Package ‘rcss’

December 29, 2017

Type Package

Title Convex Switching Systems

Version 1.5

Date 2017-12-29

Author Jeremy Yee and Juri Hinz

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 (>= 1.4)

Imports Rcpp (>= 0.11.6)

LinkingTo Rcpp, RcppArmadillo, rflann (>= 1.4)

Suggests R.rsp

VignetteBuilder R.rsp

NeedsCompilation yes

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

R topics documented:

`AcceleratedBellman` *Bellman recursion accelerated with k nearest neighbours*

Description

Approximate the value functions using k nearest neighbours.

Usage

```
AcceleratedBellman(grid, reward, scrap, control, disturb, weight, k = 1)
```

Arguments

`grid` Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.

`reward` 5-D array representing the tangent approximation of the reward. Entry [i,,a,p,t] captures the tangent at grid point i for action a taken in position p at time t. The intercept is given by [i,1,a,p,t] and slope by [i,-1,a,p,t].

`scrap` 3-D array representing the tangent approximation of the scrap. Entry [i,,p] captures the tangent at grid point i for position p. The intercept is given by [i,1,p] and slope by [i,-1,p].

`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.

`disturb` 3-D array containing the disturbance matrices. Matrix [,,i] specifies the i-th disturbance matrix.

`weight` Array containing the probability weights of the disturbance matrices.

`k` Number of nearest neighbours used for each grid point.

Value

value	4-D array tangent approximation of the value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a tangent of the value function at grid point i for position p at time t.
expected	4-D array representing the expected value functions.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 -0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
bellman <- AcceleratedBellman(grid, reward, scrap, control, disturb, weight)
```

AcceleratedExpected *Expected value function using k nearest neighbours*

Description

Approximate the expected value function using k nearest neighbours.

Usage

```
AcceleratedExpected(grid, value, disturb, weight, k = 1)
```

Arguments

`grid`	Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.
`value`	Matrix representing the tangent approximation of the future value function, where the intercept [i,1] and slope [i,-1] describes a tangent at grid point i.
`disturb`	3-D array containing the disturbance matrices. Matrix [,,i] specifies the i-th disturbance matrix.
`weight`	Array containing the probability weights of the disturbance matrices.
`k`	Number of nearest neighbours used for each grid point.

Value

Matrix representing the tangent approximation 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, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 -0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
bellman <- AcceleratedBellman(grid, reward, scrap, control, disturb, weight)
expected <- AcceleratedExpected(grid, bellman$value[,,2,2], disturb, weight)
```

`AddDual` *Additive duals*

Description

Additive duals by comparing all tangents.

Usage

```
AddDual(path, subsim, weight, value, Scrap)
```

Arguments

`path`	3-D array representing sample paths. Entry [i,,j] represents the state at time j for sample path i.
`subsim`	5-D array containing the subsimulation disturbance matrices. Matrix [,,i,j,t] represents the disturbance used in subsimulation i on sample path j at time t.
`weight`	Array specifying the probability weights of the subsimulation disturbance matrices.
`value`	4-D array tangent approximation of the value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a tangent of the value function at grid point i for position p at time t.
`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 scraps, where p is the number of positions. The $[i, p]$-th entry corresponds to the scrap at the p-th position for the i-th state.

Value

3-D array where entry [i,p,t] represents the martingale increment at time t for position p on sample path i.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
```

```
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm(100 * 50 / 2)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Scrap function
ScrapFunc <- function(state) {
    output <- array(data = 0, dim = c(nrow(state), 2))
    output[,2] <- exp(-0.06 * 0.02 * 50) * pmax(40 - state[,2], 0)
    return(output)
}
## Subsimulation disturbances
subsim <- array(0, dim = c(2, 2, 100, 100, 50))
subsim[1,1,,,] <- 1
rand2 <- rnorm(100 * 100 * 50 / 2)
rand2 <- as.vector(rbind(rand2, -rand2))
subsim[2,2,,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand2)
subsim_weight <- rep(1 / 100, 100)
## Additive duals
mart <- AddDual(path, subsim, subsim_weight, bellman$value, ScrapFunc)
```

AddDualBounds *Additive duals bound estimates*

Description

Bound estimates using the addiitive duals.

Usage

