Package 'graDiEnt'

May 8, 2026

Title Stochastic Quasi-Gradient Differential Evolution Optimization

Version 1.0.1

Description An optim-style implementation of the Stochastic Quasi-Gradient Differential Evolution (SQG-DE) optimization algorithm first published by Sala, Baldanzini, and Pierini (2018; <doi:10.1007/978-3-319-72926-8_27>). This optimization algorithm fuses the robustness of the population-based global optimization algorithm ``Differential Evolution'' with the efficiency of gradient-based optimization. The derivative-free algorithm uses population members to build stochastic gradient estimates, without any additional objective function evaluations. Sala, Baldanzini, and Pierini argue this algorithm is useful for 'difficult optimization problems under a tight function evaluation budget.' This package can run SQG-DE in parallel and sequentially.

License MIT + file LICENSE

URL https://github.com/bmgaldo/graDiEnt

BugReports https://github.com/bmgaldo/graDiEnt

Encoding UTF-8

Depends R (>= 3.5.0)

Imports stats, doParallel

RoxygenNote 7.1.2

NeedsCompilation no

Author Brendan Matthew Galdo [aut, cre] (ORCID: <https://orcid.org/0000-0002-1279-3859>)

Maintainer Brendan Matthew Galdo <Brendan.m.galdo@gmail.com>

Repository CRAN

Date/Publication 2022-05-10 16:40:02 UTC

Contents

GetAlgoParams *GetAlgoParams*

Description

Get control parameters for optim_SQGDE function.

Usage

```
GetAlgoParams(
  n_params,
  n_particles = NULL,
  n_diff = 2,
  n_iter = 1000,
  init_sd = 0.01,
  init_center = 0,
  n_cores_use = 1,
  step_size = NULL,
  jitter_size = 1e-06,
  crossover_rate = 1,
  parallel_type = "none",
  return_trace = FALSE,
  thin = 1,
  purify = Inf,
  adapt_scheme = NULL,
  give_up_init = 100,
  stop_check = 10,
  stop_tol = 1e-04,
  converge_crit = "stdev"
)
```

Arguments

n_params	The number of parameters estimated/optimized, this integer value NEEDS to be specified.
n_particles	The number of particles (population size), 3*n_params is the default value.
n_diff	The number of mutually exclusive vector pairs to stochastically approximate the gradient.
n_iter	The number of iterations to run the algorithm, 1000 is default.
init_sd	A positive scalar or n_params-dimensional numeric vector, determines the standard deviation of the Gaussian initialization distribution. The default value is 0.01.
init_center	A scalar or n_params-dimensional numeric vector, determines the mean of the Gaussian initialization distribution. The default value is 0.
n_cores_use	An integer specifying the number of cores used when using parallelization. The default value is 1.

`step_size` A positive scalar, jump size or "F" in the DE crossover step notation. The default value is 2.38/sqrt(2*n_params).

`jitter_size` A positive scalar that determines the jitter (noise) size. Noise is added during adaption step from Uniform(-jitter_size,jitter_size) distribution. 1e-6 is the default value. Set to 0 to turn off jitter.

`crossover_rate` A numeric scalar on the interval (0,1]. Determines the probability a parameter on a chain is updated on a given crossover step, sampled from a Bernoulli distribution. The default value is 1.

`parallel_type` A string specifying parallelization type. 'none','FORK', or 'PSOCK' are valid values. 'none' is default value. 'FORK' does not work with Windows OS.

`return_trace` A boolean, if true, the function returns particle trajectories. This is helpful for assessing convergence or debugging model code. The trace will be an iteration/thin x n_particles x n_params array containing parameter values and an iteration/thin x n_particles array containing particle weights.

`thin` A positive integer. Only every 'thin'-th iteration will be stored in memory. The default value is 1. Increasing thin will reduce the memory required when running the algorithim for longer.

`purify` A positive integer. On every 'purify'-th iteration the particle weights are recomputed. This is useful if the objective function is stochastic/noisy. If the objective function is deterministic, this computation is redundant. Purify is set to Inf by default, disabling it.

`adapt_scheme` A string that must be 'rand','current', or 'best' that determines the DE adaption scheme/strategy. 'rand' uses rand/1/bin DE-like scheme where a random particle and the particle-based quasi-gradient approximation are used to generate proposal updates for a given particle. 'current' uses current/1/bin, and 'best' uses best/1/bin which follow an analogous adaption scheme to rand. 'rand' is the default value.

`give_up_init` An integer for how many failed initialization attempts before stopping the optimization routine. 100 is the default value.

`stop_check` An integer for how often to check the convergence criterion. The default is 10 iterations.

`stop_tol` A convergence metric must be less than value to be labeled as converged. The default is 1e-4.

`converge_crit` A string denoting the convergence metric used, valid metrics are 'stdev' (standard deviation of population weight in the last stop_check iterations) and 'percent' (percent improvement in median particle weight in the last stop_check iterations). 'stdev' is the default.

Value

A list of control parameters for the optim_SQGDE function.

optim_SQGDE *optim_SQGDE*

Description

Runs Stochastic Quasi-Gradient Differential Evolution (SQG-DE; Sala, Baldanzini, and Pierini, 2018) to minimize an objective function f(x). To maximize a function f(x), simply pass g(x)=-f(x) to ObjFun argument.

Usage

```
optim_SQGDE(ObjFun, control_params = GetAlgoParams(), ...)
```

Arguments

ObjFun	A scalar-returning function to minimize whose first argument is a real-valued n_params-dimensional vector.
control_params	control parameters for SQG-DE algo. see GetAlgoParams function documentation for more details. The only argument you NEED to pass here is n_params.
...	additional arguments to pass ObjFun.

Value

list containing solution and it's corresponding weight (i.e. f(solution)).

Examples

```
##############
# Maximum Likelihood Example
##############

# simulate from model
dataExample=matrix(rnorm(1000,c(-1,1,0,1),c(1,1,1,1)),ncol=4,byrow = TRUE)

# list parameter names
param_names_example=c("mu_1","mu_2","mu_3","mu_4")

# negative log likelihood
ExampleObjFun=function(x,data,param_names){
  out=0

  names(x) <- param_names

  # log likelihoods
  out=out+sum(dnorm(data[,1],x["mu_1"],sd=1,log=TRUE))
  out=out+sum(dnorm(data[,2],x["mu_2"],sd=1,log=TRUE))
  out=out+sum(dnorm(data[,3],x["mu_3"],sd=1,log=TRUE))
  out=out+sum(dnorm(data[,4],x["mu_4"],sd=1,log=TRUE))
```

```
  return(out*-1)
}

########################
# run optimization
out <- optim_SQGDE(ObjFun = ExampleObjFun,
                   control_params = GetAlgoParams(n_params = length(param_names_example),
                                                  n_iter = 250,
                                                   n_particles = 12,
                                                   n_diff = 2,
                                                   return_trace = TRUE),
                    data = dataExample,
                    param_names = param_names_example)
old_par <- par() # save graphic state for user
# plot particle trajectory

par(mfrow=c(2,2))
matplot(out$particles_trace[,,1],type='l')
matplot(out$particles_trace[,,2],type='l')
matplot(out$particles_trace[,,3],type='l')
matplot(out$particles_trace[,,4],type='l')

#SQG DE solution
out$solution

#analytic solution
apply(dataExample, 2, mean)

par(old_par) # restore user graphic state
```

Index