Package ‘OOR’

August 23, 2023

Type Package

Title Optimistic Optimization in R

Version 0.1.4

Date 2023-08-22

Description Implementation of optimistic optimization methods for global optimization of deterministic or stochastic functions. The algorithms feature guarantees of the convergence to a global optimum. They require minimal assumptions on the (only local) smoothness, where the smoothness parameter does not need to be known. They are expected to be useful for the most difficult functions when we have no information on smoothness and the gradients are unknown or do not exist. Due to the weak assumptions, however, they can be mostly effective only in small dimensions, for example, for hyperparameter tuning.

License LGPL

Depends methods

URL https://github.com/mbinois/OOR

BugReports https://github.com/mbinois/OOR/issues

RoxygenNote 7.2.3

NeedsCompilation no

Author M. Binois [cre, aut, trl] (R port),
A. Carpentier [aut] (Matlab original),
J.-B. Grill [aut] (Python original),
R. Munos [aut] (Python and Matlab original),
M. Valko [aut, ctb] (Python and Matlab original)

Maintainer M. Binois <mickael.binois@inria.fr>

Repository CRAN

Date/Publication 2023-08-23 00:40:02 UTC

R topics documented:

`OOR` *Package OOR*

Description

This package implements optimistic optimization methods [1,2,3] for global optimization of deterministic or stochastic functions. The algorithms feature guarantees of the convergence to a global optimum. They require minimal assumptions on the (only local) smoothness, where the smoothness parameter does not need to be known. They are expected to be useful for the most difficult functions when we have no information on smoothness and the gradients are unknown or do not exist. Due to the weak assumptions, however, they can be mostly effective only in small dimensions, for example, for hyperparameter tuning [4].

Details

Important functions:
`StoSOO`
`POO`

Note

This package is based on the Matlab and Python implementations from the corresponding publications, available from the following webpage: `https://team.inria.fr/sequel/software/`.

References

[1] R. Munos (2011), Optimistic optimization of deterministic functions without the knowledge of its smoothness, *NIPS*, 783-791.

[2] M. Valko, A. Carpentier and R. Munos (2013), Stochastic Simultaneous Optimistic Optimization, *ICML*, 19-27 `https://inria.hal.science/hal-00789606`.

[3] J.-B. Grill, M. Valko and R. Munos (2015), Black-box optimization of noisy functions with unknown smoothness, *NIPS*, 667-675 `https://inria.hal.science/hal-01222915`.

[4] S. Samothrakis, D. Perz, S. Lucas (2013), Training gradient boosting machines using curve-fitting and information-theoretic features for causal direction detection, *NIPS Workshop on Causality*.

Examples

```
#---------------------------------------------------------------
# Example 1 : Deterministic optimization with SOO
#---------------------------------------------------------------
## Define objective
fun1 <- function(x) return(-guirland(x))

## Optimization
Sol1 <- StoSOO(par = NA, fn = fun1, nb_iter = 1000, control = list(type = "det", verbose = 1))

## Display objective function and solution fund
curve(fun1, n = 1001)
abline(v = Sol1$par, col = 'red')

#---------------------------------------------------------------
# Example 2 : Stochastic optimization with StoSOO
#---------------------------------------------------------------
set.seed(42)

## 2-dimensional noisy objective function, defined on [0, pi/4]^2
fun2 <- function(x){return(-sin1(x[1]) * sin1(1 - x[2]) + runif(1, min = -0.05, max = 0.05))}

## Optimizing
Sol2 <- StoSOO(par = rep(NA, 2), fn = fun2, upper = rep(pi/4, 2), nb_iter = 1000)

## Display solution
xgrid <- seq(0, pi/4, length.out = 101)
Xgrid <- expand.grid(xgrid, xgrid)
ref <- apply(Xgrid, 1, function(x){(-sin1(x[1]) * sin1(1 - x[2]))})
filled.contour(xgrid, xgrid, matrix(ref, 101), color.palette  = terrain.colors,
plot.axes = {axis(1); axis(2); points(Xgrid[which.min(ref),, drop = FALSE], pch = 21);
             points(Sol2$par[1], Sol2$par[2], pch = 13)})

## Not run:
#---------------------------------------------------------------
# Example 3 : Stochastic optimization with POO
#---------------------------------------------------------------
set.seed(10)
noise.level <- 0.05

## Define and display objective
fun3 <- function(x){return(double_sine(x) + runif(1, min = -noise.level, max = noise.level))}
xgrid <- seq(0, 1, length.out = 1000)
plot(xgrid, sapply(xgrid, double_sine), type = 'l', ylab = "double_sine(x)", xlab = 'x')

## Maximization
Sol3 <- POO(fun3, horizon = 1000, noise.level = noise.level)

## Display result
abline(v = Sol3$par)

## End(Not run)
```

