

Package ‘rcss’

May 27, 2017

Type Package

Title Convex Switching Systems

Version 1.4

Date 2017-05-27

Author Juri Hinz and Jeremy Yee

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

Description The numerical treatment of optimal switching problems in a finite time setting when the state evolves as a controlled Markov chain consisting of a uncontrolled continuous component following linear dynamics and a controlled Markov chain taking values in a finite set. The reward functions are assumed to be convex and Lipschitz continuous in the continuous state. The action set is finite.

URL <https://github.com/YeeJeremy/rcss>

License GPL

Depends rflann (>= 1.0)

Imports Rcpp (>= 0.11.6)

LinkingTo Rcpp, RcppArmadillo

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation yes

BugReports <https://github.com/YeeJeremy/rcss/issues>

R topics documented:

Bellman	2
BellmanAccelerated	3
Duality	5
Expected	6
ExpectedAccelerated	7
FastBellman	8
FastExpected	10

FastMartingale	11
FiniteMartingale	13
GetBounds	14
Martingale	16
Path	17
PathPolicy	18
StochasticGrid	20
TestPolicy	21
TestPolicy2	22
WrongMartingale	24
Index	26

Bellman

Bellman Recursion

Description

Approximate the value functions using the Bellman recursion.

Usage

```
Bellman(grid, reward, control, disturb, weight)
```

Arguments

grid	Matrix representing the grid, whose i-th row matrix [i,] corresponds to i-th point of the grid. The matrix [i,1] equals to 1 while the vector [i,-1] represents the system state.
reward	5-dimensional array representing the subgradient envelope of the reward function. The matrix [i,,a,p,t] captures the subgradient at grid point i for action a taken in position p at time t, with the intercept given by [i,1,a,p,t] and slope by [i,-1,a,p,t].
control	Array representing the transition probabilities of the controlled Markov chain. Two possible inputs: <ul style="list-style-type: none"> • Matrix of dimension $n_pos \times n_action$, where entry [i,j] describes the next position after selecting action j at position i. • 3-dimensional 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.
disturb	3-dimensional array containing the disturbance matrices. Matrix [,i] specifies the i-th disturbance matrix.
weight	Array containing the probability weights of the disturbance matrices.

Value

value	4-dimensional array representing the subgradient envelope of the value function, where the intercept $[i,1,p,t]$ and slope matrix $[i,-1,p,t]$ describes a subgradient of the value function at grid point i for position p at time t .
expected	4-dimensional array representing the expected value functions. Same format as the value array.
action	3-dimensional array representing the prescribed policy, where entry $[i,j,k]$ is the decision rule at grid point i for position j at time k .

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1, 1, ] <- 1
disturb[2, 2, ] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 2), c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(91, 2, 2, 2, 51))
reward[grid[, 2] <= 40, 1, 2, 2, ] <- 40
reward[grid[, 2] <= 40, 2, 2, 2, ] <- -1
bellman <- Bellman(grid, reward, control, disturb, weight)
```

BellmanAccelerated *Bellman Recursion Accelerated With K Nearest Neighbours*

Description

Approximate the value functions using the Bellman recursion and k nearest neighbours.

Usage

