

Package ‘atmopt’

May 7, 2026

Type Package

Title Analysis-of-Marginal-Tail-Means

Version 0.1.0

Author Simon Mak

Maintainer Simon Mak <smak6@gatech.edu>

Description

Provides functions for implementing the Analysis-of-marginal-Tail-Means (ATM) method, a robust optimization method for discrete black-box optimization. Technical details can be found in Mak and Wu (2018+) <[doi:10.48550/arXiv.1712.03589](https://doi.org/10.48550/arXiv.1712.03589)>. This work was supported by USARO grant W911NF-17-1-0007.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Imports DoE.base, hierNet, gtools

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2018-10-28 22:30:07 UTC

Contents

atmopt-package	2
atm.addpts	2
atm.init	5
atm.nextpts	5
atm.predict	7
atm.remlev	10
camel6	12
detpep10exp	13

Index	14
--------------	-----------

`atmopt-package`*Analysis-of-marginal-Tail-Means*

Description

The 'atmopt' package provides functions for implementing the ATM optimization method in Mak and Wu (2018+) <arXiv:1712.03589>.

Details

Package: atmopt
Type: Package
Version: 1.0
Date: 2018-10-19
License: GPL (>= 2)

The 'atmopt' package provides functions for implementing the Analysis-of-marginal-Tail-Means (ATM) method in Mak and Wu (2018+). ATM is a robust method for discrete, black-box optimization, where function evaluations are expensive and limited.

Author(s)

Simon Mak

Maintainer: Simon Mak <smak6@gatech.edu>

References

Mak, S. and Wu, C. F. J. (2018+). Analysis-of-marginal-Tail-Means (ATM): a robust method for discrete black-box optimization. Under review.

`atm.addpts`*Add new data for ATM*

Description

`atm.addpts` adds new data into an ATM object.

Usage

```
atm.addpts(atm.obj, des.new, obs.new)
```

Arguments

atm.obj Current ATM object.
 des.new Design matrix for new evaluations.
 obs.new Observations for new evaluations.

Examples

```
## Not run:
#####
# Example 1: detpep10exp (9-D)
#####
nfact <- 9 #number of factors
ntimes <- floor(nfact/3) #number of "repeats" for detpep10exp
lev <- 4 #number of levels
nlev <- rep(lev,nfact) #number of levels for each factor
nelim <- 3 #number of level eliminations
fn <- function(xx){detpep10exp(xx,ntimes,nlev)} #objective to minimize (assumed expensive)

#initialize objects
# (predicts & removes levels based on tuned ATM percentages)
fit.atm <- atm.init(nfact,nlev)
#initialize sel.min object
# (predicts minimum using smallest observed value & removes levels with largest minima)
fit.min <- atm.init(nfact,nlev)

#Run for nelim eliminations:
res.atm <- rep(NA,nelim) #for ATM results
res.min <- rep(NA,nelim) #for sel.min results
for (i in 1:nelim){

  # ATM updates:
  new.des <- atm.nextpts(fit.atm) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.atm <- atm.addpts(fit.atm,new.des,new.obs) #add data to object
  fit.atm <- atm.predict(fit.atm) #predict minimum setting
  idx.atm <- fit.atm$idx.opt
  res.atm[i] <- fn(idx.atm)
  fit.atm <- atm.remlev(fit.atm) #removes worst performing level

  # sel.min updates:
  new.des <- atm.nextpts(fit.min) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.min <- atm.addpts(fit.min,new.des,new.obs) #add data to object
  fit.min <- atm.predict(fit.min, alphas=rep(0,nfact)) #find setting with smallest observation
  idx.min <- fit.min$idx.opt
  res.min[i] <- fn(idx.min)
  #check: min(fit.min$obs.all)
  fit.min <- atm.remlev(fit.min) #removes worst performing level

}
```

```

res.atm
res.min

#conclusion: ATM finds better solutions by learning & exploiting additive structure

#####
# Example 2: camel6 (24-D)
#####
nfact <- 24 #number of factors
ntimes <- floor(nfact/2.0) #number of "repeats" for camel6
lev <- 4
nlev <- rep(lev,nfact) #number of levels for each factor
nelim <- 3 #number of level eliminations
fn <- function(xx){camel6(xx,ntimes,nlev)} #objective to minimize (assumed expensive)

#initialize objects
# (predicts & removes levels based on tuned ATM percentages)
fit.atm <- atm.init(nfact,nlev)
#initialize sel.min object
# (predicts minimum using smallest observed value & removes levels with largest minima)
fit.min <- atm.init(nfact,nlev)

#Run for nelim eliminations:
res.atm <- rep(NA,nelim) #for ATM results
res.min <- rep(NA,nelim) #for sel.min results
for (i in 1:nelim){

  # ATM updates:
  new.des <- atm.nextpts(fit.atm) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.atm <- atm.addpts(fit.atm,new.des,new.obs) #add data to object
  fit.atm <- atm.predict(fit.atm) #predict minimum setting
  idx.atm <- fit.atm$idx.opt
  res.atm[i] <- fn(idx.atm)
  fit.atm <- atm.remlev(fit.atm) #removes worst performing level

  # sel.min updates:
  new.des <- atm.nextpts(fit.min) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.min <- atm.addpts(fit.min,new.des,new.obs) #add data to object
  fit.min <- atm.predict(fit.min, alphas=rep(0,nfact)) #find setting with smallest observation
  idx.min <- fit.min$idx.opt
  res.min[i] <- fn(idx.min)
  #check: min(fit.min$obs.all)
  fit.min <- atm.remlev(fit.min) #removes worst performing level

}

res.atm
res.min

#conclusion: ATM finds better solutions by learning & exploiting additive structure

