Package 'PricingBandits'

September 9, 2026

Type Package

Title Multi-Armed Bandit Approaches to Pricing Experiments

Version 2.0.0

Description Implements multi-armed bandit approaches for pricing experiments with an unknown demand curve, as developed in Weaver, Kumar, and Jain, ``Nonparametric Pricing Bandits Leveraging Informational Externalities to Learn the Demand Curve'' <doi:10.1287/mksc.2022.0247>. Includes Upper Confidence Bound (UCB) and Thompson Sampling (TS) baselines, Gaussian process variants ('GP-UCB', 'GP-TS'), monotonic Gaussian process variants that constrain demand to be weakly decreasing in price, and heterogeneous-noise extensions. The willingness-to-pay distribution is fully user-specified via a vector of consumer valuations, so any demand environment can be simulated or replayed.

License MIT + file LICENSE

URL https://github.com/ian-weaver/PricingBandits

BugReports https://github.com/ian-weaver/PricingBandits/issues

Encoding UTF-8

Imports stats, Matrix, hash, nloptr, MASS, dplyr, TruncatedNormal, R.utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown, ggplot2

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Ian N. Weaver [aut, cre], Vineet Kumar [aut], Lalit Jain [aut]

Maintainer Ian N. Weaver <weaver.n.ian@gmail.com>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2026-09-09 14:30:02 UTC

Contents

`AggregateDataGP` *Aggregate Data for Gaussian Process (GP) Algorithms*

Description

This function processes price testing data for Gaussian Process algorithms. It computes purchase rates and adjusts observation noise variance for each price. Prices should be in the range [0, 1], and observation noise variance (sigma2_y) must be less than 0.25.

Usage

```
AggregateDataGP(PricesTested, PurchaseDecisions, TestX, sigma2_y, BatchSize)
```

Arguments

PricesTested	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
PurchaseDecisions	A vector of associated purchase decisions for the prices tested.
TestX	A set of prices (values between 0 and 1) to test.
sigma2_y	Observation noise variance (sigma_y^2), must be less than 0.25.
BatchSize	Number of consumers tested before an algorithm can update.

Value

A data frame containing:

PricesTested The prices included in the dataset.

PurchaseRates The mean purchase rate for each price.

sigma_2y The adjusted observation noise variance for each price.

Examples

```
PricesTested <- c(0.1, 0.2, 0.2, 0.9)
PurchaseDecisions <- c(1, 0, 1, 0)
TestX <- seq(10)/10
sigma2_y <- rep(0.25, 10)
BatchSize <- 10
AggregateDataGP(PricesTested, PurchaseDecisions, TestX, sigma2_y, BatchSize)
```

AggregateDataTS *Aggregate Data for Thompson Sampling (TS) Algorithm*

Description

This function processes price testing data for the Thompson Sampling algorithm. It adds prior data for untested prices and aggregates purchase decisions by price. Prices should be in the range [0, 1].

Usage

```
AggregateDataTS(PricesTested, PurchaseDecisions, TestX)
```

Arguments

PricesTested	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
PurchaseDecisions	A vector of associated purchase decisions for the prices tested.
TestX	A set of prices (values between 0 and 1) to test.

Value

A list containing:

s_t The sum of purchase decisions for each price.

n_t The number of observations for each price.

Examples

```
PricesTested <- c(0.1, 0.2, 0.2, 0.9)
PurchaseDecisions <- c(1, 0, 1, 0)
TestX <- seq(10)/10
AggregateDataTS(PricesTested, PurchaseDecisions, TestX)
```

`AggregateDataUCB` *GP Variants*

Description

Collection of functions to aggregate data for various algorithms, including UCB, Thompson Sampling, and Gaussian Processes. Aggregate Data for Upper Confidence Bound (UCB) Algorithm

This function processes price testing data for the UCB algorithm. It aggregates purchase decisions by price, includes priors for prices that have not yet been tested, and returns summarized data. Prices should be in the range [0, 1].

Usage

```
AggregateDataUCB(PricesTested, PurchaseDecisions, TestX)
```

Arguments

`PricesTested`	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
`PurchaseDecisions`	A vector of associated purchase decisions for the prices tested.
`TestX`	A set of prices (values between 0 and 1) to test.

Value

A list containing:

s_t The sum of purchase decisions for each price.

n_t The number of observations for each price.

