function [x, fval, exitflag, output] = pso(objfunc, nvars, options)

% PSO Particle Swarm Optimization Algorithm

%

% PSO attempts to solve continuous optimization problem of the form

%

% min OBJFUNC(X)

% X

%

% No constraints on the values of entries of X can be imposed, not ewen box contraints in form of

% the lower and upper bound. Any desired constraints should be dealth with by modification of the

% objective function: by introduction of penalty functions, by reformulation of the problem, or some

% other way.

%

% In the present implementation, only classical 'star topology' PSO is considered, where each

% particle is interconnected with each other particle, and there are no subswarms. Also, no

% additional, GA like, operators are applied to the particles during the search.

%

% PSO(OBJFUNC, NVARS) Optimizes objective OBJFUNC defined as a real-valued function of a single,

% real-valued vector argument. Dimensionality of the search space, that is the number of entries

% in the argument of OBJFUNC, is defined in NVARS. PSO expects at least these two arguments.

% Failure to provide any of them will result in error.

%

% PSO(OBJFUNC, NVARS, OPTIONS) Enables the user to specify a number of settings in order to

% customize or fine-tune the performance of the algorithm. OPTIONS is a structure containing

% desired values of these settings. Detailed description of available parameters is given in the

% sequel.

%

% X = PSO(...) Returns vector at which objective attains its minimal value. Due to the nature of

% PSO algorithm, this is not guaranteed to be neither exact global nor local optimum, yet PSO is

% robust optimizer, well suited for complex, multimodal problems. Well tuned, it can give VERY

% GOOD and quite commonly EXCELENT solution to the problem at hand. In most cases, this is all

% that is needed.

%

% [X, FVAL] = PSO(...) In addition to the optimal X, it returns the value of the objective at X,

% FVAL = OBJFUNC(X).

%

% [X, FVAL, EXITFLAG] = PSO(...) Returns indication concerning a reason why the algorithm stopped.

% In the current implementation the only supported return values are 0 and 1. EXITFLAG 0 denotes

% that maximum number of iterations has been achieved, while EXITFLAG 1 is used in testing mode,

% if the value of global minimum has been found prior to achieving the maximal number of

% iteration.

%

% [X, FVAL, EXITFLAG, OUTPUT] = PSO(...) Returns OUTPUT structure containing valuable information

% concerning performance of the algorithm in the present run. The exact members of this structure

% are variable, and depend on the settings specified in the OPTIONS structure. In general, there

% are four levels of detail that can be specified. The 'LOW' detail level means that the algorithm

% keeps track of objective value for the best, the mean and the worst particle in the swarm in

% each generation. If the 'MEDIUM' level is specified, exact position and index of the global best

% particle are tracked for during each iteration. If the 'HIGH' level is specified, exact position

% of each particle in the swarm is logged during the entire search process. If the swarm is large

% and the number of generations is great, high level output logging can result in considerable

% memmory consumption. Finally, the output log level can be set to 'NONE', meaning that no data is

% wanted within the OUTPUT structure. The exact way for specifying output logging level is given

% below.

%

% OPTIONS = PSO('options') Returns default options structure. Usefull when one desires to change

% values of only a handfull of options.

%

% The OPTIONS structure contains the following entries

%

% options.npart The number of particles in the swarm.

% options.niter The maximal number of iterations.

% options.cbi Initial value of the individual-best acceleration factor.

% options.cbf Final value of the individual-best acceleration factor.

% options.cgi Initial value of the global-best acceleration factor.

% options.cgf Final value of the global-best acceleration factor.

% options.wi Initial value of the inertia factor.

% options.wf Final value of the inertia factor.

% options.vmax Absolute speed limit. If specified, the speed is clamped to the range

% [-options.vmax, options.vmax]. It is the primary speed limit, if set

% to NaN, the VMAXSCALE options is used.

% options.vmaxscale Relative speed limit. Used only if absolute limit is unspecified, i.e.

% set to NaN. If used, must be a scalar quantity, and denotes the amount

% of initial population span (the INITSPAN option) used as the speed

% limit.

% options.vspaninit The initial velocity span. Initial velocities are initialized

% uniformly in [-VSPANINIT, VSPANINIT]. This option must be specified.

% options.initoffset Offset of the initial population. Can be scalar or column-vector of

% dimension NVARS.

% options.initspan Span of the initial population. Can be scalar or column-vector of

% dimension NVARS.

% options.trustoffset If set to 1 (true) and offset is vector, than the offset is

% believed to be a good solution candidate, so it is included in

% the initial swarm.