```
AddDualBounds(path, control, Reward, Scrap, dual, policy)
```

Arguments

`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 n_pos × n_action, where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions n_pos × n_action × 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.

The function should output the following:

- Matrix with dimensions $n \times p)$ representing the scraps, where p is the number of positions. The $[i, p]$-th entry corresponds to the scrap at the p-th position for the i-th state.

`dual` 3-D array where entry [i,p,t] represents the additive dual at time t for position p on sample path i.

`policy` 3-D array representing the prescribed policy for the sample paths. Entry [i,p,t] gives the prescribed action at time t for position p on sample path t.

Value

List containing:

`primal` 3-D array representing the primal values, where entry [i,p,t] represents the value at time t for position p on sample path i.

`dual` 3-D array representing the dual values. Same format as above.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm(100 * 50 / 2)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Scrap function
ScrapFunc <- function(state) {
    output <- array(data = 0, dim = c(nrow(state), 2))
    output[,2] <- exp(-0.06 * 0.02 * 50) * pmax(40 - state[,2], 0)
    return(output)
}
## Additive duals
mart <- FiniteAddDual(path, path_disturb, grid, bellman$value, bellman$expected, "fast")
bounds <- AddDualBounds(path, control, RewardFunc, ScrapFunc, mart, policy)
```

Bellman *Bellman recursion*

Description

Approximate the value functions by comparing all tangents.

Usage

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

Arguments

- `grid` Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.
- `reward` 5-D array representing the tangent approximation of the reward. Entry [i,,a,p,t] captures the tangent at grid point i for action a taken in position p at time t. The intercept is given by [i,1,a,p,t] and slope by [i,-1,a,p,t].
- `scrap` 3-D array representing the tangent approximation of the scrap. Entry [i,,p] captures the tangent at grid point i for position p. The intercept is given by [i,1,p] and slope by [i,-1,p].
- `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.
- `disturb` 3-D 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-D array tangent approximation of the value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a tangent of the value function at grid point i for position p at time t.
- `expected` 4-D array representing the expected value functions.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 -0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
bellman <- Bellman(grid, reward, scrap, control, disturb, weight)
```

Expected — *Expected value function*

Description

Approximate the expected value function by comparing all tangents.

Usage

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

Arguments

`grid`	Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.
`value`	Matrix representing the tangent approximation of the future value function, where the intercept [i,1] and slope [i,-1] describes a tangent at grid point i.
`disturb`	3-D 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 tangent approximation 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, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 -0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
bellman <- Bellman(grid, reward, scrap, control, disturb, weight)
expected <- Expected(grid, bellman$value[,,2,2], disturb, weight)
```

`FastAddDual` *Fast additive duals*

Description

Additive duals using nearest neighbours.

Usage

```
FastAddDual(path, subsim, weight, grid, value, Scrap)
```

Arguments

`path`	3-D array representing sample paths. Entry [i,,j] represents the state at time j for sample path i.
`subsim`	5-D array containing the subsimulation disturbance matrices. Matrix [,,i,j,t] represents the disturbance used in subsimulation i on sample path j at time t.
`weight`	Array specifying the probability weights of the subsimulation disturbance matrices.
`grid`	Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.

value
: 4-D array tangent approximation of the value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a tangent of the value function at grid point i for position p at time t.

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 scraps, where p is the number of positions. The $[i, p]$-th entry corresponds to the scrap at the p-th position for the i-th state.

Value

3-D array where entry [i,p,t] represents the martingale increment at time t for position p on sample path i.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm(100 * 50 / 2)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
```

```
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Scrap function
ScrapFunc <- function(state) {
    output <- array(data = 0, dim = c(nrow(state), 2))
    output[,2] <- exp(-0.06 * 0.02 * 50) * pmax(40 - state[,2], 0)
    return(output)
}
## Subsimulation disturbances
subsim <- array(0, dim = c(2, 2, 100, 100, 50))
subsim[1,1,,,] <- 1
rand2 <- rnorm(100 * 100 * 50 / 2)
rand2 <- as.vector(rbind(rand2, -rand2))
subsim[2,2,,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand2)
subsim_weight <- rep(1 / 100, 100)
## Additive duals
mart <- FastAddDual(path, subsim, subsim_weight, grid, bellman$value, ScrapFunc)
```

FastBellman *Fast Bellman Recursion*

Description

Approximate the value functions using conditional expectation matrices

Usage

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

Arguments

grid	Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.
reward	5-D array representing the tangent approximation of the reward. Entry [i,,a,p,t] captures the tangent at grid point i for action a taken in position p at time t. The intercept is given by [i,1,a,p,t] and slope by [i,-1,a,p,t].
scrap	3-D array representing the tangent approximation of the scrap. Entry [i,,p] captures the tangent at grid point i for position p. The intercept is given by [i,1,p] and slope by [i,-1,p].
control	Array representing the transition probabilities of the controlled Markov chain. Two possible inputs:

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

`disturb`	3-D 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.
`smooth`	The number of nearest neighbours used to smooth the expected value functions during the Bellman recursion.

Value

`value`	4-D array tangent approximation of the value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a subgradient of the value function at grid point i for position p at time t.
`expected`	4-D array representing the expected value functions.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 -0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
```

FastExpected *Fast expected value function*

Description

Approximate the expected value function using conditional expectaion matrices.

