Package ‘MEIGOR’

January 30, 2017

Type Package
Title MEIGO - MEtaheuristics for bIoinformatics Global Optimization
Version 1.8.0
Date 2012-03-13
Author Jose Egea, David Henriques, Alexandre Fdez. Villaverde, Thomas Cokelaer
Maintainer Jose Egea <josea.egea@upct.es>
Depends Rsolnp, snowfall, CNORode, deSolve
Suggests CellNOptR
Description Global Optimization
License GPL-3
LazyLoad yes
biocViews SystemsBiology
NeedsCompilation no

R topics documented:

MEIGOR-package .................................................. 2
essR-package ..................................................... 3
BayesFit ............................................................ 4
CeSSR .............................................................. 4
CeVNSR ............................................................ 6
cnolist .............................................................. 8
cur_params .......................................................... 8
dhc ................................................................. 11
essR ................................................................. 11
essR_multistart .................................................... 16
eucl_dist ............................................................ 16
ith_essR ............................................................ 16
ith_VNSR ........................................................... 17
MEIGO ............................................................. 17
model ............................................................... 18
nls_fobj ............................................................ 18
optim_fobj .......................................................... 18
paramsOpt .......................................................... 18
runBayesFit ......................................................... 19
MEIGOR-package

Description

A global optimization package containing several algorithms such as a scatter search implementation and variable neighborhood search (plus cooperative multicore/multimachine implementations of these) and dynamic hill climbing.

Details

<table>
<thead>
<tr>
<th>Package:</th>
<th>MEIGOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>0.99.0</td>
</tr>
<tr>
<td>Date:</td>
<td>2012-09-10</td>
</tr>
<tr>
<td>License:</td>
<td>GPLv3</td>
</tr>
<tr>
<td>LazyLoad:</td>
<td>yes</td>
</tr>
</tbody>
</table>

Author(s)

Jose Egea David Henriques Thomas Cokelaer Alejandro F. Villaverde Julio R. Banga Julio Saez-Rodriguez

Maintainer: Jose Egea <josea.egea@upct.es>

References


See Also

essR essR

Description

essR attempts to solve problems of the form:
\[
\begin{align*}
\min & \quad F(x) \\
\text{subject to:} & \quad x \\
\text{ceq}(x) &= 0 \text{ (equality constraints)} \\
\text{c}_L &\leq c(x) \leq \text{c}_U \text{ (inequality constraints)} \\
x_L &\leq x \leq x_U \text{ (bounds on the decision variables)}
\end{align*}
\]

Constraint functions, if applicable, must be declared in the same script as the objective function as a second output argument, e.g.:

```
myfunction <- function(x){
calculate fx - scalar containing the objective function value
calculate gx - vector (or empty) containing the constraints values
return(list(fx,gx))
}
```

Details

- **Package:** MEIGOR
- **Type:** Package
- **Version:** 0.99.6
- **Date:** 2012-02-04
- **License:** GPL-3
- **LazyLoad:** yes

Author(s)

Jose Egea David Henriques Thomas Cokelaer Alejandro F. Villaverde Julio R. Banga Julio Saez-Rodriguez

Maintainer: Jose Egea <josea.egea@upct.es>

References


See Also

essR

BayesFit

Description

BayesFit

CeSSR  
Global optimization algorithm for MINLPs based on Scatter Search using a Cooperative Strategy

Description

CeSSR attempts to solve problems of the form:

\[ \min f(x, p_1, p_2, \ldots, p_n) \]

subject to:

\[ c_e = 0 \]

\[ c_L \leq c(x) \leq c_U \]

\[ x_L \leq x \leq x_U \]

Usage

CeSSR(problem, opts, max_eval = Inf, max_time = Inf, n_iter, is_parallel = TRUE, type = "SOCKS", global_save_list = NULL, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>problem</td>
<td>List containing problem settings.</td>
</tr>
<tr>
<td>opts</td>
<td>A list of n_threads lists containing options for each cooperative instance of essR.</td>
</tr>
<tr>
<td>max_eval</td>
<td>Maximum bumber of evaluations. Default is Inf.</td>
</tr>
<tr>
<td>max_time</td>
<td>Maximum time, default is Inf.</td>
</tr>
<tr>
<td>n_iter</td>
<td>Number of cooperative iterations. Default is 0 which is the same as running multiple single thread (as many as n_cpus) optimization runs.</td>
</tr>
<tr>
<td>is_parallel</td>
<td>Default is TRUE. Sometimes this it is useful to use as FALSE for debugging.</td>
</tr>
<tr>
<td>type</td>
<td>Choose between &quot;SOCKS&quot; and &quot;MPI&quot;. Default is &quot;SOCKS&quot; (socket-connection). If you are using &quot;SOCKS&quot; option and you want to run multiple cpus in different machines you must specify the adress of each machine in hosts. &quot;MPI&quot; mode requires you to have Rmpi installed.</td>
</tr>
<tr>
<td>global_save_list</td>
<td>Specify the names of global variables to be exported.</td>
</tr>
<tr>
<td>...</td>
<td>Additional variables.</td>
</tr>
</tbody>
</table>
Details

Check essR documentation for more information about the input arguments.