```
BellmanAccelerated(grid, reward, control, disturb, weight, k, Neighbour)
```

Arguments

grid	Matrix representing the grid, whose i -th row matrix $[i,]$ corresponds to i -th point of the grid. The matrix $[i,1]$ equals to 1 while the vector $[i,-1]$ represents the system state.
reward	5-dimensional array representing the subgradient envelope of the reward function. The matrix $[i,,a,p,t]$ captures the subgradient at grid point i for action a taken in position p at time t , with the intercept given by $[i,1,a,p,t]$ and slope by $[i,-1,a,p,t]$.

control	<p>Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:</p> <ul style="list-style-type: none"> • Matrix of dimension $n_pos \times n_action$, where entry $[i,j]$ describes the next position after selecting action j at position i. • 3-dimensional 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.
disturb	3-dimensional array containing the disturbance matrices. Matrix $[.,i]$ specifies the i -th disturbance matrix.
weight	Array containing the probability weights of the disturbance matrices.
k	The number of nearest neighbours used for each grid point. Must be greater than 1.
Neighbour	Optional function to find the nearest neighbours. If not provided, the Neighbour function from the rflann package is used instead.

Value

value	4-dimensional array representing the subgradient envelope of the value function, where the intercept $[i,1,p,t]$ and slope matrix $[i,-1,p,t]$ describes a subgradient of the value function at grid point i for position p at time t .
expected	4-dimensional array representing the expected value functions. Same format as the value array.
action	3-dimensional array representing the prescribed policy, where entry $[i,j,k]$ is the decision rule at grid point i for position j at time k .

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1, 1, ] <- 1
disturb[2, 2, ] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 2), c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(91, 2, 2, 2, 51))
reward[grid[, 2] <= 40, 1, 2, 2, ] <- 40
reward[grid[, 2] <= 40, 2, 2, 2, ] <- -1
bellman <- BellmanAccelerated(grid, reward, control, disturb, weight, 2)
```

Duality

*Primal And Dual Values***Description**

Calculate the primal and dual values.

Usage

```
Duality(path, control, Reward, mart, path_action)
```

Arguments

path	3-dimensional array representing the generated paths. Array [i,j] represents the state at time i for sample path j.
control	<p>Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:</p> <ul style="list-style-type: none"> Matrix of dimension $n_pos \times n_action$, where entry [i,j] describes the next position after selecting action j at position i. 3-dimensional 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	<p>User supplied function to represent the reward function. The function should take in the following arguments, in this order:</p> <ul style="list-style-type: none"> $n \times d$ matrix representing the n d-dimensional states. A natural number representing the decision epoch. <p>The function should output the following:</p> <ul style="list-style-type: none"> $n \times (p \times a)$ matrix representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a \times (j - 1) + k]$-th entry corresponds to the reward from applying the k-th action to the j-th position for the i-th state.
mart	3 or 4-dimensional array representing the martingale increments.
path_action	3-dimensional array representing the prescribed policy for the sample paths. Entry [i,j,k] captures the prescribed action at time i for position j on sample path k.

Value

List containing the following 2 arrays:

primal	3-dimensional array representing the primal values, where entry [i,j,k] represents the value at time i for position j on sample path k.
dual	3-dimensional array representing the dual values. Same format as above.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
disturb_weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, disturb_weight, r_index)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
path_nn <- Neighbour(matrix(path, ncol = 2), grid, 1, "kdtree", 0, 1)$indices
subsim_disturb <- array(0, dim = c(2, 2, 20, 100, 50))
subsim_disturb[1,1,,,] <- 1
subsim_disturb[2,2,,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(100000))
subsim_weight <- rep(1 / 20, 20)
mart <- FastMartingale(bellman$value, path, path_nn, subsim_disturb,
  subsim_weight, grid, control = control)
RewardFunc <- function(state, time) {
  output <- array(data = 0, dim = c(nrow(state), 4))
  output[,4] <- pmax(40 - state[,2], 0)
  return(output)
}
path_action <- PathPolicy(path, path_nn, control, RewardFunc, bellman$expected)
duality <- Duality(path, control, RewardFunc, mart, path_action)
```

Expected

*Expected Value Function***Description**

Approximate the expected value functions.

Usage

```
Expected(grid, value, disturb, weight)
```

Arguments

grid	Matrix representing the grid, whose i-th row matrix [i,] corresponds to i-th point of the grid. The matrix [i,1] equals to 1 while the vector [i,-1] represents the system state.
value	Matrix representing the subgradient envelope of the future value function, where the intercept [i,1] and slope matrix [i,-1] describes a subgradient at grid point i.
disturb	3-dimensional array containing the disturbance matrices. Matrix [,i] specifies the i-th disturbance matrix.
weight	Array containing the probability weights of the disturbance matrices.

Value

Matrix representing the subgradient envelope of the expected value function. Same format as the value input.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91)))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1, 1,] <- 1
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 2), c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(91, 2, 2, 2, 51))
reward[grid[, 2] <= 40, 1, 2, 2,] <- 40
reward[grid[, 2] <= 40, 2, 2, 2,] <- -1
bellman <- Bellman(grid, reward, control, disturb, weight)
expected <- Expected(grid, bellman$value[, , 2, 2], disturb, weight)
```

ExpectedAccelerated *Expected Value Function Using K Nearest Neighbours*

Description

Approximate the expected value function using k nearest neighbours.