Examples

```
PricesTested <- c(0.1, 0.2, 0.2, 0.9)
PurchaseDecisions <- c(1, 0, 1, 0)
TestX <- seq(10)/10
AggregateDataUCB(PricesTested, PurchaseDecisions, TestX)
```

BasisFunction *Basis Function*

Description

Computes the values of a basis function at a test point, given a number of knots.

Usage

```
BasisFunction(x, J)
```

Arguments

x	Test point where the basis function is evaluated.
J	Number of knots for the basis function.

Details

The function computes the basis function values using a piecewise linear approach with J knots. The calculation adjusts based on the location of the test point relative to the knot positions.

Value

A vector of length J containing the basis function values at the test point.

CovarianceFromKernel *Covariance Matrix from Kernel*

Description

Computes the covariance matrix between two sets of points.

Usage

```
CovarianceFromKernel(X1, X2, kernel, sigma_f, l)
```

Arguments

X1	Matrix of m points (m x d).
X2	Matrix of n points (n x d).
kernel	Kernel function to compute covariance.
sigma_f	Hyperparameter defining the vertical scale.
l	Hyperparameter defining the horizontal scale.

Value

Covariance matrix of size m x n.

GetDiagnostics *Get Experiment Diagnostics*

Description

Returns the internal fallback counters and monotonicity log accumulated since the last `ResetDiagnostics` call. The same information is attached as the `"diagnostics"` attribute of `PricingBandit` output.

Usage

```
GetDiagnostics()
```

Value

A named list of counters.

GPTS *Gaussian Process Thompson Sampling (GPTS) Policy*

Description

This function implements the GPTS policy for pricing experiments. It uses Gaussian Process regression to generate posterior predictions and action scores.

Usage

```
GPTS(PricesTested, PurchaseDecisions, TestX, sigma2_y, BatchSize)
```

Arguments

`PricesTested`	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
`PurchaseDecisions`	A vector of associated purchase decisions for the prices tested.
`TestX`	A set of prices (values between 0 and 1) to test.
`sigma2_y`	Observation noise variance (must be less than 0.25).
`BatchSize`	Number of consumers tested before an algorithm can update.

Value

A vector of action scores for the prices in ‘TestX‘.

Examples

```
PricesTested <- c(0.1, 0.2, 0.2, 0.9)
PurchaseDecisions <- c(1, 0, 1, 0)
TestX <- seq(10)/10
sigma2_y <- rep(0.25, 11)
BatchSize <- 10
GPTS(PricesTested, PurchaseDecisions, TestX, sigma2_y, BatchSize)
```

GPTS_Mono *Gaussian Process Thompson Sampling Monotonic (GPTS_Mono) Policy*

Description

This function implements the monotonic Gaussian Process Thompson Sampling (GPTS_Mono) policy for pricing experiments. It uses Gaussian Process regression to generate posterior predictions and action scores while ensuring monotonicity.

Usage

```
GPTS_Mono(
  PricesTested,
  PurchaseDecisions,
  TestX,
  Knots,
  sigma2_y,
  BatchSize,
  BasisFunctions,
  LB1,
  LB2,
  UB
)
```

Arguments

PricesTested	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
PurchaseDecisions	A vector of associated purchase decisions for the prices tested.
TestX	A set of prices (values between 0 and 1) to test.
Knots	Knots for the basis functions used in the regression.
sigma2_y	Observation noise variance (must be less than 0.25).
BatchSize	Number of consumers tested before an algorithm can update.
BasisFunctions	Basis functions for the monotonic Gaussian Process regression.
LB1	Lower bound for the test prices.
LB2	Lower bound for the knots.
UB	Upper bound for both test prices and knots.

Value

A vector of action scores for the prices in 'TestX'.

GPUCB *Gaussian Process Upper Confidence Bound (GPUCB) Policy*

Description

This function implements the GPUCB policy for pricing experiments. It uses Gaussian Process regression to calculate action scores based on posterior predictions and confidence intervals.

Usage

```
GPUCB(PricesTested, PurchaseDecisions, TestX, sigma2_y, BatchSize)
```

Arguments

PricesTested	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
PurchaseDecisions	A vector of associated purchase decisions for the prices tested.
TestX	A set of prices (values between 0 and 1) to test.
sigma2_y	Observation noise variance (must be less than 0.25).
BatchSize	Number of consumers tested before an algorithm can update.

