Package ‘rlsm’

November 4, 2017

Type Package

Title Least Squares Monte-Carlo

Version 1.0

Date 2016-07-10

Author Jeremy Yee

Maintainer Jeremy Yee <jeremyyee@outlook.com.au>

Description Least squares Monte Carlo and duality methods for Markov decision processes.

URL https://github.com/YeeJeremy/rlsm

License GPL

Depends StochasticProcess (>= 1.0)

Imports Rcpp (>= 0.11.6)

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

BugReports https://github.com/YeeJeremy/rlsm/issues

R topics documented:

`AddDual` *AddDual*

Description

Compute additive duals.

Usage

```
AddDual(path, subsim, expected, Reward, Scrap, control, basis,
        basis_type, spline = FALSE, knots = matrix(NA, nrow = 1))
```

Arguments

`path` 3-D array representing sample paths. Entry [i,,t] represents the state at time t for sample path i.

`subsim` 4-D array containing subsimulations. Entry [i,,p,t] is for subsim i on path p at time t.

`expected` 3-D array representing the fitted coefficients for the continuation value function. Array [,p,t] gives the fit for position p at time t.

`Reward` User supplied function to represent the reward function. The function should take in the following arguments, in this order:

- $n \times d$ matrix representing the n d-dimensional states.
- A natural number representing the decison epoch.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a, p]$-th entry corresponds to the reward from applying the a-th action to the p-th position for the i-th state.

`Scrap` User supplied function to represent the scrap function. The function should take in the following argument:

- $n \times d$ matrix representing the n d-dimensional states.

`control` Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:

- Matrix of dimension n_pos $\times$ n_action, where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions n_pos $\times$ n_action $\times$ n_pos, where entry [i,j,k] is the probability of moving to position k after applying action j to position i.

`basis` Matrix specifying the regression basis. Zeros and ones.

`basis_type` The type of basis functions to use: "power" and "laguerre".

`spline` Logical value indicating whether linear splines should be used.

`knots` Matrix indicating the location of the knots. If none, use NA.

Value

3-D array containing the additive duals. Entry [i, j, t] is for path i and position j at time t.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
step <- 0.02
mu <- 0.06 * step
vol <- 0.2 * sqrt(step)
n_dec <- 51
start <- 36
strike <- 40
## LSM
n_path <- 1000
path <- GBM(start, mu, vol, n_dec, n_path, TRUE)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
basis <- matrix(c(1, 1), nrow = 1)
knots <- matrix(c(30, 40, 50), nrow = 1)
Scrap <- function(state) {
    output <- matrix(data = 0, nrow = nrow(state), ncol = 2)
    output[, 2] <- exp(-mu * (n_dec - 1)) * pmax(strike - state, 0)
    return(output)
}
Reward <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[, 2, 2] <- exp(-mu * (time - 1)) * pmax(strike - state, 0)
    return(output)
}
lsm <- LSM(path, Reward, Scrap, control, basis, TRUE, "power", TRUE, knots)
n_path2 <- 100
path2 <- GBM(start, mu, vol, n_dec, n_path2, TRUE)
policy <- PathPolicy(path2, lsm$expected, Reward, control, basis,
"power", TRUE, knots)
n_subsim <- 100
subsim <- NestedGBM(path2, mu, vol, n_subsim, TRUE)
mart <- AddDual(path2, subsim, lsm$expected, Reward, Scrap, control, basis, "power", TRUE, knots)
```

Bounds *Bounds*

Description

Compute bound estimates using additive duals.

Usage