Usage

```
ExpectedAccelerated(grid, value, disturb, weight, k, Neighbour)
```

Arguments

grid	Matrix representing the grid, whose i-th row matrix [i,] corresponds to i-th point of the grid. The matrix [i,1] equals to 1 while the vector [i,-1] represents the system state.
value	Matrix representing the subgradient envelope of the future value function, where the intercept [i,1] and slope matrix [i,-1] describes a subgradient at grid point i.
disturb	3-dimensional array containing the disturbance matrices. Matrix [,i] specifies the i-th disturbance matrix.
weight	Array containing the probability weights of the disturbance matrices.
k	The number of nearest neighbours for each grid point. Must be greater than 1.
Neighbour	Optional function to find the nearest neighbours. If not provided, the Neighbour function from the rflann package is used instead.

Value

Matrix representing the subgradient envelope of the expected value function. Same format as the value input.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1, 1, ] <- 1
disturb[2, 2, ] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 2), c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(91, 2, 2, 2, 51))
reward[grid[, 2] <= 40, 1, 2, 2, ] <- 40
reward[grid[, 2] <= 40, 2, 2, 2, ] <- -1
bellman <- BellmanAccelerated(grid, reward, control, disturb, weight, 2)
expected <- ExpectedAccelerated(grid, bellman$value[,2,2], disturb, weight, 2)
```

FastBellman

Fast Bellman Recursion

Description

Approximate the value functions using the Bellman recursion and fast methods

Usage

```
FastBellman(grid, reward, control, disturb, weight, r_index, Neighbour,
             smooth = 1, SmoothNeighbour)
```


Arguments

grid	Matrix representing the grid, whose i -th row matrix $[i, \cdot]$ corresponds to i -th point of the grid. The matrix $[i, 1]$ equals to 1 while the vector $[i, -1]$ represents the system state.
reward	5-dimensional array representing the subgradient envelope of the reward function. The matrix $[i, \cdot, a, p, t]$ captures the subgradient at grid point i for action a taken in position p at time t , with the intercept given by $[i, 1, a, p, t]$ and slope by $[i, -1, a, p, t]$.
control	Array representing the transition probabilities of the controlled Markov chain. Two possible inputs: <ul style="list-style-type: none"> • Matrix of dimension $n_pos \times n_action$, where entry $[i, j]$ describes the next position after selecting action j at position i. • 3-dimensional 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.
disturb	3-dimensional array containing the disturbance matrices. Matrix $[:, i]$ specifies the i -th disturbance matrix.
weight	Array containing the probability weights of the disturbance matrices.
r_index	Matrix representing the positions of random entries in the disturbance matrix, where entry $[i, 1]$ is the row number and $[i, 2]$ gives the column number of the i -th random entry.
Neighbour	Optional function to find the nearest neighbours. If not provided, the Neighbour function from the rflann package is used instead.
smooth	The number of nearest neighbours used to smooth the expected value functions during the Bellman recursion.
SmoothNeighbour	Optional function to find the nearest neighbours for smoothing purposes. If not provided, the Neighbour function from the rflann package is used instead.

Value

value	4-dimensional array representing the subgradient envelope of the value function, where the intercept $[i, 1, p, t]$ and slope matrix $[i, -1, p, t]$ describes a subgradient of the value function at grid point i for position p at time t .
expected	4-dimensional array representing the expected value functions. Same format as the value array.
action	3-dimensional array representing the prescribed policy, where entry $[i, j, k]$ is the decision rule at grid point i for position j at time k .

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 40,1,2,2,] <- 40
reward[grid[,2] <= 40,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, weight, r_index)
```

FastExpected

Fast Expected Value Function

Description

Approximate the expected value function using fast methods.