Value

A vector of action scores for the prices in 'TestX'.

Examples

```
PricesTested <- c(0.1, 0.2, 0.2, 0.9)
PurchaseDecisions <- c(1, 0, 1, 0)
TestX <- seq(10)/10
sigma2_y <- rep(0.25, 11)
BatchSize <- 10
GPUCB(PricesTested, PurchaseDecisions, TestX, sigma2_y, BatchSize)
```

GPUCB_Mono *Gaussian Process Upper Confidence Bound Monotonic (GPUCB_Mono) Policy*

Description

This function implements the monotonic Gaussian Process Upper Confidence Bound (GPUCB_Mono) policy for pricing experiments. It uses Gaussian Process regression to calculate action scores based on posterior predictions and confidence intervals while ensuring monotonicity.

Usage

```
GPUCB_Mono(
  PricesTested,
  PurchaseDecisions,
  TestX,
  Knots,
  sigma2_y,
  BatchSize,
  BasisFunctions,
  LB1,
  LB2,
  UB
)
```

Arguments

PricesTested	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
PurchaseDecisions	A vector of associated purchase decisions for the prices tested.
TestX	A set of prices (values between 0 and 1) to test.
Knots	Knots for the basis functions used in the regression.
sigma2_y	Observation noise variance (must be less than 0.25).
BatchSize	Number of consumers tested before an algorithm can update.
BasisFunctions	Basis functions for the monotonic Gaussian Process regression.
LB1	Lower bound for the test prices.
LB2	Lower bound for the knots.
UB	Upper bound for both test prices and knots.

Value

A vector of action scores for the prices in ‘TestX‘.

JointCovFromKernel *Joint Covariance Matrix from Kernel*

Description

Computes the joint covariance matrix for a set of points and derivatives.

Usage

```
JointCovFromKernel(X, Index, kernel, sigma_f, l)
```

Arguments

X	Vector of points.
Index	Vector indicating the derivative order at each point (0 or 1).
kernel	Kernel function to compute covariance.
sigma_f	Hyperparameter defining the vertical scale.
l	Hyperparameter defining the horizontal scale.

Value

Joint covariance matrix of size length(X) x length(X).

MABExperiment *Multi-Armed Bandit Experiment Framework*

Description

Runs a multi-armed bandit (MAB) experiment, supporting TS, UCB, Gaussian Process-based variants, and monotonic and heterogeneous variants. It allows simulation of consumer purchase decisions under different MAB policies and underlying WTP distributions.

Usage

```
MABExperiment(
  Valuations,
  Policy,
  TestX,
  NumIter,
  BatchSize,
  NumKnots = NULL,
  Knots = NULL,
  BasisFunctions = NULL,
  HeteroNoise = FALSE,
  Reset = NULL,
  Timeout = 5
)
```

Arguments

`Valuations`	A vector of consumer valuations for the product, used to simulate purchase decisions.
`Policy`	The policy function to be used in the experiment (e.g., UCB, TS, GPTS, GPUCB).
`TestX`	A vector of prices (values between 0 and 1) to test.
`NumIter`	Number of iterations (time steps) to run the experiment.
`BatchSize`	Number of consumers tested before the policy updates its action scores.
`NumKnots`	Number of knots for monotonic Gaussian Process variants (optional, default is 'NULL').
`Knots`	A vector of knot locations for monotonic Gaussian Process regression (optional, default is 'NULL').
`BasisFunctions`	A matrix of basis functions for monotonic Gaussian Process regression (optional, default is 'NULL').
`HeteroNoise`	Logical; if 'TRUE', allows for heterogeneous noise modeling (optional, default is 'FALSE').
`Reset`	Number of consumers before the experiment history is wiped (optional, default is 'NULL').
`Timeout`	Seconds allowed for each truncated-sampling attempt in the monotonic fallback chain (optional, default is 5).

Value

A data frame containing two columns:

PricesTested A vector of prices tested over the experiment.

PurchaseDecisions A vector of binary purchase decisions corresponding to each price tested.