Usage

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

Arguments

grid	Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.
value	Matrix representing the tangent approximation of the future value function, where the intercept [i,1] and slope [i,-1] describes a tangent at grid point i.
disturb	3-D 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.
smooth	The number of nearest neighbours used to smooth the expected value functions during the Bellman recursion.

Value

Matrix representing the tangent approximation 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, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 -0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
```

```
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
expected <- FastExpected(grid, bellman$value[,,2,2], disturb, weight, r_index)
```

`FastPathPolicy` *Fast prescribed policy*

Description

Policy precribed to provided sample paths using nearest neighbours

Usage

```
FastPathPolicy(path, grid, control, Reward, expected)
```

Arguments

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

`grid` Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.

`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.

`expected` 4-D array representing the tangent approximation of the expected value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a tangent at grid point i for position p at time t.

Value

3-D array representing the prescribed policy for the sample paths. Entry [i,p,t] gives the prescribed action at time t for position p on sample path t.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm(100 * 50 / 2)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
```

`FiniteAddDual` *Finite distribution case additive duals*

Description

Additive duals for finite distribution case. No nested simulation.

Usage

```
FiniteAddDual(path, disturb, grid, value, expected, build = "fast", k = 1)
```

Arguments

`path`	3-D array representing sample paths. Entry [i,,j] represents the state at time j for sample path i.
`disturb`	4-D array containing the disturbances used to generate the paths. Matrix [,,i,t] represents the disturbance at time t for sample path i.
`grid`	Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.
`value`	4-D array tangent approximation of the value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a tangent of the value function at grid point i for position p at time t.
`expected`	4-D array representing the tangent approximation of the expected value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a tangent at grid point i for position p at time t.
`build`	string indicating which build method used to obtain expected value functions: "fast", "accelerated", and "slow".
`k`	Number of nearest neighbours used for "accelerated" build.

Value

3-D array where entry [i,p,t] represents the martingale increment at time t for position p on sample path i.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
```

```
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- sample(quantile, 100 * 50 / 2, TRUE)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Scrap function
ScrapFunc <- function(state) {
    output <- array(data = 0, dim = c(nrow(state), 2))
    output[,2] <- exp(-0.06 * 0.02 * 50) * pmax(40 - state[,2], 0)
    return(output)
}
## Additive duals
mart <- FiniteAddDual(path, path_disturb, grid, bellman$value, bellman$expected, "fast")
```

`FullTestPolicy` *Full backtesting prescribed policy*

Description

Backtesting prescribed policy with value, position, action evolution.

Usage

```
FullTestPolicy(position, path, control, Reward, Scrap, policy)
```

Arguments

`position` Natural number indicating the 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 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.

The function should output the following:

- Matrix with dimensions $n \times p)$ representing the scraps, where p is the number of positions. The $[i, p]$-th entry corresponds to the scrap at the p-th position for the i-th state.

`policy` 3-D array representing the prescribed policy for the sample paths. Entry [i,p,t] gives the prescribed action at time t for position p on sample path t.

Value

`value` Matrix containing the backtesting values for each sample path. Entry[i,t] refers to the value at time t for sample path i.

`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
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm(100 * 50 / 2)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Scrap function
ScrapFunc <- function(state) {
    output <- array(data = 0, dim = c(nrow(state), 2))
    output[,2] <- exp(-0.06 * 0.02 * 50) * pmax(40 - state[,2], 0)
    return(output)
}
test <- FullTestPolicy(2, path, control, RewardFunc, ScrapFunc, policy)
```

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, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- sample(quantile, 100 * 50 / 2, TRUE)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
```

```
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Scrap function
ScrapFunc <- function(state) {
    output <- array(data = 0, dim = c(nrow(state), 2))
    output[,2] <- exp(-0.06 * 0.02 * 50) * pmax(40 - state[,2], 0)
    return(output)
}
## Additive duals
mart <- FiniteAddDual(path, path_disturb, grid, bellman$value, bellman$expected, "fast")
bounds <- AddDualBounds(path, control, RewardFunc, ScrapFunc, mart, policy)
```

`Optimal` *Optimal*

Description

Find the maximising tangent at each grid point.

Usage

```
Optimal(grid, tangent)
```

Arguments

`grid`	Matrix representing the grid. The i-th row corresponds to i-th point of the grid. The j-th column captures the dimensions. The first column must equal to 1.
`tangent`	Matrix representing the collection of tangents, where the intercept [i,1] and slope [i,-1] describes a tangent at grid point i.

Value

Matrix representing the maximum of the tangents at each grid point, where the intercept [i,1] and slope [i,-1] describes the maximising tangent at grid point i.

Author(s)

Jeremy Yee

Examples

```
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 91))))
tangent <- matrix(rnorm(81 * 2), ncol = 2)
Optimal(grid, tangent)
```

PathDisturb *Path Simulation*

Description

Simulate sample paths using disturbances.

Usage

```
PathDisturb(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 j for sample path i.