Usage

```
FastExpected(grid, value, disturb, weight, r_index, Neighbour,
             smooth = 1, SmoothNeighbour)
```

Arguments

grid	Matrix representing the grid, whose i-th row matrix [i,] corresponds to i-th point of the grid. The matrix [i,1] equals to 1 while the vector [i,-1] represents the system state.
value	Matrix representing the subgradient envelope of the future value function, where the intercept [i,1] and slope matrix [i,-1] describes a subgradient at grid point i.
disturb	3-dimensional array containing the disturbance matrices. Matrix [,i] specifies the i-th disturbance matrix.
weight	Array containing the probability weights of the disturbance matrices.
r_index	Matrix representing the positions of random entries in the disturbance matrix, where entry [i,1] is the row number and [i,2] gives the column number of the i-th random entry.
Neighbour	Optional function to find the nearest neighbours. If not provided, the Neighbour function from the rflann package is used instead.
smooth	The number of nearest neighbours used to smooth the expected value functions during the Bellman recursion.
SmoothNeighbour	Optional function to find the nearest neighbours for smoothing purposes. If not provided, the Neighbour function from the rflann package is used instead.

Value

Matrix representing the subgradient envelope of the expected value function. Same format as the value input.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 40,1,2,2,] <- 40
reward[grid[,2] <= 40,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, weight, r_index)
expected <- FastExpected(grid, bellman$value[,2,2], disturb, weight, r_index)
```

FastMartingale

Fast Martingale

Description

Compute the martingale increments using fast methods.

Usage

```
FastMartingale(value, path, path_nn, disturb, weight, grid, Neighbour,
               control)
```

Arguments

value	4-dimensional array representing the subgradient envelope of the value function, where the intercept $[i,1,p,t]$ and slope matrix $[i,-1,p,t]$ describes a subgradient of the value function at grid point i for position p at time t .
path	3-dimensional array representing the generated paths. Array $[i,j,]$ represents the state at time i for sample path j .
path_nn	Array containing the nearest neighbours, where the $(\text{dim}(\text{path})[1]*(i-1) + j)$ -th entry corresponds to the nearest neighbour for sample path i at time j . Optional if grid and path are supplied.
disturb	5-dimensional array containing the disturbance matrices. Matrix $[:,i,j,k]$ represents the disturbance used in sub-simulation i on sample path j at time k .

weight	Array specifying the probability weights of the disturbance matrices.
grid	Matrix representing the grid, where the 1st column contains only 1s and the matrix [i,-1] represents a particular state.
Neighbour	Optional function to calculate the nearest neighbours. If not provided, the Neighbour function from the rflann package is used instead.
control	<p>Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:</p> <ul style="list-style-type: none"> • Matrix of dimension $n_pos \times n_action$, where entry [i,j] describes the next position after selecting action j at position i. • 3-dimensional 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.

Value

Two possible outputs:

- Full control: 3-dimensional array, where entry [i,j,k] represents the martingale increment at time i for position j on sample path k.
- Partial control: 4-dimensional array, where entry [i,j,k,l] represents the martingale increment at time i after applying action j on sample path k to position l.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
disturb_weight <- rep(1 / 10, 10)
control_mat <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control_mat, disturb, disturb_weight, r_index)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
path_nn <- Neighbour(matrix(path, ncol = 2), grid, 1, "kdtree", 0, 1)$indices
disturb <- array(0, dim = c(2, 2, 20, 100, 50))
disturb[1,1,,,] <- 1
disturb[2,2,,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(100000))
weight <- rep(1 / 20, 20)
```

```
mart <- FastMartingale(bellman$value, path, path_nn, disturb, weight,
                      grid, control = control_mat)
```

FiniteMartingale	<i>Finite Martingales</i>
------------------	---------------------------

Description

Compute the martingale increments for finite distribution case.