Examples

```
Valuations <- runif(100, min = 0.1, max = 0.9)
TestX <- seq(10)/10
NumIter <- 100
BatchSize <- 10
MABExperiment(Valuations, UCB, TestX, NumIter, BatchSize)
```

`MakePosDefinitive` *Positive Definite Covariance Matrix*

Description

Ensures a covariance matrix is positive definite by adjusting negative eigenvalues.

Usage

```
MakePosDefinitive(CovMatrix, zilon)
```

Arguments

CovMatrix	Matrix to be corrected.
zilon	Small value to replace negative eigenvalues.

Value

Positive definite covariance matrix.

NLML *Gaussian Process Regression*

Description

Functions to perform Gaussian Process regression, optimize hyperparameters, and compute posterior predictions.

Computes the negative log marginal likelihood for Gaussian Process regression.

Usage

```
NLML(Theta, TrainX, TrainY, sigma2_y)
```

Arguments

Theta	Vector of hyperparameters (e.g., sigma_f and l).
TrainX	Matrix of training points (m x d).
TrainY	Vector of training targets (m x 1).
sigma2_y	Observation noise variance (sigma_y^2).

Value

Negative log marginal likelihood value.

NoiseSample *Noise Sampling for Heteroscedastic Gaussian Processes*

Description

This function generates a sample of heteroscedastic noise variance (σ_y^2) for a Gaussian Process. The function processes price testing data and uses posterior predictions to generate a random draw. Prices should be in the range [0, 1], and observation noise variance (σ_y^2) must be less than 0.25.

Usage

```
NoiseSample(PricesTested, PurchaseDecisions, TestX, BatchSize)
```

Arguments

PricesTested	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
PurchaseDecisions	A vector of associated purchase decisions for the prices tested.
TestX	A set of prices (values between 0 and 1) to test.
BatchSize	Number of consumers tested before an algorithm can update.

Value

A vector containing the sampled heteroscedastic noise variances (σ_y^2).

Examples

```
PricesTested <- c(0.1, 0.2, 0.2, 0.9)
PurchaseDecisions <- c(1, 0, 1, 0)
TestX <- seq(10)/10
sigma2_y <- rep(0.25, 10)
BatchSize <- 10
NoiseSample(PricesTested, PurchaseDecisions, TestX, BatchSize)
```

NonMonoPolicyEval *Evaluate Non-Monotonic Policies*

Description

This function evaluates a non-monotonic policy by executing it with the provided parameters. If the policy relies on Gaussian Processes (GP) and encounters an error, it falls back to using the action scores from the previous round.

Usage

```
NonMonoPolicyEval(
  Policy,
  PricesTested,
  PurchaseDecisions,
  TestX,
  sigma2_y,
  BatchSize,
  GP,
  PrevAS
)
```

Arguments

Policy	The policy function to evaluate.
PricesTested	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
PurchaseDecisions	A vector of associated purchase decisions for the prices tested.
TestX	A set of prices (values between 0 and 1) to test.
sigma2_y	Observation noise variance (must be less than 0.25).
BatchSize	Number of consumers tested before an algorithm can update.
GP	Logical; if TRUE, the policy relies on Gaussian Processes.
PrevAS	A vector of previous action scores to use as a fallback in case of errors.

Value

A vector of action scores for the prices in 'TestX'.

OptimalHyperparameters *Optimal Hyperparameters*

Description

Optimizes hyperparameters for Gaussian Process regression using NLopt.

Usage

```
OptimalHyperparameters(TrainX, TrainY, sigma2_y)
```

Arguments

TrainX	Matrix of training points (m x d).
TrainY	Vector of training targets (m x 1).
sigma2_y	Observation noise variance (sigma_y^2).

Value

Vector of optimized hyperparameters (sigma_f and l).

PolicyEvaluation *Evaluate Policies for Pricing Experiments*

Description

This function evaluates monotonic or non-monotonic policies by executing them with the provided parameters. For monotonic policies, it includes mechanisms to handle timeout scenarios and fallback options.

