not all objects are required but atleast one from Lines 31, 33, 34 or 35 must be supplied. Please keep in mind that some R functions from the package will have default values for these parameters.

LISTING 2. Regression basis

```
31  basis <- matrix(c(1, 1), nrow = 1)
32  btype <- "power" # currently either "power" or "laguerre"
33  intercept <- TRUE
34  knots <- matrix(c(30, 40, 50), nrow = 1)
35  BasisFunc <- function(state) { 1 / state }
36  n_rbasis <- 1
```

Suppose we are performing the regression at time t.

- The first object `basis` describes some transformation of the components of $Z_t = [Z_t^{(1)}, \ldots, Z_t^{(d)}]^T$. If `btype=''power''` and if entry $[i, j]$ is non-zero, then $(Z_t^{(i)})^j$ is included in the regression basis. If `btype=''laguerre''` and if entry $[i, j]$ is non-zero, then the j-th Laguerre polynomial of $Z_t^{(i)}$ is included in the regression basis. The object `basis` is processed row-wise.
- The object `intercept` decides whether a constant (1) is added to the regression basis.
- The object `knots` gives the location of the knots used for linear splines. If entry $[i, j]$ is given by B, then $(Z_t^{(i)} - B)^+$ is added to the basis. The object `knots` is processed row-wise.
- The object `BasisFunc` is a user defined function which which acts on an $n \times d$ matrix representing the $(Z_t(\omega_i))_{i=1}^d$ where entry $[i, j]$ gives $Z_t^{(j)}(\omega_i)$. This function should output a matrix to append to the design matrix horizontally on the right. The object `n_rbasis` gives the number of basis functions added by the `BasisFunc` function and must be supplied if `BasisFunc` is used.

The order in which the objects are processed is `basis`, `intercept`, `knots`, and `BasisFunc`. So in Listing 2, the regression basis is set to be $\{Z_t, Z_t^2, 1, (Z_t - 30)^+, (Z_t - 40)^+, (Z_t - 50)^+, 1/Z_t\}$ and in that order.

5.2. **Choice of regression.** Recall that the default method used in the lm function is QR factorization with pivoting. However, the default regression method used in the `LSM()` function is linear least squares using singular value decomposition (SVD) taking into account any rank deficiency in the design matrix. We do this for the following reasons. First, the least squares Monte Carlo method typically generate sample paths from a single point at the start. Therefore, as we perform regressions closer to $t = 0$, the design matrix is more likely to be rank deficient and so the SVD method is more stable than the QR approach. Secondly, when the number of rows in the design matrix is drastically larger than the number of columns, there is very little difference in the computational effort between SVD and QR. This is often the case for least squares Monte Carlo where the number of sample paths are

subtantially larger than the size of the regression basis. We point the reader to Section 3 in [6] for a detailed discussion.

LISTING 3. Regression

```
37  ## SVD
38  lsm1 <- LSM(path, Reward, Scrap, control, basis, intercept, btype, TRUE, knots,
        BasisFunc, n_rbasis)
39  ## QR factorization
40  RegFunc <- function(x, y, tt) {
41      out <- array(lm(y~ 0 + x)$coefficients)
42      out[is.na(out)] <- 0
43      return(out)
44  }
45  lsm2 <- LSM(path, Reward, Scrap, control, basis, intercept, btype, TRUE, knots,
        BasisFunc, n_rbasis, Reg = RegFunc)
```

In the above we test the default SVD regression approach with the QR approach from the base `lm()` function in R and we get the same results as shown below. However, in our experiments, adding $\{Z_t^3, Z_t^4\}$ to the regression basis causes SVD and QR to give different resuts. This is due to how they differ in the way they handle rank deficiency in the design matrix.

LISTING 4. Value estimates

```
46  > print(mean(lsm2$value[,2,1]))
47  [1] 4.468097
48  > print(mean(lsm1$value[,2,1]))
49  [1] 4.468097
```

In Listing 4, the `RegFunc` allows the user to specify any function from R to use in least squares Monte Carlo. The only condition is that it should return real valued coefficients for each of the feature functions in the regression basis. This is why we convert any `NA` values to 0 on Line 42. This is useful considering the large amount of statistical tools coded in R.