Usage

```
FiniteMartingale(grid, value, expected, path_disturb, path_nn, control,
                 path, build, Neighbour)
```

Arguments

grid	Matrix representing the grid, whose i-th row matrix [i,] corresponds to i-th point of the grid. The matrix [i,1] equals to 1 while the vector [i,-1] represents the system state.
value	4-dimensional array representing the subgradient envelope of the value function, where the intercept [i,1,p,t] and slope matrix [i,-1,p,t] describes a subgradient of the value function at grid point i for position p at time t.
expected	4-dimensional array representing the expected value functions. Same format as the value array.
path_disturb	4-dimensional array containing the path disturbances. Matrix [i,j] represents the disturbance at time i for sample path j.
path_nn	Array containing the nearest neighbours, where the (dim(path)[1]*(i-1) + j)-th entry corresponds to the nearest neighbour for sample path i at time j. Optional if grid and path are supplied.
control	Array representing the transition probabilities of the controlled Markov chain. Two possible inputs: <ul style="list-style-type: none"> Matrix of dimension $n_pos \times n_action$, where entry [i,j] describes the next position after selecting action j at position i. 3-dimensional 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.
path	3-dimensional array representing the generated paths. Array [i,j,] represents the state at time i for sample path j.
build	string indicating which build method used to obtain expected value functions: "fast" and "slow".
Neighbour	Optional function to calculate the nearest neighbours. If not provided, the Neighbour function from the rflann package is used instead.

Value

Two possible outputs:

- Full control: 3-dimensional array, where entry $[i,j,k]$ represents the martingale increment at time i for position j on sample path k .
- Partial control: 4-dimensional array, where entry $[i,j,k,l]$ represents the martingale increment at time i after applying action j on sample path k to position l .

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, weight, r_index)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
path_nn <- Neighbour(matrix(path, ncol = 2), grid, 1, "kdtree", 0, 1)$indices
mart <- FiniteMartingale(grid, bellman$value, bellman$expected,
  path_disturb, path_nn, control, path, "fast")
```

GetBounds

Confidence Bounds

Description

Confidence bounds for the value.

Usage

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

Arguments

duality	Object returned by the Duality 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
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
disturb_weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, disturb_weight, r_index)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
path_nn <- Neighbour(matrix(path, ncol = 2), grid, 1, "kdtree", 0, 1)$indices
subsim_disturb <- array(0, dim = c(2, 2, 20, 100, 50))
subsim_disturb[1,1,,,] <- 1
subsim_disturb[2,2,,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(100000))
subsim_weight <- rep(1 / 20, 20)
mart <- FastMartingale(bellman$value, path, path_nn, subsim_disturb,
  subsim_weight, grid, control = control)
RewardFunc <- function(state, time) {
  output <- array(data = 0, dim = c(nrow(state), 4))
  output[,4] <- pmax(40 - state[,2], 0)
  return(output)
}
path_action <- PathPolicy(path, path_nn, control, RewardFunc, bellman$expected)
duality <- Duality(path, control, RewardFunc, mart, path_action)
bounds <- GetBounds(duality, 0.05, 2)
```

Martingale

Martingale Increments

Description

Compute the martingale increments.

Usage

```
Martingale(value, disturb, weight, path, control)
```

Arguments

value	4-dimensional array representing the subgradient envelope of the value function, where the intercept $[i,1,p,t]$ and slope matrix $[i,-1,p,t]$ describes a subgradient of the value function at grid point i for position p at time t .
disturb	5-dimensional array containing the disturbance matrices. Matrix $[i,j,k]$ represents the disturbance used in sub-simulation i on sample path j at time k .
weight	Array specifying the probability weights of the disturbance matrices.
path	3-dimensional array representing the generated paths. Array $[i,j]$ represents the state at time i for sample path j .
control	<p>Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:</p> <ul style="list-style-type: none"> • Matrix of dimension $n_pos \times n_action$, where entry $[i,j]$ describes the next position after selecting action j at position i. • 3-dimensional 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.

Value

Two possible outputs:

- Full control: 3-dimensional array, where entry $[i,j,k]$ represents the martingale increment at time i for position j on sample path k .
- Partial control: 4-dimensional array, where entry $[i,j,k,l]$ represents the martingale increment at time i after applying action j on sample path k to position l .