plotStoSOO *Plot 2-D StoSOO result*

Description

Plot bivariate tree structure obtained when running StoSOO

Usage

```
plotStoSOO(
  sol,
  lower = rep(0, length(sol$par)),
  upper = rep(1, length(sol$par)),
  levels = NULL,
  add = FALSE,
  cpch = ".",
  lcols = 1,
  ylim = NULL
)
```

Arguments

`sol`	outcome of running StoSOO, with `control$light` set to FALSE
`lower, upper`	vectors of bounds on the variables.
`levels`	which levels to print. Default to all levels
`add`	if TRUE, use existing plot
`cpch`	points pch code for the centers
`lcols`	color at each level, or a single color for all levels (default)
`ylim`	vector of bounds, required in the 1d case

Examples

```
#------------------------------------------------------------
# Example 1 : Deterministic optimization with SOO, 1-dimensional function
#------------------------------------------------------------
## Define objective
fun1 <- function(x) return(-guirland(x))

## Optimization
Sol1 <- StoSOO(par = NA, fn = fun1, nb_iter = 1000,
  control = list(type = "det", verbose = 1, light = FALSE))

## Display objective function and solution found
curve(fun1, n = 1001, ylim = c(-1.3, 0))
abline(v = Sol1$par, col = 'red')
plotStoSOO(Sol1, ylim = c(-1.3, -1.1), add = TRUE)
```

```
#------------------------------------------------------------
# Example 2 : Deterministic optimization with SOO, 2-dimensional function
#------------------------------------------------------------
set.seed(42)

## 2-dimensional noiseless objective function, defined on [0, pi/4]^2
fun <- function(x){return(-sin1(x[1]) * sin1(1 - x[2]))}

## Optimizing
Sol <- StoSOO(par = rep(NA, 2), fn = fun, upper = rep(pi/4, 2), nb_iter = 1000,
  control = list(type = 'det', light = FALSE))

## Display solution
plotStoSOO(Sol, upper = rep(pi/4, 2))
```

POO *Parallel Optimistic Optimization*

Description

Global optimization of a blackbox function given a finite budget of noisy evaluations, via the Parallel Optimistic Optimization algorithm. The knowledge of the function's smoothness is not required.

Usage

```
POO(f, horizon = 100, noise.level, rhomax = 20, nu = 1)
```

Arguments

`f`	function to maximize.
`horizon`	maximum number of function evaluations.
`noise.level`	scalar bound on the noise value.
`rhomax`	number of equidistant `rho` values in [0,1], that are used by the corresponding HOO subroutines, see Details.
`nu`	scalar (> 0) assessing the complexity of the function, along with `rho` (see the near optimality definition in the reference below).

Details

Only 1-dimensional functions defined on [0, 1] are handled so far. POO uses Hierarchical Optimistic Optimisation (HOO) as a subroutine, whose number is set by `rhomax`. `POO` handles more difficult functions than `StoSOO`.

Value

Random point evaluated by the best HOO, in the form of a list with elements:

- par parameter value at this point,
- value noisy value at `par`,
- best_rho best `rho` value.

Author(s)

M. Binois (translation in R code), J.-B. Grill, M. Valko and R. Munos (Python code)

References

J.-B. Grill, M. Valko and R. Munos (2015), Black-box optimization of noisy functions with unknown smoothness, *NIPS*, 667-675 https://inria.hal.science/hal-01222915. Python code: https://team.inria.fr/sequel/software/POO/.

