Package ‘rlsm’

December 18, 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

Suggests StochasticProcess (>= 1.0)

Imports Rcpp (>= 0.11.6)

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

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

R topics documented:

AddDual 2
Bounds 4
FullTestPolicy 6
GetBounds 8
LSM 9
PathPolicy 11
TestPolicy 13

Index **16**

AddDual *AddDual*

Description

Compute additive duals.

Usage

```
AddDual(path, subsim, expected, Reward, Scrap, control, basis =
matrix(c(1), nrow = 1), basis_type = "power", spline = FALSE, knots =
matrix(NA, nrow = 1), Basis = function(){}, n_rbasis = 0)
```

Arguments

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

`subsim` 4-D array containing subsimulations. Entry [i,,j,k] is for nested simulation i on path j at time k.

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

`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 time.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards where n is the number of sample paths, a is the number of action, and p is the number of positions. The $[i, j, k]$-th entry corresponds to the reward from applying the j-th action to the k-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.

The function should output the following:

- Matrix with dimensions $n \times p$ representing the scrap where n is the number of sample paths and p is the number of positions. The $[i, j]$-th entry corresponds to the scrap at position j for the i-th path.

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

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

basis	Logical matrix describing some transformation of the components of the state. If *btype=="power"* and if entry [i,j] is non-zero, then j-th power of the i-th component of the state is included in the regression basis. If *btype=="laguerre"* and if entry $[i,j]$ is non-zero, then the j-th Laguerre polynomial of i-th component of the state is included in the regression basis. The object $basis$ is processed row-wise.
basis_type	The type of tranformation to use for *basis*: "power" and "laguerre".
spline	Logical value indicating whether linear splines should be used.
knots	Real valued matrix indicating the location of the knots for the linear splines. If entry [i,j] gives value x, then a knot at x is used for the j-th component of the state. If there is no knot, use **NA** for matrix entry. For each row, the numbers should be placed before the **NA** values.
Basis	User supplied function to represent other basis functions. The function should take in the following argument: • $n \times d$ matrix representing the n d-dimensional states. The function should output the following: • Matrix with dimensions $n \times n_r basis$ representing the matrix to append to the design matrix horizontally on the right.
n_rbasis	The number of basis functions added by the *Basis* function above. Must be used if *Basis* is given.

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

```
library(StochasticProcess)
## 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,j,k] represents the j-th component of the state at time k 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 time.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards where n is the number of sample paths, a is the number of action, and p is the number of positions. The $[i, j, k]$-th entry corresponds to the reward from applying the j-th action to the k-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.

The function should output the following:

- Matrix with dimensions $n \times p$ representing the scrap where n is the number of sample paths and p is the number of positions. The $[i, j]$-th entry corresponds to the scrap at position j for the i-th path.

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

- Matrix of dimension $p \times a$ where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions $p \times a \times p$ 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, k] is for path i and position j at time k.

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

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

```
library(StochasticProcess)
## 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,,j] represents the state at time j for sample path i.

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

- Matrix of dimension $p \times a$ where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions $p \times a \times p$ 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 time.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards where n is the number of sample paths, a is the number of action, and p is the number of positions. The $[i, j, k]$-th entry corresponds to the reward from applying the j-th action to the k-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.

The function should output the following:

- Matrix with dimensions $n \times p$ representing the scrap where n is the number of sample paths and p is the number of positions. The $[i, j]$-th entry corresponds to the scrap at position j for the i-th path.