Value

- **f_mean**: Vector with size of n_iter+1 containing the mean value of the objective function in each iteration.
- **f_sd**: Vector with size of n_iter+1 containing the standard deviation value of the objective function in each iteration.
- **fbest**: Vector with size of n_iter+1 containing the best value of the objective function in each iteration.
- **iteration_res**: A list containing the results from every eSSR instance initialized. It follows the format: results$iteration_res[[iteration+1]][[thread_number]]. See also essR
- **numeval**: Vector with size of n_iter+1 containing the number objective function evaluations at the end of each iteration.
- **time**: Vector with size of n_iter+1 containing the time spent at the end of an iteration.
- **x_sd**: A list containing the standard deviation of decision each variable at the end of an iteration. It follows the format: results$iteration_res[[iteration+1]][[thread_number]]
- **xbest**: A list containing the best set of decision variables found and the end of each iteration.

See Also

essR

Examples

```r
rosen10<-function(x){
  f<-0;
  n=length(x);
  for (i in 1:(n-1)){
    f <- f + 100*(x[i]^2 - x[i+1])^2 + (x[i]-1)^2;
  }
  return(f)
}

nvar=20;
problem<-list(f=rosen10, x_L=rep(-1000,nvar), x_U=rep(1000,nvar));

#Set 1 nodes and 2 cpu’s per node
n_nodes=1;
n_cpus_per_node=3;

#Set different values for dim_refset, bal and n2 for each of the 10 cpu’s to be used
dim1 = 23;   bal1 = 0;   n2_1 = 0;
dim2 = 33;   bal2 = 0;   n2_2 = 0;
dim3 = 46;   bal3 = 0;   n2_3 = 2;
dim4 = 56;   bal4 = 0;   n2_4 = 4;
dim5 = 72;   bal5 = 0.25; n2_5 = 7;
dim6 = 72;   bal6 = 0.25; n2_6 = 10;
dim7 = 88;   bal7 = 0.25; n2_7 = 15;
dim8 = 101;  bal8 = 0.5;  n2_8 = 20;
```
dim9 = 111; bal9 = 0.25; n2_9 = 50;
dim10 = 123; bal10 = 0.25; n2_10 = 100;

opts_dim=c(dim1,dim2,dim3,dim4,dim5,dim6,dim7,dim8,dim9,dim10);
opts_bal=c(bal1,bal2,bal3,bal4,bal5,bal6,bal7,bal8,bal9,bal10);
opts_n2=c(n2_1,n2_2,n2_3,n2_4,n2_5,n2_6,n2_7,n2_8,n2_9,n2_10);
D=10;

#Initialize counter and options
counter=0;
opts=list();
hosts=c();

for(i in 1:n_nodes){
    for(j in 1:n_cpus_per_node){
        counter=counter+1;
        
        #Set the name of every thread
        if(i<10)hosts=c(hosts,paste(’/quotesingle.Var node0’/quotesingle.Var i,sep=””));
        if(i>=10 & i<100)hosts=c(hosts,paste(’/quotesingle.Var node’/quotesingle.Var i,sep=””));
        
        opts[[counter]]=list();
        
        #Set specific options for each thread
        opts[[counter]]$local_balance = opts_bal[counter];
        opts[[counter]]$dim_refset = opts_dim[counter];
        opts[[counter]]$local_n2 = opts_n2[counter];
        
        #Set common options for each thread
        opts[[counter]]$maxeval = 10000;
        opts[[counter]]$local_solver = ”dhec”;
        
        #Options not set will take default values for every thread
    }
}

#Set the address of each machine, defined inside the ’for’ loop
opts$hosts=c(’localhost’,’localhost’,’localhost’);

#Do not define the additional options for cooperative methods (e.g., ce_maxtime, ce_isparallel, etc..)
#They will take their default values
opts$ce_niter=2;
opts$ce_type=’SOCKS’;
opts$ce_isparallel=TRUE;

#Call the solver
Results<-MEIGO(problem,opts,algorithm=”CeSSR”)

CeVNSR

Global optimization algorithm for MINLPs based on VNS using a Cooperative Strategy
Description

Solves optimization problems with integer variables. Using several cooperative instances of VNS.

Usage

CeVNSR( problem, opts, max_eval = Inf, max_time = Inf,
        n_iter = 1, is_parallel = TRUE, type = "SOCKS",
        global_save_list = NULL, ...)

Arguments

- **problem**: List containing problem settings.
- **opts**: A list of n_threads lists containing options for each cooperative instance of essR.
- **max_eval**: Maximum number of evaluations. Default is Inf.
- **max_time**: Maximum time, default is Inf.
- **n_iter**: Number of cooperative iterations. Default is 0 which is the same as running multiple single thread (as many as n_cpus) optimization runs.
- **is_parallel**: Default is TRUE. Sometimes this it is useful to use as FALSE for debugging.
- **type**: Choose between "SOCKS" and "MPI". Default is "SOCKS" (socket-connection). If you are using "SOCKS" option and you want to run multiple cpus in different machines you must specify the adress of each machine in host.
- **global_save_list**: Specify the names of global variables to be exported.
- **...**: Additional variables.