% options.initpopulation The user-suplied initial population. If this is set to something

% meaningfull, then INITSPAN, INITOFFSET and TRUSTOFFSET are

% ignored. If set to NaN then the above mentioned offset is used.

% options.verbose\_period The verbose period, i.e. the number of iterations after which the

% results are prompted to the user. If set to 0, then verbosing is

% skipped.

% options.plot If set to 1, evolution of the global best is ploted to the user after

% the optimization process. The objective value of the best, mean

% and worse particle troughout the optimization process are plotted

% in the single graph.

% options.output\_level The output log level. Possible values are: 'none', 'low',

% 'medium', 'high'. Each log level denotes a specific amount of

% information to be returned to the end user. If less than 4 output

% arguments are specified log level is ignored, since it would only

% occupate (possibly) large amount of useless memory.

% options.globalmin Global minimum, used for testing only. If specified, the algorithm

% will stop as soon as the difference between GLOBALMIN option and

% current global best becomes less than TOL option.

% options.tol Precision tolerance, used for testing only. It is maximal difference

% between current global best and GLOBALMIN option at which the

% algorithm stops. If GLOBALMIN options is set to NaN this option is

% ignored, and the algorithm stops after the maximal number of iteration

% has been achieved.

%

% The OUTPUT structure is the fallowing

%

% output.itersno The actual number of iterations. Usualy the same as NITER options, but

% can be less if in testing mode (if GLOBALMIN option is specified).

%

% If at least LOW level output logging is specified:

%

% output.gbest\_array

% output.gmean\_array

% output.gworst\_array Arrays containing the objective value of the best, mean and worst

% particle in each iteration. In fact, gmean\_array does not contain

% objective value for any concrete particle, but instead it contains the

% mean objective value for the entire swarm in each iteration.

%

% If at least MEDIUM level output logging is specified:

%

% output.gbes\*\*x\_array The array ontaining indices of the best particle in each iteration.

% output.Xbest Matrix of dimension NITERxNVARS, containing, as rows, global best

% particles in each iteration.

%

% Only if HIGH level output logging is specified:

%

% output.X 3D matrix of dimension NPARTxNVARSxNITER containing the entire

% population in each iteration.

%

%

% The following examples ilustrate the use of PSO function in several common cases.

%

% Suppose we are attempting to optimize 5-dimensional objective f(x) = x\*x'. Since, it is assumed

% that objective receives a row-vector, the objective is in fact a 5D paraboliod. The easiest way

% to optimize would be to write

%

% obj = @(x) x\*x';

% [xopt, fopt] = pso(obj, 5);

%

% The preseding code lines would yield XOPT as the found near-optimal solution and FOPT as the

% objective value in such point. Of course, other outputs can be obtained as described earlier.

%

% If one should choose to change the default options (say to specify different number of particles

% and iterations), the code would look something like this

%

% obj = @(x) x\*x';

% options = pso('options');

% options.niter = 500;

% options.npart = 60;

% [xopt, fopt] = pso(obj, 5);

%

% Other options can be specified as well on the exactly the same way. The best way to explore the

% option structure is to experiment.

%

% PSO is compatible with MATLAB 7 and higher. Some modifications are needed for it to work under

% lower versions of MATLAB.

% Copyright (C) Milan Rapaic ([email]repmilan@yahoo.com[/email]), 2007-05-10 ... 2008-10-12

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Checking the number of input and output arguments. %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

msg = nargchk(1, 3, nargin);

if ~isempty(msg)

error('mrr:myoptim:pso:pso:narginerr', 'Inadequate number of input arguments.');

end

msg = nargchk(0, 4, nargout);

if ~isempty(msg)

error('mrr:myoptim:pso:pso:nargouterr', 'Inadequate number of output arguments.');

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The body of the algorithm. %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin==1 && ischar(objfunc) && strcmp(objfunc, 'options')

% User desired only to access the default OPTIONS structure.

if nargout<=1

x = getDefaultOptions();

else

% The user required multiple outputs, yet only default options can be returned.

error('mrr:myoptim:pso:pso:nargouterr', ...

'Cannot expext more than one output when only OPTIONS are required.');

end

else

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% User requested optimization to be conducted on a given objective. %

% The following code deals with initializations and validations of options structure. %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If no options are specified, use the default ones.

if nargin<3, options=getDefaultOptions(); end

% Determination of output level, that is of amount of data to be collected in OUTPUT structure.

if nargout == 4

% User supplied four output arguments, therefore output level is determined from the OPTIONS

% structure. The output\_level variable is used to code the desired log level: 0 ('none'), 1

% ('low'), 2 ('medium') and 3 ('high).

if strcmp(options.output\_level, 'none')

if options.plot == 0

output\_level = 0;

else

output\_level = 1;

end

elseif strcmp(options.output\_level, 'low')

output\_level = 1;

elseif strcmp(options.output\_level, 'medium')

output\_level = 2;

elseif strcmp(options.output\_level, 'high')

output\_level = 3;

else

error('mrr:myoptim:pso:pso:optionserr:output\_level', ...

