YALMIP Wiki

Basics

The following pice of code introduces essentially everything you ever need to learn. It defines variables, constraints, objectives, options, checks result and extracts solution (Note that the code specifies the solver to CPLEX. If you don't have CPLEX installed, simply remove that part of the option list if you want to run the code).

```
x = sdpvar(2,1);
Constraints = [sum(x) <= 1, x(1)==0, x(2) >= 0.5];
Objective = x'*x;
options = sdpsettings('verbose',1,'solver','cplex','cplex.qpmethod',1);
sol = solvesdp(Constraints,Objective,options);
if sol.problem == 0
 solution = double(x);
else
 display('Hmm, something went wrong!');
 sol.info
 yalmiperror(sol.problem)
end
```

Having seen that, let us start from the beginning.

YALMIPs symbolic variable

The most important command in YALMIP is sdpvar. This command is used to the define decision variables. To define a matrix (or scalar) P with n rows and m columns, we write

```
P = sdpvar(n,m)
```

A square matrix is symmetric by default!. To obtain a fully parameterized (i.e. not necessarily symmetric) square matrix, a third argument is needed.

```
P = sdpvar(3,3,'full')
```

The third argument can be used to obtain a number of pre-defined types of variables, such as Toeplitz, Hankel, diagonal, symmetric and skew-symmetric matrices. See the help text on sdpvar for details. Alternatively, the associated MATLAB commands can be applied to a vector.

```
x = sdpvar(n,1);
D = diag(x) ;     % Diagonal matrix
H = hankel(x);    % Hankel matrix
T = toeplitz(x); % Hankel matrix
```

Scalars can be defined in three different ways.

```
x = sdpvar(1,1); y = sdpvar(1,1);
x = sdpvar(1);   y = sdpvar(1);
sdpvar x y
```

Note that due to a bug in MATLAB, the last command-line syntax fails in some cases (inside functions), if the variable name is the same as some built-in function or variable (i, j, e, beta, gamma).

The sdpvar objects are manipulated in MATLAB as any other variable and most functions are overloaded. Hence, the following commands are valid

```
P = sdpvar(3,3) + diag(sdpvar(3,1));
X = [P P;P eye(length(P))] + 2*trace(P);
Y = X + sum(sum(P*rand(length(P)))) + P(end,end)+hankel(X(:,1));
```

In some situations, coding is simplified with a multi-dimensional variable. This is supported in YALMIP with two different constructs, cell arrays and multi-dimensional sdpvar objects.

The cell array is nothing but an abstraction of the following code

```
for i = 1:5
  X{i} = sdpvar(2,3);
```

```
end
```

By using vector dimensions in sdpvar, the same cell array can be setup as follows

```
X = sdpvar([2 2 2 2 2],[3 3 3 3 3]);
```

The cell array can now be used as usual in MATLAB.

The drawback with the approach above is that the variable X not can be used directly, as a standard sdpvar object. As an alternative, a completely general multi-dimensional sdpvar is available. We can create an essentially equivalent object with this call.

```
X = sdpvar(2,3,5);
```

The difference is that we can operate directly on this object, using standard MATLAB code.

```
Y = sum(X,3)
X((:,:,2)
```

Note that the two first slices are symmetric (if the two first dimensions are the same), according to standard YALMIP syntax. To create a fully paramterized higher-dimensional, use trailing flags as in the standard case.

```
X = sdpvar(2,2,2,2,'full');
```

For an illustration of multi/dimensional variables, check out the Sudoku example.

Constraints

To define a collection of constraints, we simply define and concatenate them. The meaning of a constraint is context-dependent. If the left-hand side and right-hand side are Hermitian, the constraint is interpreted in terms of positive definiteness, otherwise element-wise. Hence, declaring a symmetric matrix and a positive definiteness constraint is done with

```
n = 3;
P = sdpvar(n,n);
C = [P>=0];
```

while a symmetric matrix with positive elements is defined with, e.g.,

```
P = sdpvar(n,n);
C = [P(:)>=0];
```

Note that this defines the off-diagnoal constraints twice. A good SDP solver will perhaps detect this during preprocessing and reduce the model, but we can of-course define only the unique elements manually using standard MATLAB code

```
C = [triu(P)>=0];
```

or

```
C = [P(find(triu(ones(n))))>=0];
```

According to the rules above, a non-square matrix (or generally a non-symmetric) with positive elements can be defined using the >= operator immediately

```
P = sdpvar(n,2*n);
C = [P>=0];
```

and so can a fully parameterized matrix with positive elements

```
P = sdpvar(n,n,'full');
C = [P>=0];
```

A list of several constraints is defined by just adding or, alternatively, concatenating them.

```
P = sdpvar(n,n);
C = [P>=0] + [P(1,1)>=2];
C = [P>=0, P(1,1)>=2];
```

Of course, the involved expressions can be arbitrary sdpvar objects, and equality constraints (==) can be defined, as well as constraints using <=.

```
C = [P>=0, P(1,1)<=2, sum(sum(P))==10];
```

A convenient way to define several constraints is to use double-sided constraints.

```
F = [0 <= P(1,1) <= 2];
```

After having defined variables and constraints, you are ready to solve problems. Check out the remaining tutorials to learn this.