Details

- **problem[[ith_thread]]**=VNS_problem; **opts[[ith_thread]]**=VNS_opts;
- VNS_problem and VNS_opts correspond to lists as seen in the rvnds_hamming documentation.

Value

- **f_mean**: Vector with size of n_iter+1 containing the mean value of the objective function in each iteration.
- **f_sd**: Vector with size of n_iter+1 containing the standard deviation value of the objective function in each iteration.
- **fbest**: Vector with size of n_iter+1 containing the best value of the objective function in each iteration.
- **iteration_res**: A list containing the results from every VNS instance initialized. It follows the format: results$iteration_res[[iteration+1]][[thread_number]].
- **numeval**: Vector with size of n_iter+1 containing the number objective function evaluations at the end of each iteration.
- **time**: Vector with size of n_iter+1 containing the time spent at the end of an iteration.
- **x_sd**: A list containing the standard deviation of decision each variable at the end of an iteration. It follows the format: results$iteration_res[[iteration+1]][[thread_number]]
- **xbest**: A list containing the best set of decision variables found and the end of each iteration.
See Also

rvnds_hamming MEIGO

Examples

```r
c_f<function(x){
f<-0;
n=length(x);
for (i in 1:(n-1)){
f <- f + 100*(x[i]^2 - x[i+1])^2 + (x[i]-1)^2;
}
return(f)
}
nvar=20;
problem<-list(f=rosen10, x_L=rep(-1000,nvar), x_U=rep(1000,nvar))
opts=list();
 opts[[1]]=list(use_local=1,aggr=1,local_search=1,decomp=1,maxdist=0.8,maxeval=2000);
 opts[[2]]=list(use_local=1,aggr=0,local_search=2,decomp=0,maxdist=0.5,maxeval=2000);
 opts[[3]]=list(use_local=1,aggr=0,local_search=2,decomp=0,maxdist=0.5,maxeval=2000);
 opts[[4]]=list(use_local=1,aggr=0,local_search=2,decomp=0,maxdist=0.5,maxeval=2000);
 opts$hosts=c('localhost','localhost','localhost','localhost');
 opts$ce_niter=2;
 opts$ce_type="SOCKS";
 opts$ce_isparallel= TRUE;
 Results=MEIGO(problem,opts, algorithm="CeVNSR");
```

---

<table>
<thead>
<tr>
<th>cnolist</th>
<th>A CNOlist from CellNOptR package</th>
</tr>
</thead>
</table>

**Description**

A CNOlist from CellNOptR to use with provided examples

---

<table>
<thead>
<tr>
<th>cur_params</th>
<th>Current values of all model parameters</th>
</tr>
</thead>
</table>

**Description**

For a given set of values for the parameters to be estimated, this method returns an array containing the actual (not log-transformed) values of all model parameters, not just those to be estimated, in the same order as specified in the model. This is helpful when simulating the model at a given position in parameter space.
cur_params

Usage

cur_params(output, options, position = NULL)

Arguments

options list with entries as explained below. Options set – defines the problem and sets some parameters to control the MCMC algorithm. model: List of model parameters - to estimate. The parameter objects must each have a 'value' attribute containing the parameter's numerical value. estimate_params: list. List of parameters to estimate, all of which must also be listed in 'options$model$parameters'. initial_values: list of float, optional. Starting values for parameters to estimate. If omitted, will use the nominal values from 'options$model$parameters' step_fn: callable f(output), optional. User callback, called on every MCMC iteration. likelihood_fn: callable f(output, position). User likelihood function. prior_fn: callable f(output, position), optional. User prior function. If omitted, a flat prior will be used. nsteps: int. Number of MCMC iterations to perform. use_hessian: logical, optional. Wheter to use the Hessian to guide the walk. Defaults to FALSE. rtol: float or list of float, optional. Relative tolerance for ode solver. atol: float or list of float, optional. Absolute tolerance for ode solver. norm_step_size: float, optional. MCMC step size. Defaults to a reasonable value. hessian_period: int, optional. Number of MCMC steps between Hessian recalculation. Defaults to a reasonable value. hessian_period: int, optional. Scaling factor used in generating Hessian-guided steps. Defaults to a reasonable value. sigma_adj_interval: int, optional. How often to adjust 'output$sig_value' while annealing to meet 'accept_rate_target'. Defaults to a reasonable value. sigma_max: float, optional. Maximum value for 'output$sig_value'. Defaults to a reasonable value. sigma_min: float, optional. Minimum value for 'output$sig_value'. Defaults to a reasonable value. sigma_step: float, optional. Increment for 'output$sig_value' adjustments. Defaults to a reasonable value. thermo_temp: float in the range [0,1], optional. Temperature for thermodynamic integration support. Used to scale likelihood when calculating the posterior value. Defaults to 1, i.e. no effect.