```
Bounds(path, Reward, Scrap, control, mart, path_action)
```

Arguments

`path` 3-D array representing sample paths. Entry [i,,t] represents the state at time t for sample path i.

`Reward` User supplied function to represent the reward function. The function should take in the following arguments, in this order:

- $n \times d$ matrix representing the n d-dimensional states.
- A natural number representing the decison epoch.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a, p]$-th entry corresponds to the reward from applying the a-th action to the p-th position for the i-th state.

`Scrap` User supplied function to represent the scrap function. The function should take in the following argument:

- $n \times d$ matrix representing the n d-dimensional states.

`control` Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:

- Matrix of dimension n_pos $\times$ n_action, where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions n_pos $\times$ n_action $\times$ n_pos, where entry [i,j,k] is the probability of moving to position k after applying action j to position i.

`mart` 3-D array containing the additive duals. Entry [i, j, t] is for path i and position j at time t.

`path_action` 3-D array containing the prescribed policy. Entry [i,p,t] is for path i and position p at time t.

Value

`primal` 3-D array containing the lower bound estimates. Entry [i,p,t] is for path i and position p at time t.

`dual` 3-D array containing the lower bound estimates. Entry [i,p,t] is for path i and position p at time t.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
step <- 0.02
mu <- 0.06 * step
vol <- 0.2 * sqrt(step)
n_dec <- 51
start <- 36
strike <- 40
## LSM
n_path <- 1000
path <- GBM(start, mu, vol, n_dec, n_path, TRUE)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
basis <- matrix(c(1, 1), nrow = 1)
knots <- matrix(c(30, 40, 50), nrow = 1)
Scrap <- function(state) {
    output <- matrix(data = 0, nrow = nrow(state), ncol = 2)
    output[, 2] <- exp(-mu * (n_dec - 1)) * pmax(strike - state, 0)
    return(output)
}
Reward <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[, 2, 2] <- exp(-mu * (time - 1)) * pmax(strike - state, 0)
    return(output)
}
lsm <- LSM(path, Reward, Scrap, control, basis, TRUE, "power", TRUE, knots)
n_path2 <- 100
path2 <- GBM(start, mu, vol, n_dec, n_path2, TRUE)
policy <- PathPolicy(path2, lsm$expected, Reward, control, basis,
"power", TRUE, knots)
n_subsim <- 100
subsim <- NestedGBM(path2, mu, vol, n_subsim, TRUE)
mart <- AddDual(path2, subsim, lsm$expected, Reward, Scrap, control,
basis, "power", TRUE, knots)
bounds <- Bounds(path2, Reward, Scrap, control, mart, policy)
```

FullTestPolicy *FullTestPolicy*

Description

Full testing of prescribed policy for sample paths.

Usage

```
FullTestPolicy(start_position, path, control, Reward, Scrap, path_action)
```

Arguments

`start_position` Starting position.

path — 3-D array representing sample paths. Entry [i,,t] represents the state at time t for sample path i.

control — Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:

- Matrix of dimension n_pos $\times$ n_action, where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions n_pos $\times$ n_action $\times$ n_pos, where entry [i,j,k] is the probability of moving to position k after applying action j to position i.

Reward — User supplied function to represent the reward function. The function should take in the following arguments, in this order:

- $n \times d$ matrix representing the n d-dimensional states.
- A natural number representing the decison epoch.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a, p]$-th entry corresponds to the reward from applying the a-th action to the p-th position for the i-th state.

Scrap — User supplied function to represent the scrap function. The function should take in the following argument:

- $n \times d$ matrix representing the n d-dimensional states.

path_action — 3-D array containing the prescribed policy. Entry [i,p,t] is for path i and position p at time t.

Value

value — Array containing the path values.

position — Matrix containing the evolution of the position. Entry[i,t] refers to the position at time t for sample path i.

action — Matrix containing the actions taken. Entry[i,t] refers to the action at time t for sample path i.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
step <- 0.02
mu <- 0.06 * step
vol <- 0.2 * sqrt(step)
n_dec <- 51
start <- 36
strike <- 40
## LSM
```

```
n_path <- 1000
path <- GBM(start, mu, vol, n_dec, n_path, TRUE)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
basis <- matrix(c(1, 1), nrow = 1)
knots <- matrix(c(30, 40, 50), nrow = 1)
Scrap <- function(state) {
    output <- matrix(data = 0, nrow = nrow(state), ncol = 2)
    output[, 2] <- exp(-mu * (n_dec - 1)) * pmax(strike - state, 0)
    return(output)
}
Reward <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[, 2, 2] <- exp(-mu * (time - 1)) * pmax(strike - state, 0)
    return(output)
}
lsm <- LSM(path, Reward, Scrap, control, basis, TRUE, "power", TRUE, knots)
n_path2 <- 1000
path2 <- GBM(start, mu, vol, n_dec, n_path2, TRUE)
policy <- PathPolicy(path2, lsm$expected, Reward, control, basis,
"power", TRUE, knots)
test <- FullTestPolicy(2, path, control, Reward, Scrap, policy)
```

`GetBounds` *Confidence Bounds*

Description

Confidence bounds for the value.

Usage