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
disturb_weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, disturb_weight, r_index)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
disturb <- array(0, dim = c(2, 2, 20, 100, 50))
disturb[1,1,,,] <- 1
disturb[2,2,,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(100000))
weight <- rep(1 / 20, 20)
mart <- Martingale(bellman$value, disturb, weight, path, control)
```

Path

Path Simulation

Description

Simulate sample paths.

Usage

```
Path(start, disturb)
```

Arguments

start	Array representing the start. The first entry must be 1 and array [-1] represents the starting state.
disturb	4-dimensional array containing the path disturbances. Matrix $[:,i,j]$ represents the disturbance at time i for sample path j .

Value

3-dimensional array representing the generated paths. Array $[i,j,]$ represents the state at time i for sample path j .

Author(s)

Jeremy Yee

Examples

```
## Simulating AR(2) process
start <- c(1, 0, 0)
n_dim <- length(start)
n_path <- 10
psi1 <- 0.3
psi2 <- 0.65
n_dec <- 21
path_disturb <- array(data = matrix(c(1, 0, 0,
                                     0, 0, 1,
                                     0, psi2, psi1), ncol = 3, byrow = TRUE),
                      dim = c(n_dim, n_dim, (n_dec - 1), n_path))
path_disturb[3,1,,] <- runif((n_dec - 1) * n_path, -1, 1)
path <- Path(start, path_disturb)
```

PathPolicy	<i>Prescribed Policy</i>
------------	--------------------------

Description

Policies prescribed to selected sample paths.

Usage

```
PathPolicy(path, path_nn, control, Reward, expected, grid)
```

Arguments

path	3-dimensional array representing the generated paths. Array [i,j] represents the state at time i for sample path j.
path_nn	Array containing the nearest neighbours, where the $(\dim(\text{path})[1]*(i-1) + j)$ -th entry corresponds to the nearest neighbour for sample path i at time j. Optional if grid and path are supplied.
control	Array representing the transition probabilities of the controlled Markov chain. Two possible inputs: <ul style="list-style-type: none"> Matrix of dimension $n_{\text{pos}} \times n_{\text{action}}$, where entry [i,j] describes the next position after selecting action j at position i. 3-dimensional array with dimensions $n_{\text{pos}} \times n_{\text{action}} \times n_{\text{pos}}$, 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: <ul style="list-style-type: none"> $n \times d$ matrix representing the n d-dimensional states. A natural number representing the decision epoch.

The function should output the following:

	<ul style="list-style-type: none"> • $n \times (p \times a)$ matrix representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a \times (j - 1) + k]$-th entry corresponds to the reward from applying the k-th action to the j-th position for the i-th state.
expected	4-dimensional array representing the subgradient envelope of the expected value function, where the intercept $[i,1,p,t]$ and slope matrix $[i,-1,p,t]$ describes a subgradient at grid point i for position p at time t .
grid	Matrix representing the grid, whose i -th row matrix $[i,]$ corresponds to i -th point of the grid. The matrix $[i,1]$ equals to 1 while the vector $[i,-1]$ represents the system state.

Value

3-dimensional array representing the prescribed policy for the sample paths. Entry $[i,j,k]$ gives the prescribed action at time i for position j on sample path k .

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
disturb_weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, disturb_weight, r_index)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
path_nn <- Neighbour(matrix(path, ncol = 2), grid, 1, "kdtree", 0, 1)$indices
RewardFunc <- function(state, time) {
  output <- array(data = 0, dim = c(nrow(state), 4))
  output[,4] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
  return(output)
}
path_action <- PathPolicy(path, path_nn, control, RewardFunc, bellman$expected)
```

StochasticGrid	<i>Stochastic grid</i>
----------------	------------------------

Description

Generate a grid using k-means clustering.