output List of output values with entries as explained below. num_estimate: int. Number of parameters to estimate. estimate_idx: list of int. Indices of parameters to estimate in the model’s full parameter list. initial_values: list of float. Starting values for parameters to estimate, taken from the parameters’ nominal values in the model or explicitly specified in ‘options’. initial_position: list of float. Starting position of the MCMC walk in parameter space (log10 of ‘initial_values’). position: list of float. Current position of MCMC walk in parameter space, i.e. the most recently accepted move. test_position: list of float. Proposed MCMC move. acceptance: int. Number of accepted moves. T: float. Current value of the simulated annealing temperature. T_decay: float. Constant for exponential decay of ‘T’, automatically calculated such that T will decay from ‘options$anneal_length’ down to 1 over the first ‘options$anneal_length’ setps. sig_value: float. Current value of sigma, the scaling factor for the proposal distribution.

**position** list of float, optional. log10 of the values of the parameters being estimated. If omitted, ‘output$position’ (the most recent accepted output move) will be used. The model’s nominal values will be used for all parameters *not* being estimated, regardless.

**Value**

A list of the values of all model parameters.

**Examples**

```r
data("simpleExample", package="MEIGOR")

initial_pars = createLBodeContPars(model, LB_n=1, LB_k=0.1, LB_tau=0.01, UB_n=5, UB_k=0.9, UB_tau=10, random=TRUE)
simData = plotLBodeFitness(cnolist, model, initial_pars, reltol=1e-05, atol=1e-03, maxStepSize=0.01)

f_bayesFit <- function(position, params=initial_pars, exp_var=opts$exp_var) {
  # convert from log
  params$parValues = 10^position
  ysim = getLBodeDataSim(cnolist=cnolist, model=model, ode_parameters=params)
data_as_vec = unlist(cnolist$valueSignals)
sim_as_vec = unlist(cnolist$valueSignals)
  sim_as_vec[is.na(sim_as_vec)] = 0
}
```

sim_as_vec[is.nan(sim_as_vec)]= 0
return(sum((data_as_vec-sim_as_vec)^2/(2*exp_var^2)))
}

prior_mean = log10(initial_pars$parValues)
prior_var = 10

opts <- list("model"=NULL, "estimate_params"=NULL,"initial_values"=NULL, "tspan"=NULL, "step_fn"=NULL, "likelihood_fn"=NULL, "prior_fn"=NULL, "nsteps"=NULL, "use_hessian"=FALSE, "rtol"=NULL, "atol"=NULL, "norm_step_size"=0.75, "hessian_period"=25000, "hessian_scale"=0.005, "sigma_adj_interval"=NULL, "anneal_length"=NULL, "T_init"=10, "accept_rate_target"=0.3, "sigma_max"=1, "sigma_min"=0.25, "sigma_step"=0.125, "thermo_temp"=1, "seed"=NULL)

opts$nsteps = 2000
opts$likelihood_fn = f_bayesFit
opts$use_hessian = TRUE
opts$hessian_period = opts$nsteps/10
opts$model = list(parameters=list(name=initial_pars$parNames, value=initial_pars$parValues))
opts$estimate_params = initial_pars$parValues
opts$exp_var = 0.01

res = runBayesFit(opts)

initial_pars$parValues = cur_params(output=res, options=opts)

### Local search algorithm within eSS

**Description**
Local search algorithm within eSS

**Note**
For internal use of MEIGOR.

### essR

**Global optimization algorithm for MINLPs based on Scatter Search**

**Description**

essR attempts to solve problems of the form:

\[
\min f(x, p_1, p_2, \ldots, p_n)
\]

subject to:

\[
c_x = 0
\]

\[
c_L \leq c(x) \leq c_U
\]

\[
x_L \leq x \leq x_U
\]
Usage

essR(problem, opts = list(maxeval = NULL, maxtime = NULL), ...)

Arguments

problem List containing problem definition.

opts List containing options (if set as opts <- numeric(0) default options will be

... Additional variables passed to the objective function

Details

Problem definition:

problem$f: Name of the file containing the objective function (String).

problem$x_L: Lower bounds of decision variables (vector).

problem$x_U: Upper bounds of decision variables (vector).

problem$x_0: Initial point(s) (optional; vector or matrix).

problem$f_0: Function values of initial point(s) (optional). These values MUST correspond to feasible points.

NOTE: The dimension of f_0 and x_0 may be different. For example, if we want to introduce 5 initial points but we only know the values for 3 of them, x_0 would have 5 rows whereas f_0 would have only 3 elements. In this example, it is mandatory that the first 3 rows of x_0 correspond to the values of f_0.

Fill the following fields if your problem has non-linear constraints:

problem$neq: Number of equality constraints (Integer; do not define it if there are no equality constraints).

problem$c_L: Lower bounds of nonlinear inequality constraints (vector).

problem$c_U: Upper bounds of nonlinear inequality constraints (vector).

problem$int_var: Number of integer variables (Integer).

problem$bin_var: Number of binary variables (Integer).

problem$vtr: Objective function value to be reached (optional).