Examples

```
## Not run:
#------------------------------------------------------------
# Maximization with POO
#------------------------------------------------------------
set.seed(10)
noise.level <- 0.05

## Define and display objective
ftest <- function(x){return(double_sine(x) + runif(1, min = -noise.level, max = noise.level))}
xgrid <- seq(0, 1, length.out = 1000)
plot(xgrid, sapply(xgrid, double_sine), type = 'l', ylab = "double_sine(x)", xlab = 'x')

## Optimization
Sol <- POO(ftest, horizon = 1000, noise.level = noise.level)

## Display result
abline(v = Sol$par)

## End(Not run)
```

`StoSOO` *StoSOO and SOO algorithms*

Description

Global optimization of a blackbox function given a finite budget of noisy evaluations, via the Stochastic-Simultaneous Optimistic Optimisation algorithm. The deterministic-SOO method is available for noiseless observations. The knowledge of the function's smoothness is not required.

Usage

```
StoSOO(
  par,
  fn,
  ...,
  lower = rep(0, length(par)),
  upper = rep(1, length(par)),
  nb_iter,
  control = list(verbose = 0, type = "sto", max = FALSE, light = TRUE)
)
```

Arguments

`par`	vector with length defining the dimensionality of the optimization problem. Providing actual values of `par` is not necessary (NAs are just fine). Included primarily for compatibility with `optim`.
`fn`	scalar function to be minimized, with first argument to be optimized over.
`...`	optional additional arguments to `fn`.
`lower, upper`	vectors of bounds on the variables.
`nb_iter`	number of function evaluations allocated to optimization.
`control`	list of control parameters:

- verbose: verbosity level, either `0` (default), `1` or greater than `1`,
- type: either 'det' for optimizing a deterministic function or 'sto' for a stochastic one,
- k_max: maximum number of evaluations per leaf (default: from analysis),
- h_max: maximum depth of the tree (default: from analysis),
- delta: confidence (default: `1/sqrt(nb_iter)` - from analysis),
- light: set to `FALSE` to return the search tree,
- max: if `TRUE`, performs maximization.

Details

The optional `tree` element returned is a list, whose first element is the root node and the last element the deepest nodes. A each level, `x` provides the center(s), one per row, whose corresponding bounds are given by `x_min` and `x_max`. Then:

- `leaf` indicates if `x` is a leaf (1 if `TRUE`);
- `new` indicates if `x` has been sampled last;
- `sums` gives the sum of values at `x`;
- `bs` is for the upper bounds at `x`;
- `ks` is the number of evaluations at `x`;
- `values` stores the values evaluated as they come (mostly useful in the deterministic case)

Value

list with components:

- `par` best set of parameters (for a stochastic function, it corresponds to the minimum reached over the deepest unexpanded node),
- `value` value of `fn` at `par`,
- `tree` search tree built during the execution, not returned unless `control$light == TRUE`.

Author(s)

M. Binois (translation in R code), M. Valko, A. Carpentier, R. Munos (Matlab code)

References

R. Munos (2011), Optimistic optimization of deterministic functions without the knowledge of its smoothness, *NIPS*, 783-791.

M. Valko, A. Carpentier and R. Munos (2013), Stochastic Simultaneous Optimistic Optimization, *ICML*, 19-27 https://inria.hal.science/hal-00789606. Matlab code: https://team.inria.fr/sequel/software/StoSOO/.

P. Preux, R. Munos, M. Valko (2014), Bandits attack function optimization, *IEEE Congress on Evolutionary Computation (CEC)*, 2245-2252.