Usage

```
StochasticGrid(start, disturb, n_grid, max_iter, warning)
```

Arguments

start	Array representing the start. The first entry must be 1 and array [-1] represents the starting state.
disturb	4 dimensional array containing the path disturbances. Matrix $[:,i,j]$ represents the disturbance at time i for sample path j .
n_grid	Number of grid points in the stochastic grid.
max_iter	Maximum iterations in the k-means clustering algorithm.
warning	Boolean indicating whether messages from the k-means clustering algorithm are to be displayed

Value

Matrix representing the stochastic matrix. Each row represents a particular grid point. The first column contains only 1s.

Author(s)

Jeremy Yee

Examples

```
## Generate a stochastic matrix using an AR(2) process
start <- c(1, 0, 0)
n_dim <- length(start)
n_path <- 10
psi1 <- 0.3
psi2 <- 0.65
n_dec <- 21
path_disturb <- array(data = matrix(c(1, 0, 0,
                                     0, 0, 1,
                                     0, psi2, psi1), ncol = 3, byrow = TRUE),
                      dim = c(n_dim, n_dim, (n_dec - 1), n_path))
path_disturb[3,1,,] <- runif((n_dec - 1) * n_path, -1, 1)
n_grid <- 10
grid <- StochasticGrid(start, path_disturb, n_grid, 10, TRUE)
```

TestPolicy

Backtesting Prescribed policy

Description

Backtesting prescribed policy.

Usage

```
TestPolicy(position, path, control, Reward, path_action)
```

Arguments

position	Natural number indicating the starting position.
path	3-dimensional array representing the generated paths. Array [i,j,] represents the state at time i for sample path j.
control	<p>Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:</p> <ul style="list-style-type: none"> Matrix of dimension $n_pos \times n_action$, where entry [i,j] describes the next position after selecting action j at position i. 3-dimensional 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	<p>User supplied function to represent the rewards for all positions and actions. The function should take in the following arguments, in this order:</p> <ul style="list-style-type: none"> $n \times d$ matrix representing the n d-dimensional states. A natural number representing the decision epoch. <p>The function should output the following:</p> <ul style="list-style-type: none"> $n \times (p \times a)$ matrix representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a \times (j - 1) + k]$-th entry of the matrix corresponds to the reward from applying the k-th action to the j-th position for the i-th state. The ordering of the positions and actions should coincide with the control matrix.
path_action	3-dimensional array representing the prescribed policy for the sample paths. Entry [i,j,k] captures the prescribed action at time i for position j on sample path k.

Value

Array containing the backtesting values for each sample path.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
disturb_weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, disturb_weight, r_index)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
path_nn <- Neighbour(matrix(path, ncol = 2), grid, 1, "kdtree", 0, 1)$indices
RewardFunc <- function(state, time) {
  output <- array(data = 0, dim = c(nrow(state), 4))
  output[,4] <- pmax(40 - state[,2], 0)
  return(output)
}
path_action <- PathPolicy(path, path_nn, control, RewardFunc, bellman$expected)
test <- TestPolicy(2, path, control, RewardFunc, path_action)
```

TestPolicy2

Backtesting Prescribed policy

Description

Backtesting prescribed policy with position evolution

Usage