'Invalid value of the OUTPUT\_LEVEL options specified.');

end

else

% User has not supplied forth output argument. The only reason to log information during the

% run is to be able to plot to to the user after the optimization process. Therefore, if

% ploting is requested low level logging is used.

if options.plot == 1

output\_level = 1;

else

output\_level = 0;

end

end

% Maximum velocity can be specified in absolute amount, or relative to the initial span.

% If both values are specified, the absolute velocity limit is taken into account, while the

% relative is ignored. Whatever the initial specification of the maximal velocity, the curent

% code block will generate a column vector, vmax, containing maximal velocity along each

% dimension of search space.

if ~all(isnan(options.vmax))

% It is not allowed to let some of the entries of the VMAX option to be NaN and others to

% have numerical or Inf values.

if any(isnan(options.vmax))

error('mrr:myoptim:pso:pso:optionserr:vmax', ...

'VMAX option cannot have some Inf and some numerical (or Inf) values.');

end

% Warning of the confusing entries within the OPTIONS structure.

if ~isnan(options.vmaxscale)

warning('mrr:myoptim:pso:pso:optionserr:vmaxconflict', ...

'Both relative and absolute velocity limit are specified. The relative limit is ignored.');

end

if length(options.vmax) == 1

vmax = options.vmax\*ones(nvars, 1);

elseif length(options.vmax) == nvars

% Maximal velocity should be a column-vector or a scalar.

if size(options.vmax, 1) ~= length(options.vmax)

error('mrr:myopim:pso:pso:optionserr:vmax', ...

'VMAX option should be specified as column-vector, or as a scalar value.');

end

vmax = options.vmax;

else

error('mrr:myoptim:pso:pso:optionserr:vmax', ...

'Inadequate dimension of VMAX option. Should be a scalar, or a column vector with NVARS elements.');

end

else

% It is not valid to specify both VMAX and VMAXSCALE option as NaN.

if isnan(options.vmaxscale)

error('mrr:myoptim:pso:pso:optionserr:vmaxscale', ...

'Either VMAX or VMAXSCALE options should be different than NaN.');

end

% Contrary to the VMAX options, VMAXSCALE option must allways be a scalar. The initial span

% should take into account the different scaling among the cooedinates of the search space.

if length(options.vmaxscale) == 1

if length(options.initspan) == 1

vmax = options.vmaxscale\*options.initspan\*ones(nvars, 1);

else

% If the dimension of INITSPAN option is not correct, the function will break later,

% therefore, no need to check validity now.

vmax = options.vmaxscale\*options.initspan;

end

else

error('mrr:myoptim:pso:pso:optionserr:vmax', ...

'Inadequate dimension of VMAXSCALE option. Must be a scalar.');

end

end

vmax = repmat(vmax', options.npart, 1);

% Initial population.

% If the initial population is not supplied by the user, each particle of the initial population

% is spred in [INITOFFSET-INITSPAN, INITOFFSET+INITSPAN] where both INITOFFSET and INITSPAN

% are specified within the OPTIONS structure. Both of these options are either scalars or

% column-vectors of appropriate size. If INITPOPULATION option is specified, both INITOFFSET and

% INITSPAN options are ignored.

if ~isnan(options.initpopulation)

% The user supplied complete initial population within the OPTIONS structure.

% The size of the supplied population must be consistent with population size and number of

% variables. If no, an error is reported.

[pno, pdim] = size(options.initpopulation);

if (pno ~= options.npart) || (pdim ~= nvars)

error('mrr:myoptim:pso:pso:optionserr:initpopulation', ...

['The format of initial population is inconsistent with desired population', ...

'size or dimension of search space - INITPOPULATION options is invalid']);

end

X = options.initpopulation;

elseif (length(options.initoffset) == 1) && (length(options.initspan) == 1)

% The same offset and span is specified for each dimension of the search space

X = (rand(options.npart, nvars)-0.5)\*2\*options.initspan + options.initoffset;

elseif (length(options.initoffset) ~= size(options.initoffset, 1)) || ...

(length(options.initspan) ~= size(options.initspan, 1))

error('mrr:myoptim:pso:pso:optionserr:initoffset\_initspan', ...