User options:

opts$maxeval: Maximum number of function evaluations (Default 1000).

opts$maxtime: Maximum CPU time in seconds (Default 60).

opts$iterprint: Print each iteration on screen: 0-Deactivated; 1-Activated (Default 1).

opts$plot: Plots convergence curves: 0-Deactivated; 1-Plot curves on line; 2-Plot final results (Default 0).

opts$weight: Weight that multiplies the penalty term added to the objective function in constrained problems (Default 1e6).

opts$log_var: Indexes of the variables which will be used to generate diverse solutions in different orders of magnitude (vector).

opts$tolc: Maximum absolute violation of the constraints (Default 1e-5).

opts$prob_bound: Probability (0-1) of biasing the search towards the bounds (Default 0.5).

opts$inter_save: Saves results in a mat file in intermediate iterations. Useful for very long runs (Binary; Default = 0).

Global options:
opts$dim_refset: Number of elements in Refset (Integer; automatically calculated).

opts$n_diverse: Number of solutions generated by the diversificator (Default 10*nvar).

opts$combination: Type of combination of Refset elements (Default 1); 1: hyper-rectangles; 2: linear combinations.

Local options:

opts$local_solver: Choose local solver 0: Local search deactivated (Default), "NM", "BFGS", "CG", "LBFGSB", "SA", "SOLNP".

opts$local_tol: Level of tolerance in local search.

opts$local_riterprint: Print each iteration of local solver on screen (Binary; default = 0).

opts$local_n1: Number of iterations before applying local search for the 1st time (Default 1).

opts$local_n2: Minimum number of iterations in the global phase between 2 local calls (Default 10).

opts$local_balance: Balances between quality (=0) and diversity (=1) for choosing initial points for the local search (default 0.5).

opts$local_finish: Applies local search to the best solution found once the optimization if finished (same values as opts.local_solver).

opts$local_bestx: When activated (i.e. =1) only applies local search to the best solution found to date, ignoring filters (Default=0).

Value

fbest: Best objective function value found after the optimization
xbest: Vector providing the best function value
cpu_time: Time in seconds consumed in the optimization
f: Vector containing the best objective function value after each iteration
x: Matrix containing the best vector after each iteration
time: Vector containing the cpu time consumed after each iteration
neval: Vector containing the number of function evaluations after each iteration
numeval: Number of function evaluations
local_solutions: Local solutions found by the local solver (in rows)
local_solutions_values: Function values of the local solutions

end_crit: Criterion to finish the optimization:

1 Maximal number of function evaluations achieved
2 Maximum allowed CPU Time achieved
3 Value to reach achieved

Note

R code of the eSS optimization code from: Process Engineering Group IIM-CSIC.

Constraint functions, if applicable, must be declared in the same script as the objective function as a second output argument, e.g.:

```r
myfunction <- function(x){
calculate fx - scalar containing the objective function value
```
calculate gx - vector (or empty) containing the constraints values

return(list(fx,gx))
}

Author(s)
Jose Egea

References
If you use essR and publish the results, please cite the following papers:

Examples
#1 Unconstrained problem
ex1 <- function(x){
  return(y)
}

#global optimum
#x*=[0.0898, -0.7127];
#or
#x*=[-0.0898, 0.7127];
#
#f(x*)= -1.03163;

#========================= PROBLEM SPECIFICATIONS ===========================
problem<-list(f="ex1",x_L=rep(-1,2),x_U=rep(1,2))
opts<-list(maxeval=500, ndiverse=10, dim_refset=4, local_solver="solnp", local_n2=1)
#========================= END OF PROBLEM SPECIFICATIONS ======================

Results<-essR(problem,opts);

#2 Constrained problem
ex2<-function(x){
  F=-x[1]-x[2];
  g<-rep(0,2);
  return(list(F=F,g=g))
}

# global optimum
#x*=[2.32952, 3.17849];
#f(x*)=-5.50801
# Problem Specifications

```r
# ========================= PROBLEM SPECIFICATIONS ==========================
problem <- list(f="ex2", x_L=rep(0,2), x_U=c(3,4), c_L=rep(-Inf,2), c_U=c(2,36))
opts <- list(maxeval=750, local_solver="solnp", local_n2=1)
# ========================= END OF PROBLEM SPECIFICATIONS =====================
```

Results <- essR(problem, opts);

#3 Constrained problem with equality constraints

```r
ex3 <- function(x,k1,k2,k3,k4){
f = -x[4];
# Equality constraints
g <- rep(0,5);
g[2] = x[1] - 1 + k1*x[1]*x[5];
# Inequality constraint

g[5] = x[5]^0.5 + x[6]^0.5;
return(list(f = f, g = g));
}
# Global optimum
# x* = [0.77152, 0.516994, 0.204189, 0.388811, 3.0355, 5.0973];
# f(x*) = -0.388811;
```