```
GetBounds(duality, alpha, position)
```

Arguments

`duality`	Object returned by the Bounds function.
`alpha`	Specifies the (1-alpha) confidence bounds.
`position`	Natural number indicating the starting position.

Value

Array representing the (1-alpha) confidence bounds for the value of the specified position.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
step <- 0.02
mu <- 0.06 * step
vol <- 0.2 * sqrt(step)
n_dec <- 51
start <- 36
strike <- 40
## LSM
n_path <- 1000
path <- GBM(start, mu, vol, n_dec, n_path, TRUE)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
basis <- matrix(c(1, 1), nrow = 1)
knots <- matrix(c(30, 40, 50), nrow = 1)
Scrap <- function(state) {
    output <- matrix(data = 0, nrow = nrow(state), ncol = 2)
    output[, 2] <- exp(-mu * (n_dec - 1)) * pmax(strike - state, 0)
    return(output)
}
Reward <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[, 2, 2] <- exp(-mu * (time - 1)) * pmax(strike - state, 0)
    return(output)
}
lsm <- LSM(path, Reward, Scrap, control, basis, TRUE, "power", TRUE, knots)
n_path2 <- 100
path2 <- GBM(start, mu, vol, n_dec, n_path2, TRUE)
policy <- PathPolicy(path2, lsm$expected, Reward, control, basis,
"power", TRUE, knots)
n_subsim <- 100
subsim <- NestedGBM(path2, mu, vol, n_subsim, TRUE)
mart <- AddDual(path2, subsim, lsm$expected, Reward, Scrap, control,
basis, "power", TRUE, knots)
bounds <- Bounds(path2, Reward, Scrap, control, mart, policy)
confidenceInterval <-GetBounds(bounds, 0.05, 2)
```

LSM *Least squares Monte Carlo*

Description

Perform the least squares Monte Carlo algorithm.

Usage

```
LSM(path, Reward, Scrap, control, basis, intercept, basis_type,
    spline = FALSE, knots = matrix(NA, nrow = 1))
```

Arguments

`path` 3-D array representing sample paths. Entry [i,,j] represents the state at time j for sample path i.

`Reward` User supplied function to represent the reward function. The function should take in the following arguments, in this order:

- $n \times d$ matrix representing the n d-dimensional states.
- A natural number representing the decison epoch.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a, p]$-th entry corresponds to the reward from applying the a-th action to the p-th position for the i-th state.

`Scrap` User supplied function to represent the scrap function. The function should take in the following argument:

- $n \times d$ matrix representing the n d-dimensional states.

`control` Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:

- Matrix of dimension n_pos $\times$ n_action, where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions n_pos $\times$ n_action $\times$ n_pos, where entry [i,j,k] is the probability of moving to position k after applying action j to position i.

`basis` Matrix specifying the regression basis. Zeros and ones.

`intercept` Logical value indicating whether the intercept should be included.

`basis_type` The type of basis functions to use: "power" and "laguerre".

`spline` Logical value indicating whether linear splines should be used.

`knots` Matrix indicating the location of the knots. If none, use NA.

Value

`value` 3-D array containing the path values. Entry [i,p,t] is for path i and position p at time t.

`expected` 3-D array representing the fitted coefficients for the continuation value function. Array [,p,t] gives the fit for position p at time t.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
step <- 0.02
mu <- 0.06 * step
```

```
vol <- 0.2 * sqrt(step)
n_dec <- 51
start <- 36
strike <- 40
## LSM
n_path <- 1000
path <- GBM(start, mu, vol, n_dec, n_path, TRUE)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
basis <- matrix(c(1, 1), nrow = 1)
knots <- matrix(c(30, 40, 50), nrow = 1)
Scrap <- function(state) {
    output <- matrix(data = 0, nrow = nrow(state), ncol = 2)
    output[, 2] <- exp(-mu * (n_dec - 1)) * pmax(strike - state, 0)
    return(output)
}
Reward <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[, 2, 2] <- exp(-mu * (time - 1)) * pmax(strike - state, 0)
    return(output)
}
lsm <- LSM(path, Reward, Scrap, control, basis, TRUE, "power", TRUE, knots)
```

PathPolicy *PathPolicy*

Description

Obtaining the prescribed policy for sample paths

Usage