Usage

```
PolicyEvaluation(
  Policy,
  PricesTested,
  PurchaseDecisions,
  TestX,
  Knots,
  sigma2_y,
  BatchSize,
  BasisFunctions,
  GP,
  Mono,
  PrevAS,
  Timeout = 5
)
```

Arguments

Policy	The policy function to evaluate.
PricesTested	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
PurchaseDecisions	A vector of associated purchase decisions for the prices tested.
TestX	A set of prices (values between 0 and 1) to test.
Knots	Knots for the basis functions used in monotonic policies.
sigma2_y	Observation noise variance (must be less than 0.25).
BatchSize	Number of consumers tested before an algorithm can update.
BasisFunctions	Basis functions for monotonic Gaussian Process regression.
GP	Logical; if TRUE, the policy relies on Gaussian Processes.
Mono	Logical; if TRUE, evaluates a monotonic policy.
PrevAS	A vector of previous action scores to use as a fallback in case of errors.
Timeout	Seconds allowed for each truncated-sampling attempt in the monotonic fallback chain (default 5).

Value

A vector of action scores for the prices in ‘TestX‘.

`PosteriorPrediction` *Posterior Prediction (Joint GP with Derivatives)*

Description

Computes the posterior mean and covariance for Gaussian Process regression with derivatives.

Usage

```
PosteriorPrediction(TrainX, TrainY, TestX, TestD, sigma_f, l, sigma2_y)
```

Arguments

`TrainX`	Vector of training points.
`TrainY`	Vector of training targets.
`TestX`	Vector of test points.
`TestD`	Vector of test points for derivative prediction.
`sigma_f`	Hyperparameter for kernel vertical scale.
`l`	Hyperparameter for kernel horizontal scale.
`sigma2_y`	Observation noise variance (sigma_y^2).

Value

List containing posterior mean vector and covariance matrix.

`PricingBandit` *Run a Pricing Bandit Experiment*

Description

Wrapper for running a single multi-armed bandit pricing experiment with any of the supported policies, selected by name. The willingness-to-pay (WTP) distribution is entirely user-specified: pass one draw per consumer via ‘valuations‘ (e.g. ‘rbeta(2500, 2, 9)‘, draws from an empirical CDF, or any other process). At each round the policy posts a price from ‘prices‘; the consumer purchases if and only if their valuation exceeds the price; the policy re-optimizes every ‘batch_size‘ consumers.

Available policies:

"UCB" Upper Confidence Bound on independent arms.

"TS" Thompson Sampling with Beta posteriors on independent arms.

"GP-UCB" Gaussian Process UCB: correlated arms through a GP demand curve.

"GP-TS" Gaussian Process Thompson Sampling.

"GP-UCB-M" Monotonic GP-UCB: demand draws are monotone everywhere by construction (basis-function reconstruction from derivatives constrained to be non-positive).

"GP-TS-M" Monotonic GP-TS (same construction).

Usage

```
PricingBandit(
  valuations,
  prices,
  policy = "GP-TS-M",
  batch_size = 10,
  num_knots = 11,
  hetero = FALSE,
  reset = NULL,
  timeout = 5
)
```

Arguments

valuations	Numeric vector of consumer valuations (WTP), one per consumer. Its length determines the number of iterations.
prices	Numeric vector of candidate prices (values between 0 and 1) - the arms.
policy	Character; one of ‘"UCB"‘, ‘"TS"‘, ‘"GP-UCB"‘, ‘"GP-TS"‘, ‘"GP-UCB-M"‘, ‘"GP-TS-M"‘.
batch_size	Number of consumers tested before the policy updates (default 10).
num_knots	Number of knots for the monotonic ("-M") variants (default 11, the value used throughout the paper). The default is deliberately independent of the number of arms: with many more knots the truncated sampler operates in a much higher-dimensional, near-degenerate space and can break down, so 11 is recommended even for dense price grids (e.g. 100 arms).
hetero	Logical; if ‘TRUE‘, use heterogeneous observation noise sampled each update via NoiseSample (default ‘FALSE‘).
reset	Optional; number of consumers after which the experiment history is wiped (for, e.g., time-varying demand). Default ‘NULL‘ (never reset).
timeout	Seconds allowed for each truncated-sampling attempt in the monotonic fallback chain before the next fallback is tried (default 5). Increase on slow machines to give the exact sampler more time; decrease to fail over to the cheaper approximations sooner.

Value

A data frame with columns ‘PricesTested‘ and ‘PurchaseDecisions‘ (one row per consumer), with attributes:

policy The policy name used.

diagnostics A list of fallback counters (see GetDiagnostics).