# Problem Specifications

```r
# ========================= PROBLEM SPECIFICATIONS ==========================
problem <- list(f="ex3", x_L=rep(0,6), x_U=c(rep(1,4),16,16), neq=4, c_L=-Inf, c_U=4)
opts <- list(maxtime=7, local_solver="solnp", local_n2=10)
# ========================= END OF PROBLEM SPECIFICATIONS =====================
```

k1 = 0.09755988;
k2 = 0.0391908;
k3 = 0.99*k1;
k4 = 0.9*k3;
Results <- essR(problem, opts, k1, k2, k3, k4);

#4 Mixed integer problem

```r
ex4 <- function(x){
g <- rep(0,3);
```
ith_essR

return(list(F=F, g=g));
}

# global optimum
x*=[2.23607, 0, 1, 0];
f(x*)=-40.9575;

#========================= PROBLEM SPECIFICATIONS ===========================
problem<-list(f="ex4", x_L=rep(0,4), x_U=rep(10,4), x_0=c(3,4,5,1), int_var=3, c_L=rep(-Inf,3), c_U=c(8,10,5))
opts<-list(maxtime=2)
#========================= END OF PROBLEM SPECIFICATIONS =====================

Results<-essR(problem,opts);

---

**essR_multistart**  
**Multistart function for eSS**

**Description**

Multistart function for eSS

**Note**

For internal use of CNORode.

---

**eucl_dist**  
**Computes the euclidean distance between the rows of two different matrices**

**Description**

This functions is used internally by essR to compute the euclidean distance between the rows of two different matrices. The matrices must have the same number of columns.

**Note**

For internal use of MEIGOR.

---

**ith_essR**  
**Auxiliary function to perform parallel runs**

**Description**

Auxiliary function to perform parallel runs

**Note**

For internal use of MEIGOR.
ith_VNSR  

**Auxiliary function to perform parallel runs**

**Description**

Auxiliary function to perform parallel runs

**Note**

For internal use of MEIGOR.

---

**MEIGO  

**MEIGO main function**

**Description**

Wrapper around the different optimisation methods

**Usage**

```
MEIGO(problem, opts, algorithm, ...)
```

**Arguments**

- **problem** List containing problem settings.
- **opts** A list of n_threads lists containing options for each cooperative instance of essR.
- **algorithm** One of VNS, ESS, MULTISTART, CESSR, CEVNSR. Check the documentation of each algorithm for more information.
- ... Additional input arguments.

**See Also**

essR rvnds_hamming CeVNSR CeSSR

**Examples**

```r
#global optimum

#x*=[0.0898, -0.7127];
# or
#x*=[-0.0898, 0.7127];
#
#f(x*)=-1.03163;

ex1 <- function(x){
return(y)
}
```

#========================= PROBLEM SPECIFICATIONS ===========================
paramsOpt

```r
problem <- list(f = ex1, x_L = rep(-1, 2), x_U = rep(1, 2))
opts <- list(maxeval = 500, ndiverse = 40, local_solver = 'DHC', local_finish = 'LBFGSB', local_iterprint = 1)

Results <- MEIGO(problem, opts, algorithm = "ESS");
```

---

### model

**Description**

A model from CellNoptR to use with provided examples

---

### nls_fobj

**Description**

Auxiliary function to evaluate constraints

**Note**

For internal use of MEIGOR.

---

### optim_fobj

**Description**

Gateway function to evaluate the objective function when the local solvers are invoked.

**Note**

For internal use of MEIGOR.

---

### paramsOpt

**Description**

Optimal parameters for simulation with CNORode to use with provided examples
runBayesFit

Running the BayesFit optimisation

Description

"runBayesFit" defines the prior function and runs the BayesFit estimation

Usage

runBayesFit(opts)

Arguments

opts list with entries as explained below. Options set – defines the problem and sets some parameters to control the MCMC algorithm. model: List of model parameters - to estimate. The parameter objects must each have a 'value' attribute containing the parameter's numerical value. estimate_params: list. List of parameters to estimate, all of which must also be listed in 'options$model$parameters'. initial_values: list of float, optional. Starting values for parameters to estimate. If omitted, will use the nominal values from 'options$model$parameters' step_fn: callable f(output), optional. User callback, called on every MCMC iteration. likelihood_fn: callable f(output, position). User likelihood function. prior_fn: callable f(output, position), optional. User prior function. If omitted, a flat prior will be used. nsteps: int. Number of MCMC iterations to perform. use_hessian: logical, optional. Wheter to use the Hessian to guide the walk. Defaults to FALSE. rtol: float or list of float, optional. Relative tolerance for ode solver. atol: float or list of float, optional. Absolute tolerance for ode solver. norm_step_size: float, optional. MCMC step size. Defaults to a reasonable value. hessian_period: int, optional. Number of MCMC steps between Hessian recalculations. Defaults to a reasonable but fairly large value, as Hessian calculation is expensive. hessian_scale: float, optional. Scaling factor used in generating Hessian-guided steps. Defaults to a reasonable value. sigma_adj_interval: int, optional. How often to adjust 'output$sig_value' while annealing to meet 'accept_rate_target'. Defaults to a reasonable value. anneal_length: int, optional. Length of initial "burn-in" annealing period. Defaults to 10 'nsteps', or if 'use_hessian' is TRUE, to 'hessian_period' (i.e. anneal until first hessian is calculated) T_init: float, optional. Initial temperature for annealing. Defaults to a reasonable value. accept_rate_target: float, optional. Desired acceptance rate during annealing. Defaults to a reasonable value. See also 'sigma_adj_interval' above. sigma_max: float, optional. Maximum value for 'output$sig_value'. Defaults to a reasonable value. sigma_min: float, optional. Minimum value for 'output$sig_value'. Defaults to a reasonable value. sigma_step: float, optional. Increment for 'output$sig_value' adjustments. Defaults to a reasonable value. To eliminate adaptive step size, set sigma_step to 1. thermo_temp: float in the range [0,1], optional. Temperature for thermodynamic integration support. Used to scale likelihood when calculating the posterior value. Defaults to 1, i.e. no effect.