```

```
## End(Not run)
```

atm.init	<i>Initializing ATM object</i>
----------	--------------------------------

Description

atm.init initialize the ATM object to use for optimization.

Usage

```
atm.init(nfact, nlev)
```

Arguments

nfact	Number of factors to optimize.
nlev	A vector containing the number of levels for each factor.

Examples

```
nfact <- 9 #number of factors  
lev <- 4  
nlev <- rep(lev,nfact) #number of levels for each factor  
fit <- atm.init(nfact,nlev) #initialize ATM object
```

atm.nextpts	<i>Computes new design points for ATM</i>
-------------	---

Description

atm.nextpts computes new design points to evaluate using a randomized orthogonal array (OA).

Usage

```
atm.nextpts(atm.obj, reps=NULL)
```

Arguments

atm.obj	Current ATM object.
reps	Number of desired replications for OA.

Examples

```

## Not run:
#####
# Example 1: detpep10exp (9-D)
#####
nfact <- 9 #number of factors
ntimes <- floor(nfact/3) #number of "repeats" for detpep10exp
lev <- 4 #number of levels
nlev <- rep(lev,nfact) #number of levels for each factor
nelim <- 3 #number of level eliminations
fn <- function(xx){detpep10exp(xx,ntimes,nlev)} #objective to minimize (assumed expensive)

#initialize objects
# (predicts & removes levels based on tuned ATM percentages)
fit.atm <- atm.init(nfact,nlev)
#initialize sel.min object
# (predicts minimum using smallest observed value & removes levels with largest minima)
fit.min <- atm.init(nfact,nlev)

#Run for nelim eliminations:
res.atm <- rep(NA,nelim) #for ATM results
res.min <- rep(NA,nelim) #for sel.min results
for (i in 1:nelim){

  # ATM updates:
  new.des <- atm.nextpts(fit.atm) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.atm <- atm.addpts(fit.atm,new.des,new.obs) #add data to object
  fit.atm <- atm.predict(fit.atm) #predict minimum setting
  idx.atm <- fit.atm$idx.opt
  res.atm[i] <- fn(idx.atm)
  fit.atm <- atm.remlev(fit.atm) #removes worst performing level

  # sel.min updates:
  new.des <- atm.nextpts(fit.min) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.min <- atm.addpts(fit.min,new.des,new.obs) #add data to object
  fit.min <- atm.predict(fit.min, alphas=rep(0,nfact)) #find setting with smallest observation
  idx.min <- fit.min$idx.opt
  res.min[i] <- fn(idx.min)
  #check: min(fit.min$obs.all)
  fit.min <- atm.remlev(fit.min) #removes worst performing level

}

res.atm
res.min

#conclusion: ATM finds better solutions by learning & exploiting additive structure

#####
# Example 2: camel6 (24-D)