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_path, (n_dec - 1)))
path_disturb[3,1,,] <- runif(n_path * (n_dec - 1), -1, 1)
path <- PathDisturb(start, path_disturb)
```

`PathPolicy` *Prescribed policy*

Description

Policy prescribed to provided sample paths

Usage

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

Arguments

`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 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.

`expected` 4-D array representing the tangent approximation of the expected value function, where the intercept [i,1,p,t] and slope [i,-1,p,t] describes a tangent at grid point i for position p at time t.

Value

3-D array representing the prescribed policy for the sample paths. Entry [i,p,t] gives the prescribed action at time t for position p on sample path t.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm(100 * 50 / 2)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- PathPolicy(path, control, RewardFunc, bellman$expected)
```

`StochasticGrid` *Stochastic grid*

Description

Generate a grid using k-means clustering.

Usage

```
StochasticGrid(path, n_grid, max_iter, warning)
```

Arguments

path	3-D array representing sample paths. Entry [i,,j] represents the state at time j for sample path i.
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 grid. Each row represents a particular grid point. The first column contains only 1.

Author(s)

Jeremy Yee

Examples

```
## Generate paths
start <- c(1, 36)
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm((50 * 100) / 2)
rand1 <- as.vector(rbind(rand1, -rand1))
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
sgrid <- StochasticGrid(path, 81, 10)
```

TestPolicy — *Backtesting Prescribed policy*

Description

Backtesting prescribed policy.

Usage

```
TestPolicy(position, path, control, Reward, Scrap, policy)
```

Arguments

position	Natural number indicating the 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 n_pos × n_action, where entry [i,j] describes the next position after selecting action j at position i.
- 3-D array with dimensions n_pos × n_action × 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.

The function should output the following:

- Matrix with dimensions $n \times p)$ representing the scraps, where p is the number of positions. The $[i, p]$-th entry corresponds to the scrap at the p-th position for the i-th state.

`policy` 3-D array representing the prescribed policy for the sample paths. Entry [i,p,t] gives the prescribed action at time t for position p on sample path t.

Value

Array containing the backtesting values for each sample path.

Author(s)

Jeremy Yee

Examples

```
## Bermuda put option
grid <- as.matrix(cbind(rep(1, 81), c(seq(20, 60, length = 81))))
disturb <- array(0, dim = c(2, 2, 100))
disturb[1, 1,] <- 1
quantile <- qnorm(seq(0, 1, length = (100 + 2))[c(-1, -(100 + 2))])
disturb[2, 2,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * quantile)
weight <- rep(1 / 100, 100)
control <- matrix(c(c(1, 2),c(1, 1)), nrow = 2)
reward <- array(data = 0, dim = c(81, 2, 2, 2, 50))
in_money <- grid[, 2] <= 40
reward[in_money, 1, 2, 2,] <- 40
reward[in_money, 2, 2, 2,] <- -1
for (tt in 1:50){
```

```
  reward[,,2,2,tt] <- exp(-0.06 * 0.02 * (tt - 1)) * reward[,,2,2,tt]
}
scrap <- array(data = 0, dim = c(81, 2, 2))
scrap[in_money, 1, 2] <- 40
scrap[in_money, 2, 2] <- -1
scrap[,,2] <- exp(-0.06 * 0.02 * 50) * scrap[,,2]
r_index <- matrix(c(2, 2), ncol = 2)
bellman <- FastBellman(grid, reward, scrap, control, disturb, weight, r_index)
set.seed(12345)
start <- c(1, 36) ## starting state
path_disturb <- array(0, dim = c(2, 2, 100, 50))
path_disturb[1, 1,,] <- 1
rand1 <- rnorm(100 * 50 / 2)
rand1 <- as.vector(rbind(rand1, -rand1))  ## anti-thetic disturbances
path_disturb[2, 2,,] <- exp((0.06 - 0.5 * 0.2^2) * 0.02 + 0.2 * sqrt(0.02) * rand1)
path <- PathDisturb(start, path_disturb)
## Reward function
RewardFunc <- function(state, time) {
    output <- array(data = 0, dim = c(nrow(state), 2, 2))
    output[,2, 2] <- exp(-0.06 * 0.02 * (time - 1)) * pmax(40 - state[,2], 0)
    return(output)
}
policy <- FastPathPolicy(path, grid, control, RewardFunc, bellman$expected)
## Scrap function
ScrapFunc <- function(state) {
    output <- array(data = 0, dim = c(nrow(state), 2))
    output[,2] <- exp(-0.06 * 0.02 * 50) * pmax(40 - state[,2], 0)
    return(output)
}
test <- TestPolicy(2, path, control, RewardFunc, ScrapFunc, policy)
```

Index