Examples

```
#------------------------------------------------------------
# Example 1 : Deterministic optimization with SOO
#------------------------------------------------------------
## Define objective
fun1 <- function(x) return(-guirland(x))

## Optimization
Sol1 <- StoSOO(par = NA, fn = fun1, nb_iter = 1000, control = list(type = "det", verbose = 1))

## Display objective function and solution found
curve(fun1, n = 1001)
abline(v = Sol1$par, col = 'red')

#------------------------------------------------------------
# Example 2 : Stochastic optimization with StoSOO
#------------------------------------------------------------
set.seed(42)

## Same objective function with uniform noise
fun2 <- function(x){return(fun1(x) + runif(1, min = -0.1, max = 0.1))}

## Optimization
Sol2 <- StoSOO(par = NA, fn = fun2, nb_iter = 1000, control = list(type = "sto", verbose = 1))
```

```
## Display solution
abline(v = Sol2$par, col = 'blue')

#------------------------------------------------------------
# Example 3 : Stochastic optimization with StoSOO, 2-dimensional function
#------------------------------------------------------------

set.seed(42)

## 2-dimensional noisy objective function, defined on [0, pi/4]^2
fun3 <- function(x){return(-sin1(x[1]) * sin1(1 - x[2]) + runif(1, min = -0.05, max = 0.05))}

## Optimizing
Sol3 <- StoSOO(par = rep(NA, 2), fn = fun3, upper = rep(pi/4, 2), nb_iter = 1000)

## Display solution
xgrid <- seq(0, pi/4, length.out = 101)
Xgrid <- expand.grid(xgrid, xgrid)
ref <- apply(Xgrid, 1, function(x){(-sin1(x[1]) * sin1(1 - x[2]))})
filled.contour(xgrid, xgrid, matrix(ref, 101), color.palette  = terrain.colors,
plot.axes = {axis(1); axis(2); points(Xgrid[which.min(ref),, drop = FALSE], pch = 21);
             points(Sol3$par[1],Sol3$par[2], pch = 13)})


#------------------------------------------------------------
# Example 4 : Deterministic optimization with StoSOO, 2-dimensional function with plots
#------------------------------------------------------------
set.seed(42)

## 2-dimensional noiseless objective function, defined on [0, pi/4]^2
fun4 <- function(x){return(-sin1(x[1]) * sin1(1 - x[2]))}

## Optimizing
Sol4 <- StoSOO(par = rep(NA, 2), fn = fun4, upper = rep(pi/4, 2), nb_iter = 1000,
  control = list(type = 'det', light = FALSE))

## Display solution
xgrid <- seq(0, pi/4, length.out = 101)
Xgrid <- expand.grid(xgrid, xgrid)
ref <- apply(Xgrid, 1, function(x){(-sin1(x[1]) * sin1(1 - x[2]))})
filled.contour(xgrid, xgrid, matrix(ref, 101), color.palette  = terrain.colors,
plot.axes = {axis(1); axis(2); plotStoSOO(Sol4, add = TRUE, upper = rep(pi/4, 2));
             points(Xgrid[which.min(ref),, drop = FALSE], pch = 21);
             points(Sol4$par[1], Sol4$par[2], pch = 13, col = 2)})
```

`Test functions` *Test functions of* `x`

Description

Several test functions of varying complexity are available. They are defined on [0,1].

Usage

```
guirland(x)

sin1(x)

difficult(x)

difficult2(x)

double_sine(x, rho1 = 0.3, rho2 = 0.8, tmax = 0.5)
```

Arguments

`x`	vector specifying the location where the function is to be evaluated.
`rho1, rho2, tmax`	additional parameters for double_sine.

Details

These test functions are translated from the Matlab and Python codes in the references.

References

M. Valko, A. Carpentier and R. Munos (2013), Stochastic Simultaneous Optimistic Optimization, *ICML*, 19-27 https://inria.hal.science/hal-00789606. Matlab code: https://team.inria.fr/sequel/software/StoSOO/.

J.-B. Grill, M. Valko and R. Munos (2015), Black-box optimization of noisy functions with unknown smoothness, *NIPS*, 667-675 https://inria.hal.science/hal-01222915. Python code: https://team.inria.fr/sequel/software/POO/.

Examples

```
par(mfrow = c(2,3))

curve(guirland, n = 501)
curve(sin1)
curve(difficult, xlim = c(1e-8, 1), n = 1001)
xgrid <- seq(0, 1, length.out = 500)
plot(xgrid, sapply(xgrid, difficult2), type = 'l', ylab = "difficult2(x)")
plot(xgrid, sapply(xgrid, double_sine), type = 'l', ylab = "double_sine(x) (default)")
double_sine2 <- function(x) double_sine(x, rho1 = 0.8, rho2 = 0.3)
plot(xgrid, sapply(xgrid, double_sine2), type = 'l', ylab = "double_sine(x) (modified)")

par(mfrow = c(1,1))
```

Index