```

```
#####
nfact <- 24 #number of factors
ntimes <- floor(nfact/2.0) #number of "repeats" for camel6
lev <- 4
nlev <- rep(lev,nfact) #number of levels for each factor
nelim <- 3 #number of level eliminations
fn <- function(xx){camel6(xx,ntimes,nlev)} #objective to minimize (assumed expensive)

#initialize objects
# (predicts & removes levels based on tuned ATM percentages)
fit.atm <- atm.init(nfact,nlev)
#initialize sel.min object
# (predicts minimum using smallest observed value & removes levels with largest minima)
fit.min <- atm.init(nfact,nlev)

#Run for nelim eliminations:
res.atm <- rep(NA,nelim) #for ATM results
res.min <- rep(NA,nelim) #for sel.min results
for (i in 1:nelim){

  # ATM updates:
  new.des <- atm.nextpts(fit.atm) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.atm <- atm.addpts(fit.atm,new.des,new.obs) #add data to object
  fit.atm <- atm.predict(fit.atm) #predict minimum setting
  idx.atm <- fit.atm$idx.opt
  res.atm[i] <- fn(idx.atm)
  fit.atm <- atm.remlev(fit.atm) #removes worst performing level

  # sel.min updates:
  new.des <- atm.nextpts(fit.min) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.min <- atm.addpts(fit.min,new.des,new.obs) #add data to object
  fit.min <- atm.predict(fit.min, alphas=rep(0,nfact)) #find setting with smallest observation
  idx.min <- fit.min$idx.opt
  res.min[i] <- fn(idx.min)
  #check: min(fit.min$obs.all)
  fit.min <- atm.remlev(fit.min) #removes worst performing level

}

res.atm
res.min

#conclusion: ATM finds better solutions by learning & exploiting additive structure

## End(Not run)
```

Description

atm.init predicts the minimum setting for an ATM object.

Usage

```
atm.predict(atm.obj, alphas=NULL, ntimes=1, nsub=100, prob.am=0.5, prob.pw=1.0, reps=NULL)
```

Arguments

atm.obj	Current ATM object.
alphas	A p-vector for ATM percentiles. NULL if tuned from data.
ntimes	Number of resamples for tuning ATM percentages.
nsub	Number of candidate percentiles to consider.
prob.am	In case of ties in percentage estimation, the probability of choosing marginal means (if optimal) for minimization.
prob.pw	In case of ties in percentage estimation, probability of picking-the-winner (if optimal) for minimization.
reps	Number of replications for internal OA in tuning ATM percentages.