'Both INITOFFSET and INITSPAN options must be either scalars or column-vectors.');

elseif (length(options.initoffset) ~= nvars) || (length(options.initspan) ~= nvars)

error('mrr:myoptim:pso:pso:optionserr:init', ...

'Both INITOFFSET and INITSPAN options must be scalars or column-vectors of length NVARS.');

else

initoffset = repmat(options.initoffset', options.npart, 1);

initspan = repmat(options.initspan', options.npart, 1);

X = (rand(options.npart, nvars)-0.5)\*2.\*initspan + initoffset;

% TRUSTOFFSET option is used when OFFSET option is, in fact, previously known good (or very

% good) solution to the problem at hand. When set to logical true (1), offset is inserted in

% the initial population. Thus, it is guaranteed that objective value at solution is not

% greater than objective value at that, previously known, good point.

if (options.trustoffset)

X(1, :) = options.initoffset';

end

end

% Initial velocities.

% Velocities are initialized uniformly in [-VSPANINIT, VSPANINIT].

if any(isnan(options.vspaninit))

error('mrr:myoptim:pso:pso:optionserr:vspaninit', ...

'VSPANINIT option must not contain NaN entries.');

elseif isscalar(options.vspaninit)

V = (rand(options.npart, nvars)-0.5)\*2\*options.vspaninit;

else

if (length(options.vspaninit) ~= size(options.vspaninit, 1)) || ...

(length(options.vspaninit) ~= nvars)

error('mrr:myoptim:pso:pso:optionserr:vspaninit', ...

'VSPANINIT option must be either scalar or column-vector of length NVARS');

end

V = (rand(options.npart, nvars)-0.5)\*2.\*repmat(options.vspaninit', options.npart, 1);

end

% Initial scores (objective values).

% Initialization of the best personal score and position, as well as global best score and

% position.

Y = calcobjfunc(objfunc, X);

Ybest = Y; % The best individual score for each particle - initialization.

Xbest = X; % The best individual position for each particle -

% initialization.

[GYbest, gbest] = min(Ybest); % GYbest is the best score within the entire swarm.

% gbest is the index of particle that achived YGbest.

gbest = gbest(1); % In case when more than one particle achieved the best

% score, we choose the one with the lowest index as the

% best one.

% These variables are used in testing mode only.

tolbreak = ~isnan(options.globalmin);

foundglobal = 0;

if tolbreak && ~isscalar(options.globalmin)

error('mrr:myoptim:pso:pso:optionserr:globalmin', ...

'globalmin option, if specified, option must be a scalar value equal to the global minimum of the objective function');

end

% Initialization of the OUTPUT structure.

% The output structure is filled and initialized differently depending on the OUTPUT\_LEVEL

% options, or equivalently depending on the output\_level variable.

if output\_level >= 0

% NONE log level

output.itersno = options.niter;

if output\_level >= 1

% LOW log level

output.gbest\_array = NaN\*ones(options.niter+1, 1);

output.gmean\_array = NaN\*ones(options.niter+1, 1);

output.gworst\_array = NaN\*ones(options.niter+1, 1);

output.gbest\_array(1) = GYbest;

output.gmean\_array(1) = mean(Ybest);

output.gworst\_array(1) = max(Ybest);

if output\_level >= 2

% MEDIUM log level

output.gbes\*\*x\_array = NaN\*ones(options.niter+1, 1);

output.Xbest = NaN\*ones(options.niter+1, nvars);

output.gbes\*\*x\_array(1) = gbest;

output.Xbest(1, :) = X(gbest, :);

if output\_level == 3

% HIGH log level

output.X = NaN\*zeros(options.npart, nvars, options.niter+1);

output.X(:,:,1) = X;

end

end

end

end

if options.verbose\_period ~= 0

disp 'PSO algorithm: Initiating the optimization process.'

end

% Denotes normal algorithm termination.

exitflag = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The main loop. %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for iter = 1:options.niter

% Verbosing, if neccessary.

if options.verbose\_period ~= 0

if rem(iter, options.verbose\_period) == 0

disp(['iteration ', int2str(iter), '. best criteria = ', num2str(GYbest)]);

end

end

% Calculating PSO parameters

w = linrate(options.wf, options.wi, options.niter, 0, iter);

cp = linrate(options.cbf, options.cbi, options.niter, 0, iter);

cg = linrate(options.cgf, options.cgi, options.niter, 0, iter);

% For later calculations only

GXbest = repmat(Xbest(gbest, :), options.npart, 1);

% Calculating speeds

V = w\*V + cp\*rand(size(V)).\*(Xbest-X) + cg\*rand(size(V)).\*(GXbest-X);