```
TestPolicy2(position, path, control, Reward, path_action)
```

Arguments

- | | |
|----------|---|
| position | Natural number indicating the starting position. |
| path | 3-dimensional array representing the generated paths. Array [i,j,] represents the state at time i for sample path j. |
| control | <p>Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:</p> <ul style="list-style-type: none"> • Matrix of dimension $n_{pos} \times n_{action}$, where entry [i,j] describes the next position after selecting action j at position i. |

	<ul style="list-style-type: none"> • 3-dimensional array with dimensions $n_{\text{pos}} \times n_{\text{action}} \times n_{\text{pos}}$, where entry $[i,j,k]$ is the probability of moving to position k after applying action j to position i.
Reward	<p>User supplied function to represent the rewards for all positions and actions. The function should take in the following arguments, in this order:</p> <ul style="list-style-type: none"> • $n \times d$ matrix representing the n d-dimensional states. • A natural number representing the decision epoch. <p>The function should output the following:</p> <ul style="list-style-type: none"> • $n \times (p \times a)$ matrix representing the rewards, where p is the number of positions and a is the number of actions in the problem. The $[i, a \times (j - 1) + k]$-th entry of the matrix corresponds to the reward from applying the k-th action to the j-th position for the i-th state. The ordering of the positions and actions should coincide with the control matrix.
path_action	3-dimensional array representing the prescribed policy for the sample paths. Entry $[i,j,k]$ captures the prescribed action at time i for position j on sample path k .

Value

value	Array containing the backtesting values for each sample path.
position	Matrix containing the evolution of the position. Entry $[i,j]$ refers to the position at time j for sample path i .
action	Matrix containing the actions taken. Entry $[i,j]$ refers to the action at time j for sample path i .

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
disturb_weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, control, disturb, disturb_weight, r_index)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
```

```

path_nn <- Neighbour(matrix(path, ncol = 2), grid, 1, "kdtree", 0, 1)$indices
RewardFunc <- function(state, time) {
  output <- array(data = 0, dim = c(nrow(state), 4))
  output[,4] <- pmax(40 - state[,2], 0)
  return(output)
}
path_action <- PathPolicy(path, path_nn, control, RewardFunc, bellman$expected)
test <- TestPolicy2(2, path, control, RewardFunc, path_action)

```

WrongMartingale

Wrong Martingales

Description

Compute the incorrect martingale increments for finite distribution case. Used for one of our papers. DO NOT USE.

Usage

```
WrongMartingale(value, expected, path, control)
```

Arguments

- | | |
|----------|---|
| value | 4-dimensional array representing the subgradient envelope of the value function, where the intercept $[i,1,p,t]$ and slope matrix $[i,-1,p,t]$ describes a subgradient of the value function at grid point i for position p at time t . |
| expected | 4-dimensional array representing the expected value functions. Same format as the value array. |
| path | 3-dimensional array representing the generated paths. Array $[i,j,]$ represents the state at time i for sample path j . |
| control | <p>Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:</p> <ul style="list-style-type: none"> • Matrix of dimension $n_pos \times n_action$, where entry $[i,j]$ describes the next position after selecting action j at position i. • 3-dimensional 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. |

Value

Two possible outputs:

- Full control: 3-dimensional array, where entry $[i,j,k]$ represents the martingale increment at time i for position j on sample path k .
- Partial control: 4-dimensional array, where entry $[i,j,k,l]$ represents the martingale increment at time i after applying action j on sample path k to position l .

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 91), c(seq(10, 100, length = 91))))
disturb <- array(0, dim = c(2, 2, 10))
disturb[1,1,] <- 1
disturb[2,2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(10))
weight <- rep(1 / 10, 10)
control <- matrix(c(c(1, 1), c(2, 1)), nrow = 2, byrow = TRUE)
reward <- array(0, dim = c(91, 2, 2, 2, 51))
reward[grid[,2] <= 4,1,2,2,] <- 40
reward[grid[,2] <= 4,2,2,2,] <- -1
bellman <- Bellman(grid, reward, control, disturb, weight)
path_disturb <- array(0, dim = c(2, 2, 50, 100))
path_disturb[1,1,,] <- 1
path_disturb[2,2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rnorm(5000))
start <- c(1, 36)
path <- Path(start, path_disturb)
path_nn <- Neighbour(matrix(path, ncol = 2), grid, 1, "kdtree", 0, 1)$indices
mart <- WrongMartingale(bellman$value, bellman$expected, path, control)
```

Index

Bellman, [2](#)
BellmanAccelerated, [3](#)

Duality, [5](#)

Expected, [6](#)
ExpectedAccelerated, [7](#)

FastBellman, [8](#)
FastExpected, [10](#)
FastMartingale, [11](#)
FiniteMartingale, [13](#)

GetBounds, [14](#)

Martingale, [16](#)

Path, [17](#)
PathPolicy, [18](#)

StochasticGrid, [20](#)

TestPolicy, [21](#)
TestPolicy2, [22](#)

WrongMartingale, [24](#)