Examples

```
## Not run:
#####
# Example 1: detpep10exp (9-D)
#####
nfact <- 9 #number of factors
ntimes <- floor(nfact/3) #number of "repeats" for detpep10exp
lev <- 4 #number of levels
nlev <- rep(lev,nfact) #number of levels for each factor
nelim <- 3 #number of level eliminations
fn <- function(xx){detpep10exp(xx,ntimes,nlev)} #objective to minimize (assumed expensive)

#initialize objects
# (predicts & removes levels based on tuned ATM percentages)
fit.atm <- atm.init(nfact,nlev)
#initialize sel.min object
# (predicts minimum using smallest observed value & removes levels with largest minima)
fit.min <- atm.init(nfact,nlev)

#Run for nelim eliminations:
res.atm <- rep(NA,nelim) #for ATM results
res.min <- rep(NA,nelim) #for sel.min results
for (i in 1:nelim){

  # ATM updates:
  new.des <- atm.nextpts(fit.atm) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.atm <- atm.addpts(fit.atm,new.des,new.obs) #add data to object
  fit.atm <- atm.predict(fit.atm) #predict minimum setting
```

```

    idx.atm <- fit.atm$idx.opt
    res.atm[i] <- fn(idx.atm)
    fit.atm <- atm.remlev(fit.atm) #removes worst performing level

    # sel.min updates:
    new.des <- atm.nextpts(fit.min) #get design points
    new.obs <- apply(new.des,1,fn) #sample function
    fit.min <- atm.addpts(fit.min,new.des,new.obs) #add data to object
    fit.min <- atm.predict(fit.min, alphas=rep(0,nfact)) #find setting with smallest observation
    idx.min <- fit.min$idx.opt
    res.min[i] <- fn(idx.min)
    #check: min(fit.min$obs.all)
    fit.min <- atm.remlev(fit.min) #removes worst performing level

}

res.atm
res.min

#conclusion: ATM finds better solutions by learning & exploiting additive structure

#####
# Example 2: camel6 (24-D)
#####
nfact <- 24 #number of factors
ntimes <- floor(nfact/2.0) #number of "repeats" for camel6
lev <- 4
nlev <- rep(lev,nfact) #number of levels for each factor
nelim <- 3 #number of level eliminations
fn <- function(xx){camel6(xx,ntimes,nlev)} #objective to minimize (assumed expensive)

#initialize objects
# (predicts & removes levels based on tuned ATM percentages)
fit.atm <- atm.init(nfact,nlev)
#initialize sel.min object
# (predicts minimum using smallest observed value & removes levels with largest minima)
fit.min <- atm.init(nfact,nlev)

#Run for nelim eliminations:
res.atm <- rep(NA,nelim) #for ATM results
res.min <- rep(NA,nelim) #for sel.min results
for (i in 1:nelim){

  # ATM updates:
  new.des <- atm.nextpts(fit.atm) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.atm <- atm.addpts(fit.atm,new.des,new.obs) #add data to object
  fit.atm <- atm.predict(fit.atm) #predict minimum setting
  idx.atm <- fit.atm$idx.opt
  res.atm[i] <- fn(idx.atm)
  fit.atm <- atm.remlev(fit.atm) #removes worst performing level

  # sel.min updates:

```

```

new.des <- atm.nextpts(fit.min) #get design points
new.obs <- apply(new.des,1,fn) #sample function
fit.min <- atm.addpts(fit.min,new.des,new.obs) #add data to object
fit.min <- atm.predict(fit.min, alphas=rep(0,nfact)) #find setting with smallest observation
idx.min <- fit.min$idx.opt
res.min[i] <- fn(idx.min)
#check: min(fit.min$obs.all)
fit.min <- atm.remlev(fit.min) #removes worst performing level

}

res.atm
res.min

#conclusion: ATM finds better solutions by learning & exploiting additive structure

## End(Not run)

```

atm.remlev

Removing worst levels for ATM

Description

atm.remlev removes the worst (i.e., highest) levels from each factor in ATM.

Usage

```
atm.remlev(atm.obj)
```

Arguments

atm.obj Current ATM object.

Examples

```

## Not run:
#####
# Example 1: detpep10exp (9-D)
#####
nfact <- 9 #number of factors
ntimes <- floor(nfact/3) #number of "repeats" for detpep10exp
lev <- 4 #number of levels
nlev <- rep(lev,nfact) #number of levels for each factor
nelim <- 3 #number of level eliminations
fn <- function(xx){detpep10exp(xx,ntimes,nlev)} #objective to minimize (assumed expensive)

#initialize objects
# (predicts & removes levels based on tuned ATM percentages)
fit.atm <- atm.init(nfact,nlev)
#initialize sel.min object