5.3. **Lower and upper bounds.** Let us finally demonstrate the construction of the upper and lower bounds. Line 54 extracts the prescribed policy using (2). Line 55 computes (5). Line 56 computes the lower and upper bound estimates in Section 4. Note that we use the `NestedGBM()` function from the *StochasticProcess* package to generate the nested simulation in (5) but the user is free to generate it in anyway they want to. The object `subsim` should be a 4 dimensional array where entry $[i, j, k, l]$ represents the j-th component of $f_l(W_l^{(i)}, Z_{l-1}(\omega_i))$. For the case that $j = 1$, we can represent it as a 3 dimensional array $[i, k, l]$ instead as done below.

LISTING 5. Lower and upper bounds

```
50  n_path2 <- 100
51  path2 <- GBM(start, kappa, vol, n_dec, n_path2, TRUE)
52  n_subsim <- 100 ## Number of nested simulations I
```

```
53  subsim <- NestedGBM(path2, kappa, vol, n_subsim, TRUE) # nested simulations
54  policy <- PathPolicy(path2, lsm1$expected, Reward, control, basis, btype, TRUE, knots,
        BasisFunc, n_rbasis) # Prescribed policy
55  mart <- AddDual(path2, subsim, lsm1$expected, Reward, Scrap, control, basis, btype,
        TRUE, knots, BasisFunc, n_rbasis) # varphi
56  bounds <- Bounds(path2, Reward, Scrap, control, mart, policy)
```

The below then generates the 99% confidence interval for the fair price of the option using the function approximations from LSM.

LISTING 6. 99% confidence intervals

```
57  > print(GetBounds(bounds, 0.01, 2))
58  [1] 4.361008 4.567014
```

6. FINAL THOUGHTS

Let us finally discuss the computational times. It takes around 0.35 cpu seconds for the `LSM()` on Line 38 to run and the same amount of time to compute both the lower and upper bounds on Lines 55 and 56. This package provides an easy way for users of least squares Monte Carlo to experiment with the large amount of statistical tools developed by the *R* community.

REFERENCES

[1] L. Andersen and M. Broadie, *Primal-dual simulation algorithm for pricing multidimensional american options*, Management Science **50** (2004), no. 9, 1222–1234.

[2] D. Belomestny, A. Kolodko, and J. Schoenmakers, *Regression methods for stochastic control problems and their convergence analysis*, SIAM Journal on Control and Optimization **48** (2010), no. 5, 3562–3588.

[3] D. Brown, J. Smith, and P. Sun, *Information relaxations and duality in stochastic dynamic programs*, Operations Research **58** (2010), no. 4, 785–801.

[4] J. Carriere, *Valuation of the early-exercise price for options using simulations and nonparametric regression*, Insurance: Mathematics and Economics **19** (1996), 19–30.

[5] E. Clement, D. Lamberton, and P. Protter, *An analysis of the Longstaff-Schwartz algorithm for American option pricing*, Finance and Stochastics **6** (2002), no. 4, 449–471.

[6] J. Demmel, *Applied numerical linear algebra*, Society for Industrial and Applied Mathematics, 1997.

[7] D. Eddelbuettel and R. Francois, *Rcpp: Seamless R and C++ integration*, Journal of Statistical Software **40** (2011), no. 8, 1–18.

[8] M. Haugh and L. Kogan, *Pricing American options: A duality approach*, Operations Research **52** (2004), no. 2, 258–270.

[9] F. Longstaff and E. Schwartz, *Valuing American options by simulation: a simple least-squares approach*, Review of Financial Studies **14** (2001), no. 1, 113–147.

[10] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, 2013, ISBN 3-900051-07-0.

[11] L. Rogers, *Pathwise stochastic optimal control*, SIAM J. Control Optimisation **46** (2007), no. 3, 1116–1132.

[12] S. Tompaidis and C. Yang, *Pricing american-style options by monte carlo simulation: alternatives to ordinary least squares*, Journal of computational finance **18** (2014), no. 1, 121–143.

[13] J. Tsitsiklis and B. Van Roy, *Optimal stopping of Markov processes: Hilbert space, theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives*, IEEE Transactions on Automatic Control **44** (1999), no. 10, 1840–1851.

[14] J. Yee, *rlsm: Least squres monte carlo*, 2017, Available at https://github.com/YeeJeremy/rlsm, R package version 1.0.

[15] _____, *StochasticProcess*, 2017, Available at https://github.com/YeeJeremy/StochasticProcess, R package version 1.0.