V = min(vmax, abs(V)).\*sign(V);

% Population is moving

X = X + V;

Y = calcobjfunc(objfunc, X);

% Calculating new individually best values

mask = Y<Ybest;

mask = repmat(mask, 1, nvars);

Xbest = mask.\*X +(~mask).\*Xbest;

Ybest = min(Y,Ybest);

% Calculating new globally best value

[GYbest, gbest] = min(Ybest);

gbest = gbest(1);

% Filling in the OUTPUT structure.

if output\_level >= 0

% NONE log level

if output\_level >= 1

% LOW log level

output.gbest\_array(iter+1) = GYbest;

output.gmean\_array(iter+1) = mean(Ybest);

output.gworst\_array(iter+1) = max(Ybest);

if output\_level >= 2

% MEDIUM log level

output.gbes\*\*x\_array(iter+1) = gbest;

output.Xbest(iter+1, :) = X(gbest, :);

if output\_level == 3

% HIGH log level

output.X(:,:,iter+1) = X;

end

end

end

end

% The code used in testing mode only.

if tolbreak && abs(GYbest - options.globalmin)<options.tol

output.itersno = iter;

foundglobal = 1;

break

end

end

if options.verbose\_period ~= 0

disp 'Optimization process finished.'

end

% Setting up the output variables.

% X is set to be the final global best position, since that is the best position ever achieved

% by any of the particles in the swarm. FVAL is set to be the value of the objective in X.

x = Xbest(gbest, :); x = x(:);

fval = GYbest;

% The global moptimum has been found prior to achieving the maximal number of iteration.

if foundglobal, exitflag = 1; end;

% Plotting the algorithm behavior at each iteration.

if options.plot

r = 0:options.niter;

figure

plot(r, output.gbest\_array, 'k.', r, output.gmean\_array, 'r.', r, output.gworst\_array, 'b.');

str = sprintf('Best objective value : %g', fval);

title(str);

legend({'best objective', 'mean objective', 'worst objective'})

end

end

function Y = calcobjfunc(func, X)

% CALCOBJFUNC A helper function used to calculate objective function value for a series of points.

np = size(X,1);

Y = zeros(np,1);

for i = 1:np

Y(i) = func(X(i,:));

end

function opts = getDefaultOptions

% GETDEFAULTOPTIONS Returns a structure containing the default options.

%

% This function, in fact, defines default values of the options within the options structure.

opts.npart = 30; % The number of particles.

opts.niter = 100; % The number of iterations.

opts.cbi = 2.5; % Initial value of the individual-best acceleration factor.

opts.cbf = 0.5; % Final value of the individual-best acceleration factor.

opts.cgi = 0.5; % Initial value of the global-best acceleration factor.

opts.cgf = 2.5; % Final value of the global-best acceleration factor.

opts.wi = 0.9; % Initial value of the inertia factor.

opts.wf = 0.4; % Final value of the inertia factor.

opts.vmax = Inf; % Absolute speed limit. It is the primary speed limit.

opts.vmaxscale = NaN; % Relative speed limit. Used only if absolute limit is unspecified.

opts.vspaninit = 1; % The initial velocity span. Initial velocities are initialized

% uniformly in [-VSPANINIT, VSPANINIT].

opts.initoffset = 0; % Offset of the initial population.

opts.initspan = 1; % Span of the initial population.

opts.trustoffset = 0; % If set to 1 (true) and offset is vector, than the offset is

% believed to be a good solution candidate, so it is included in

% the initial swarm.

opts.initpopulation = NaN; % The user-suplied initial population. If this is set to something

% meaningfull, then INITSPAN, INITOFFSET and TRUSTOFFSET are

% ignored.

opts.verbose\_period = 10; % The verbose period, i.e. the number of iterations after which the

% results are prompted to the user. If set to 0, then verbosing is

% skipped.

opts.plot = 0; % If set to 1, evolution of the gbest is ploted to the user after

% the optimization process. The objective value of the best, mean

% and worse particle troughout the optimization process are plotted

% in the single graph.

opts.output\_level = 'low'; % The output log level. Possible values are: 'none', 'low',

% 'medium', 'high'.

opts.globalmin = NaN; % Global minimum, used for testing only

opts.tol = 1e-6; % Precision tolerance, used for testing only

function x = linrate(xmax, xmin, tmax, tmin, t)

% LINRATE Linear interpolation of value X in instant T, defined by previously known points

% (tmin, xmin), (tmax, xmax)

x = xmin + ((xmax-xmin)/(tmax-tmin))\*(tmax-t);