Value

The output after the optimisation is finished - a list with entries as explained in 'Arguments'.
Examples

data("simpleExample", package="MEIGOR")
initial_pars = createLBOdeContPars(model, LB_n=1, LB_k=0.1, LB_tau=0.01, UB_n=5, UB_k=0.9, UB_tau=10, random=TRUE)
simData = plotLBOdeFitness(cnolist, model, initial_pars, reltol=1e-05, atol=1e-03, maxStepSize=0.01)

f_bayesFit <- function(position, params=initial_pars, exp_var=opts$exp_var) {
  # convert from log
  params$parValues = 10^position
  ysim = getLBOdeDataSim(cnolist=cnolist, model=model,
    ode_parameters=params)
  data_as_vec = unlist(cnolist$valueSignals)
  sim_as_vec = unlist(ysim)
  # set nan (NAs) to 0
  sim_as_vec[is.na(sim_as_vec)]= 0
  sim_as_vec[is.nan(sim_as_vec)]= 0
  return(sum((data_as_vec-sim_as_vec)^2/(2*exp_var^2)))
}

prior_mean = log10(initial_pars$parValues)
prior_var = 10

opts <- list("model"=NULL, "estimate_params"=NULL,"initial_values"=NULL,
  "tspan"=NULL, "step_fn"=NULL, "likelihood_fn"=NULL,
  "prior_fn"=NULL, "nsteps"=NULL, "use_hessian"=FALSE,
  "rtol"=NULL, "atol"=NULL, "norm_step_size"=0.75,
  "hessian_period"=25000, "hessian_scale"=0.085,
  "sigma_adj_interval"=NULL, "anneal_length"=NULL,
  "T_init"=10, "accept_rate_target"=0.3, "sigma_max"=1,
  "sigma_min"=0.25, "sigma_step"=0.125, "thermo_temp"=1, "seed"=NULL)

opts$nsteps = 2000
opts$likelihood_fn = f_bayesFit
opts$use_hessian = TRUE
opts$hessian_period = opts$nsteps/10
opts$model = list(parameters=list(name=initial_pars$parNames,
  value=initial_pars$parValues))
opts$estimate_params = initial_pars$parValues
opts$exp_var = 0.01

res = runBayesFit(opts)
initial_pars$parValues = cur_params(output=res, options=opts)

---

**rvnds_hamming**

**Main VNS function**

**Description**

VNS Kernel function

**Usage**

```r
ervnds_hamming(problem, opts, ...)
```
Arguments

- **problem**: List containing problem settings definition.
- **opts**: List containing options (if set as opts <- numeric(0) default options will be loaded).
- ... Additional variables passed to the objective function

Details

- **problem$f**: Name of the file containing the objective function (String).
- **problem$x_L**: Lower bounds of decision variables (vector).
- **problem$x_U**: Upper bounds of decision variables (vector).
- **problem$x_0**: Initial point(s) (optional; vector or matrix).
- **problem$f_0**: Function values of initial point(s) (optional). These values MUST correspond to feasible points.

User options:

- **opts$maxeval**: Maximum number of function evaluations (Default 1000).
- **opts$maxtime**: Maximum CPU time in seconds (Default 60).
- **opts$maxdist**: Percentage of the problem dimension which will be perturbed in the furthest neighborhood (varies between 0 and 1, default is 0.5).
- **opts$use_local**: Uses local search (1) or not (0). The default is 1.

The following options only apply when the local search is activated:

- **opts$use_aggr**: Aggressive search. The local search is only applied when the best solution has been improved (1=aggressive search, 0=non-aggressive search, default:0).
- **opts$local search type**: Applies a first (=1) or a best (=2) improvement scheme for the local search (Default: 1).
- **opts$decomp**: Decompose the local search (=1) using only the variables perturbed in the global phase. Default: 1.