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

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

```
library(StochasticProcess)
## 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

```
library(StochasticProcess)
## 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 = matrix(c(1), nrow = 1),
    intercept = TRUE, basis_type = "power", spline = FALSE, knots =
    matrix(NA, nrow = 1), Basis = function(){}, n_rbasis = 0, Reg)
```

Arguments

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

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards where n is the number of sample paths, a is the number of action, and p is the number of positions. The $[i, j, k]$-th entry corresponds to the reward from applying the j-th action to the k-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.

The function should output the following:

- Matrix with dimensions $n \times p$ representing the scrap where n is the number of sample paths and p is the number of positions. The $[i, j]$-th entry corresponds to the scrap at position j for the i-th path.

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

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

`basis` Logical matrix describing some transformation of the components of the state. If *btype=="power"* and if entry [i,j] is non-zero, then j-th power of the i-th component of the state is included in the regression basis. If *btype=="laguerre"* and if entry $[i,j]$ is non-zero, then the j-th Laguerre polynomial of i-th component of the state is included in the regression basis. The object $basis$ is processed row-wise.

`intercept` Logical value indicating whether a constant 1 is included in regression basis

`basis_type` The type of tranformation to use for *basis*: "power" and "laguerre".

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

`knots` Real valued matrix indicating the location of the knots for the linear splines. If entry [i,j] gives value x, then a knot at x is used for the j-th component of the state. If there is no knot, use **NA** for matrix entry. For each row, the numbers should be placed before the **NA** values.

`Basis` User supplied function to represent other basis functions. The function should take in the following argument:

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

The function should output the following:

- Matrix with dimensions $n \times n_r basis$ representing the matrix to append to the design matrix horizontally on the right.

`n_rbasis` The number of basis functions added by the *Basis* function above. Must be used if *Basis* is given.

`Reg` User defined regression method. Not needed unless user doesn't want to use SVD.

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

```
library(StochasticProcess)
## 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 = matrix(c(1), nrow =
1), basis_type = "power", spline = FALSE, knots = matrix(NA, nrow = 1),
Basis = function(){}, n_rbasis = 0)
```

Arguments

`path`	3-D array representing sample paths. Entry [i,j,k] represents the j-th component of the state at time k for sample path i.
`expected`	3-D array representing the fitted coefficients for the continuation value function. Array [,i,j] gives the fit for position i at time j.
`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 time.

The function should output the following:

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

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

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

`basis` Logical matrix describing some transformation of the components of the state. If *btype=="power"* and if entry [i,j] is non-zero, then j-th power of the i-th component of the state is included in the regression basis. If *btype=="laguerre"* and if entry $[i,j]$ is non-zero, then the j-th Laguerre polynomial of i-th component of the state is included in the regression basis. The object $basis$ is processed row-wise.

`basis_type` The type of tranformation to use for *basis*: "power" and "laguerre".

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

`knots` Real valued matrix indicating the location of the knots for the linear splines. If entry [i,j] gives value x, then a knot at x is used for the j-th component of the state. If there is no knot, use **NA** for matrix entry. For each row, the numbers should be placed before the **NA** values.

`Basis` User supplied function to represent other basis functions. The function should take in the following argument:

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

The function should output the following:

- Matrix with dimensions $n \times n_r basis$ representing the matrix to append to the design matrix horizontally on the right.

`n_rbasis` The number of basis functions added by the *Basis* function above. Must be used if *Basis* is given.

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

```
library(StochasticProcess)
## 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,,j] represents the state at time j for sample path i.
`control`	Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:

- Matrix of dimension $p \times a$ where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions $p \times a \times p$ 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 time.

The function should output the following:

- 3-D array with dimensions $n \times (a \times p)$ representing the rewards where n is the number of sample paths, a is the number of action, and p is the number of positions. The $[i, j, k]$-th entry corresponds to the reward from applying the j-th action to the k-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.

The function should output the following:

- Matrix with dimensions $n \times p$ representing the scrap where n is the number of sample paths and p is the number of positions. The $[i, j]$-th entry corresponds to the scrap at position j for the i-th path.

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

Value

Array containing the values for each path.

Author(s)

Jeremy Yee

Examples

```
library(StochasticProcess)
## 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

AddDual, 2

Bounds, 4

FullTestPolicy, 6

GetBounds, 8

LSM, 9

PathPolicy, 11

TestPolicy, 13