```
PathPolicy(path, expected, Reward, control, basis, intercept, basis_type,
    spline = FALSE, knots = matrix(NA, nrow = 1))
```

Arguments

path — 3-D array representing sample paths. Entry [i,,t] represents the state at time t for sample path i.

expected — 3-D array representing the fitted coefficients for the continuation value function. Array [,p,t] gives the fit for position p at time t.

Reward — User supplied function to represent the reward function. The function should take in the following arguments, in this order:

- $n \times d$ matrix representing the n d-dimensional states.
- A natural number representing the decison epoch.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a, p]$-th entry corresponds to the reward from applying the a-th action to the p-th position for the i-th state.

`control` Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:

- Matrix of dimension n_pos $\times$ n_action, where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions n_pos $\times$ n_action $\times$ n_pos, where entry [i,j,k] is the probability of moving to position k after applying action j to position i.

`basis` Matrix specifying the regression basis. Zeros and ones.

`intercept` Logical value indicating whether the intercept should be included.

`basis_type` The type of basis functions to use: "power" and "laguerre".

`spline` Logical value indicating whether linear splines should be used.

`knots` Matrix indicating the location of the knots. If none, use NA.

Value

3-D array containing the prescribed policy. Entry [i,p,t] is for path i and position p at time t.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
step <- 0.02
mu <- 0.06 * step
vol <- 0.2 * sqrt(step)
n_dec <- 51
start <- 36
strike <- 40
## LSM
n_path <- 1000
path <- GBM(start, mu, vol, n_dec, n_path, TRUE)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
basis <- matrix(c(1, 1), nrow = 1)
knots <- matrix(c(30, 40, 50), nrow = 1)
Scrap <- function(state) {
    output <- matrix(data = 0, nrow = nrow(state), ncol = 2)
    output[, 2] <- exp(-mu * (n_dec - 1)) * pmax(strike - state, 0)
    return(output)
}
Reward <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[, 2, 2] <- exp(-mu * (time - 1)) * pmax(strike - state, 0)
    return(output)
```

```
}
lsm <- LSM(path, Reward, Scrap, control, basis, TRUE, "power", TRUE, knots)
n_path2 <- 1000
path2 <- GBM(start, mu, vol, n_dec, n_path2, TRUE)
policy <- PathPolicy(path2, lsm$expected, Reward, control, basis, "power", TRUE, knots)
```

TestPolicy *TestPolicy*

Description

Testing prescribed policy for sample paths.

Usage

```
TestPolicy(start_position, path, control, Reward, Scrap, path_action)
```

Arguments

`start_position` Starting position.

`path` 3-D array representing sample paths. Entry [i,,t] represents the state at time t for sample path i.

`control` Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:

- Matrix of dimension n_pos $\times$ n_action, where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions n_pos $\times$ n_action $\times$ n_pos, where entry [i,j,k] is the probability of moving to position k after applying action j to position i.

`Reward` User supplied function to represent the reward function. The function should take in the following arguments, in this order:

- $n \times d$ matrix representing the n d-dimensional states.
- A natural number representing the decison epoch.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a, p]$-th entry corresponds to the reward from applying the a-th action to the p-th position for the i-th state.

`Scrap` User supplied function to represent the scrap function. The function should take in the following argument:

- $n \times d$ matrix representing the n d-dimensional states.

`path_action` 3-D array containing the prescribed policy. Entry [i,p,t] is for path i and position p at time t.

Value

Array containing the values for each path.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
step <- 0.02
mu <- 0.06 * step
vol <- 0.2 * sqrt(step)
n_dec <- 51
start <- 36
strike <- 40
## LSM
n_path <- 1000
path <- GBM(start, mu, vol, n_dec, n_path, TRUE)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
basis <- matrix(c(1, 1), nrow = 1)
knots <- matrix(c(30, 40, 50), nrow = 1)
Scrap <- function(state) {
    output <- matrix(data = 0, nrow = nrow(state), ncol = 2)
    output[, 2] <- exp(-mu * (n_dec - 1)) * pmax(strike - state, 0)
    return(output)
}
Reward <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[, 2, 2] <- exp(-mu * (time - 1)) * pmax(strike - state, 0)
    return(output)
}
lsm <- LSM(path, Reward, Scrap, control, basis, TRUE, "power", TRUE, knots)
n_path2 <- 1000
path2 <- GBM(start, mu, vol, n_dec, n_path2, TRUE)
policy <- PathPolicy(path2, lsm$expected, Reward, control, basis,
"power", TRUE, knots)
test <- TestPolicy(2, path, control, Reward, Scrap, policy)
```

Index