Value

- **fbest**: Best objective function value found after the optimization
- **xbest**: Vector providing the best function value
- **cpu_time**: Time in seconds consumed in the optimization
- **func**: Vector containing the best objective function value after each iteration
- **x**: Matrix containing the best vector after each iteration
- **time**: Vector containing the cpu time consumed after each iteration
- **neval**: Vector containing the number of function evaluations after each iteration
- **numeval**: Number of function evaluations

Examples

```r
rosen10<-function(x){
  f<-0;
  n=length(x);
  for (i in 1:(n-1)){
```
\[ f \gets f + 100 \times (x[i]^2 - x[i+1])^2 + (x[i]-1)^2; \]
\]
return(f)
\}

nvar<-10;

problem<-list(f="rosen10", x_L=rep(-5,nvar), x_U=rep(1,nvar))

opts<-list(maxeval=2000, maxtime=3600*69, use_local=1, aggr=0, local_search_type=1, decomp=1, maxdist=0.5)

algorithm<-"VNS";

Results<-MEIGO(problem,opts,algorithm);

---

**rvnds_local**  
Local search in VNS

**Description**  
Local search in VNS

**Note**  
For internal use of MEIGOR.

---

**solnp_eq**  
Gateway function to evaluate the equality constraints when solnp is invoked as local solver

**Description**  
This function is used by essR to evaluate the equality constraints when solnp is invoked as local solver

**Note**  
For internal use of MEIGOR.

---

**solnp_fobj**  
Gateway function to evaluate the objective function when solnp is invoked as local solver

**Description**  
This function is used internally by essR to evaluate the objective function when solnp is invoked as local solver

**Note**  
For internal use of MEIGOR.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>solnp_ineq</td>
<td>Gateway function to evaluate the inequality constraints when solnp is invoked as local solver</td>
</tr>
<tr>
<td>ssm_beyond</td>
<td>Function that expands the search direction when a good offspring solution has been found</td>
</tr>
<tr>
<td>ssm_defaults</td>
<td>Sets the default options for eSSR</td>
</tr>
<tr>
<td>ssm_evalfc</td>
<td>Gateway function to evaluate the objective function in essR</td>
</tr>
</tbody>
</table>

**Description**

This function is used internally by essR to evaluate the inequality constraints when solnp is invoked as local solver.

**Note**

For internal use of MEIGOR.

This function is used internally by essR to expand the search direction when a good offspring solution has been found.

**Note**

For internal use of MEIGOR.

This function is used internally essR to set default options.

**Note**

For internal use of MEIGOR.

This function is used internally by essR to evaluate the objective function.

**Note**

For internal use of MEIGOR.
### ssm_isdif2

**Calculates relative errors between two vectors**

**Description**

This function is used internally by essR to calculate relative errors between two vectors.

**Note**

For internal use of MEIGOR.

### ssm_localsolver

**Configure local solver**

**Description**

Sets the different options and parameters for the local solvers invoked by essR.

**Note**

For internal use of MEIGOR.

### ssm_optset

**Assigns values to the options defined by the user**

**Description**

This function is used internally by essR for assigning values to the options defined by the user.

**Note**

For internal use of MEIGOR.

### ssm_penalty_function

**Calculates the penalized objective function in constrained problems**

**Description**

This function is used internally by essR to calculate the penalized objective function in constrained problems.

**Note**

For internal use of MEIGOR.
**ssm_round_int**  
*Rounds variables declared as integer of binary*

**Description**

This function is used internally by essR to round variables declared as integer of binary.

**Note**

For internal use of MEIGOR.

---

**vns_defaults**  
*Default options for VNS*

**Description**

Default options for VNS

**Usage**

vns_defaults(...)

**Arguments**

...

---

**vns_optset**  
*Set VNS options*

**Description**

Set VNS options

**Note**

For internal use of MEIGOR.
Index

*Topic Neighbourhood
  CeVNSR, 6
*Topic cooperative
  CeSSR, 4
  CeVNSR, 6
*Topic metaheuristic
  CeSSR, 4
*Topic optimization
  essR, 11
  essR-package, 3
  MEIGOR-package, 2
*Topic package
  essR-package, 3
  MEIGOR-package, 2
*Topic scatter
  CeSSR, 4
  essR, 11
  essR-package, 3
  MEIGOR-package, 2
*Topic search
  CeSSR, 4
  CeVNSR, 6
  essR, 11
  essR-package, 3
  MEIGOR-package, 2
*Topic strategies
  CeSSR, 4
  CeVNSR, 6
*Topic variable
  CeVNSR, 6

BayesFit, 4
BayesFit-package (BayesFit), 4
CeSSR, 4, 17
CeVNSR, 6, 17
cnolist, 8
cur_params, 8
dhc, 11

essR, 3–5, 11, 17
essR-package, 3
essR_multistart, 16
eucl_dist, 16
ith_essR, 16
ith_VNSR, 17
MEIGO, 8, 17
MEIGOR-package, 2
model, 18
nls_fobj, 18
optim_fobj, 18
paramsOpt, 18
runBayesFit, 19
rvnds_hamming, 7, 8, 17, 20
rvnds_local, 22
solnp_eq, 22
solnp_fobj, 22
solnp_ineq, 23
ssm_beyond, 23
ssm_defaults, 23
ssm_evalfc, 23
ssm_isdif2, 24
ssm_localsolver, 24
ssm_optset, 24
ssm_penalty_function, 24
ssm_round_int, 25
vns_defaults, 25
vns_optset, 25