Examples

```
set.seed(1)
valuations <- rbeta(200, 2, 9)
out <- PricingBandit(valuations, prices = seq(10)/10, policy = "TS")
table(out$PricesTested)
```

`RBFKernel` *Kernel Functions*

Description

Collection of functions to compute RBF kernels and covariance matrices.

Calculates the RBF kernel between two points.

Usage

```
RBFKernel(x_i, x_j, sigma_f, l)
```

Arguments

`x_i`	A point with d dimensions.
`x_j`	A point with d dimensions.
`sigma_f`	Hyperparameter defining the vertical scale.
`l`	Hyperparameter defining the horizontal scale.

Value

Gaussian kernel value between two points.

`RBFKernel_01` *RBF Kernel (Point to Derivative)*

Description

Calculates the RBF kernel between a point and derivative.

Usage

```
RBFKernel_01(x_i, x_j, sigma_f, l)
```

Arguments

`x_i`	A point with d dimensions.
`x_j`	A point (derivative) with d dimensions.
`sigma_f`	Hyperparameter defining the vertical scale.
`l`	Hyperparameter defining the horizontal scale.

Value

Kernel value between the point and derivative.

`RBFKernel_11` *RBF Kernel (Derivative to Derivative)*

Description

Calculates the RBF kernel between two derivatives.

Usage

```
RBFKernel_11(x_i, x_j, sigma_f, l)
```

Arguments

`x_i`	A point (derivative) with d dimensions.
`x_j`	A point (derivative) with d dimensions.
`sigma_f`	Hyperparameter defining the vertical scale.
`l`	Hyperparameter defining the horizontal scale.

Value

Kernel value between the two derivatives.

`RBFKernel_All` *Generalized RBF Kernel*

Description

Computes RBF kernels for combinations of points and derivatives.

Usage

```
RBFKernel_All(x_i, x_j, d_i, d_j, sigma_f, l)
```

Arguments

`x_i`	A point with d dimensions.
`x_j`	A point with d dimensions.
`d_i`	Order of derivative for point x_i (0 or 1).
`d_j`	Order of derivative for point x_j (0 or 1).
`sigma_f`	Hyperparameter defining the vertical scale.
`l`	Hyperparameter defining the horizontal scale.

Value

Kernel value based on the derivative orders of the input points.

`ResetDiagnostics` *Reset Experiment Diagnostics*

Description

Resets all internal fallback counters and the monotonicity log. Called automatically at the start of `MABExperiment`; can be called manually before using policy functions directly.

Usage

```
ResetDiagnostics()
```

Value

Invisibly, `NULL`.

`TS` *Thompson Sampling (TS) Policy*

Description

This function implements the Thompson Sampling policy for pricing experiments. It calculates action scores by drawing from a Beta distribution based on aggregated data.

Usage

```
TS(PricesTested, PurchaseDecisions, TestX)
```

Arguments

`PricesTested`	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
`PurchaseDecisions`	A vector of associated purchase decisions for the prices tested.
`TestX`	A set of prices (values between 0 and 1) to test.

Value

A vector of action scores for the prices in 'TestX'.

Examples

```
PricesTested <- c(0.1, 0.2, 0.2, 0.9)
PurchaseDecisions <- c(1, 0, 1, 0)
TestX <- seq(10)/10
TS(PricesTested, PurchaseDecisions, TestX)
```

UCB *Bandit Policies for Pricing Experiments*

Description

Collection of functions implementing various policies for pricing experiments, including Upper Confidence Bound (UCB), Thompson Sampling (TS), Gaussian Process-based policies, and their monotonic extensions. Upper Confidence Bound (UCB) Policy

This function implements the UCB policy for pricing experiments. It calculates action scores based on aggregated data and selects prices to test using an upper confidence bound.

Usage

```
UCB(PricesTested, PurchaseDecisions, TestX)
```

Arguments

`PricesTested`	A vector of prices (values between 0 and 1) that have been tested so far in the experiment.
`PurchaseDecisions`	A vector of associated purchase decisions for the prices tested.
`TestX`	A set of prices (values between 0 and 1) to test.

Value

A vector of action scores for the prices in 'TestX'.

Examples

```
PricesTested <- c(0.1, 0.2, 0.2, 0.9)
PurchaseDecisions <- c(1, 0, 1, 0)
TestX <- seq(10)/10
UCB(PricesTested, PurchaseDecisions, TestX)
```

Index