MATLAB File Help: optimizer View code for optimizer Default Topics

optimizer

optimizer Container for optimization problem

```
OPT = optimizer(Constraints,Objective,options,x,u) exports an object that
contains precompiled numerical data to be solved for varying arguments
x, returning the optimal value of the expression u.

optimizer typically only works efficiently if the varying data x enters
the optmization problem affinely. If not, the precompiled problems will
be nonconvex, despite the problem being simple for a fixed value of the
parameter (see Wiki for beta support of a much more general optimizer)

In principle, if an optimization problem has a parameter x, and we
repeatedly want to solve the problem for varying x to compute a
variable u, we can, instead of repeatedly constructing optimization
problems for fixed values of x, introduce a symbolic x, and then
simply add an equality
 solvesdp([Constraints,x == value],Objective);
 uopt = double(u)
There will still be overhead from the SOLVESDP call, so we can
precompile the whole structure, and let YALMIP handle the addition of
the equality constraint for the fixed value, and automatically extract
the solution variables we are interested in
 OPT = optimizer(Constraints,Objective,options,x,u)
 uopt1 = OPT{value1}
 uopt2 = OPT{value2}
 uopt3 = OPT{value3}

By default, display is turned off (since optimizer is used in
situations where many problems are solved repeatedly. To turn on
display, set the verbose option in sdpsetting to 2.

Example

 The following problem creates an LP with varying upper and lower
 bounds on the decision variable.

 The optimizing argument is obtained by indexing (with {}) the optimizer
 object with the point of interest. The argument should be a column
 vector (if the argument has a width larger than 1, YALMIP assumes that
 the optimal solution should be computed in several points)

  A = randn(10,3);
  b = rand(10,1)*19;
  c = randn(3,1);

  z = sdpvar(3,1);
  sdpvar UB LB

  Constraints = [A*z <= b, LB <= z <= UB];
  Objective = c'*z
  % We want the optimal z as a function of [LB;UB]
  optZ = optimizer(Constraints,Objective,[],[LB; UB],z);
```

```
% Compute the optimal z when LB=1, UB = 3;
zopt = optZ{[1; 3]}

% Compute two solutions, one for (LB,UB) [1;3] and one for (LB,UB) [2;6]
zopt = optZ{[[1; 3], [2;6]]}

% A second output argument can be used to catch infeasibility
[zopt,infeasible] = optZ{[1; 3]}

% To avoid the need to vectorize in order to handle multiple
  parameters, a cell-based format can be used, both for inputs and
  outputs. Note that the optimizer object now is called with a cell
  and returns a cell

optZ = optimizer(Constraints,Objective,[],{LB,UB},{z,sum(z)})
[zopt,infeasible] = optZ{{1,3}};
zopt{1}
zopt{2}
```

Overloaded methods:

optproblem/optimizer