```

```

# (predicts minimum using smallest observed value & removes levels with largest minima)
fit.min <- atm.init(nfact,nlev)

#Run for nelim eliminations:
res.atm <- rep(NA,nelim) #for ATM results
res.min <- rep(NA,nelim) #for sel.min results
for (i in 1:nelim){

  # ATM updates:
  new.des <- atm.nextpts(fit.atm) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.atm <- atm.addpts(fit.atm,new.des,new.obs) #add data to object
  fit.atm <- atm.predict(fit.atm) #predict minimum setting
  idx.atm <- fit.atm$idx.opt
  res.atm[i] <- fn(idx.atm)
  fit.atm <- atm.remlev(fit.atm) #removes worst performing level

  # sel.min updates:
  new.des <- atm.nextpts(fit.min) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.min <- atm.addpts(fit.min,new.des,new.obs) #add data to object
  fit.min <- atm.predict(fit.min, alphas=rep(0,nfact)) #find setting with smallest observation
  idx.min <- fit.min$idx.opt
  res.min[i] <- fn(idx.min)
  #check: min(fit.min$obs.all)
  fit.min <- atm.remlev(fit.min) #removes worst performing level

}

res.atm
res.min

#conclusion: ATM finds better solutions by learning & exploiting additive structure

#####
# Example 2: camel6 (24-D)
#####
nfact <- 24 #number of factors
ntimes <- floor(nfact/2.0) #number of "repeats" for camel6
lev <- 4
nlev <- rep(lev,nfact) #number of levels for each factor
nelim <- 3 #number of level eliminations
fn <- function(xx){camel6(xx,ntimes,nlev)} #objective to minimize (assumed expensive)

#initialize objects
# (predicts & removes levels based on tuned ATM percentages)
fit.atm <- atm.init(nfact,nlev)
#initialize sel.min object
# (predicts minimum using smallest observed value & removes levels with largest minima)
fit.min <- atm.init(nfact,nlev)

#Run for nelim eliminations:
res.atm <- rep(NA,nelim) #for ATM results

```

```

res.min <- rep(NA,nelim) #for sel.min results
for (i in 1:nelim){

  # ATM updates:
  new.des <- atm.nextpts(fit.atm) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.atm <- atm.addpts(fit.atm,new.des,new.obs) #add data to object
  fit.atm <- atm.predict(fit.atm) #predict minimum setting
  idx.atm <- fit.atm$idx.opt
  res.atm[i] <- fn(idx.atm)
  fit.atm <- atm.remlev(fit.atm) #removes worst performing level

  # sel.min updates:
  new.des <- atm.nextpts(fit.min) #get design points
  new.obs <- apply(new.des,1,fn) #sample function
  fit.min <- atm.addpts(fit.min,new.des,new.obs) #add data to object
  fit.min <- atm.predict(fit.min, alphas=rep(0,nfact)) #find setting with smallest observation
  idx.min <- fit.min$idx.opt
  res.min[i] <- fn(idx.min)
  #check: min(fit.min$obs.all)
  fit.min <- atm.remlev(fit.min) #removes worst performing level

}

res.atm
res.min

#conclusion: ATM finds better solutions by learning & exploiting additive structure

## End(Not run)

```

 camel6

Six-hump discrete test function

Description

A discrete test function constructed from the six-hump camel function in Ali et al. (2005).

Usage

```
camel6(xx,ntimes,nlev)
```

Arguments

xx	A p -vector for input factors.
ntimes	Number of duplications for the function (base function is 2D).
nlev	A p -vector corresponding to the number of levels for each factor(discretized on equally-spaced intervals).

Examples

```
xx <- c(1,2,1,2,1,2) #input factors
nlev <- rep(4,length(xx)) #number of levels for each factor
ntimes <- length(xx)/2 #base function is in 2D, so duplicate 3 times
camel6(xx,ntimes,nlev)
```

detpep10exp

DetPep10Exp discrete test function

Description

A discrete test function constructed from the modified exponential function in Dette and Pepelyshev (2010).

Usage

```
detpep10exp(xx,ntimes,nlev)
```

Arguments

xx	A p -vector for input factors.
ntimes	Number of duplications for the function (base function is 3D).
nlev	A p -vector corresponding to the number of levels for each factor(discretized on equally-spaced intervals).

Examples

```
xx <- c(1,2,1,2,1,2) #input factors
nlev <- rep(4,length(xx)) #number of levels for each factor
ntimes <- length(xx)/3 #base function is in 2D, so duplicate 2 times
detpep10exp(xx,ntimes,nlev)
```

Index

* **Robust optimization, discrete
optimization, black-box
optimization**
atmopt-package, [2](#)

atm.addpts, [2](#)

atm.init, [5](#)

atm.nextpts, [5](#)

atm.predict, [7](#)

atm.remlev, [10](#)

atmopt (atmopt-package), [2](#)

atmopt-package, [2](#)

camel6, [12](#)

detpep10exp, [13](#)