Solving semidefinite programs using `cvx`

There are now many software packages that solve SDPs efficiently, once you've put the problem into a standard format. But writing and debugging code that transforms your problem to a standard format can be a long and painful task. Fortunately, there are other codes that allow you to use a natural high level description of an SDP or LMI, and do the conversion automatically.

You will use `cvx`, a Matlab-based package for convex optimization that in particular handles SDPs and LMIs in a convenient way. You should get `cvx` from the URL

```
www.stanford.edu/~boyd/cvx/
```

and install it, after reading Appendix A of the user guide. You certainly don't need to read the entire user guide, but you might want to read the first few sections, or skim over it.

Once you have installed `cvx`, you can start using `cvx` by entering a `cvx` SDP *specification* into a Matlab script or function, or directly from the command prompt. To delineate `cvx` SDP specifications from surrounding Matlab code, they are preceded with the statement `cvx_begin sdp` and followed with the statement `cvx_end`. An SDP specification generally includes `cvx`-specific commands for declaring variables, specifying LMI constraints and linear objective functions.

We'll illustrate the process with some examples.

Let's start with the following problem. We want to find a symmetric matrix $P \in \mathbf{R}^{n\times n}$ that satisfies the strict LMIs

$$A^T P + PA < 0, \qquad P > 0,$$

where A is a given square matrix. Of course, you know that this can be done if and only if $\dot{x} = Ax$ is stable; moreover, when A is stable, you can find such a P by solving the Lyapunov equation $A^T P + PA + I = 0$. But we'll solve it using `cvx`. `cvx` doesn't have strict inequalities, but we can get around that. The inequalities above are homogeneous in P, so we can replace them with the nonstrict inequalities

$$A^T P + PA \leq -I, \qquad P \geq I.$$

(Why?) Now we're ready for `cvx`.

Using `cvx`, this problem can be specified as follows:

```
cvx_begin sdp
    variable P(n,n) symmetric
```

```
    A'*P + P*A <= -eye(n)
    P >= eye(n)
cvx_end
```

We're assuming here that the matrix `A`, and the constant `n` (the size of `A`) have already been defined. When Matlab processes this code segment, it forms the SDP and uses a package called SeDuMi to actually solve it. If a P is found, then `P` will be an ordinary numerical matrix, that you can examine. You should check, for example, that it really does satisfy the two LMIs. If there is no such P, then `P`'s entries will be set to `NaN`. You can check the status of the LMI problem by examining the string `cvx_status` after the `cvx_end` command has been processed.

You should try out this segment on a few examples with A stable, and a few with A not stable, to see what happens.

You can add a (linear) objective to the problem if you like. For example, to minimize the $\mathbf{Tr}\, P$ over all P that satisfy the the two LMIs, you can use the `cvx` code

```
cvx_begin sdp
    variable P(n,n) symmetric
    minimize(trace(P))
    A'*P + P*A <= -eye(n)
    P >= eye(n)
cvx_end
```

Our next example is to find a *diagonal* matrix D satisfying the inequalities

$$A^T D + DA \leq -I, \qquad D \geq I.$$

You can solve this with the `cvx` code

```
cvx_begin sdp
    variable D(n,n) diagonal
    A'*D + D*A <= -eye(n)
    D >= eye(n)
cvx_end
```

After running the code, if `cvx_status` contains the string `Solved`, then the variable `D` is a diagonal matrix satisfying the Lyapunov inequality. If no such matrix exists, `cvx_status` will contain the string `Infeasible`.

Now suppose you want to find a symmetric matrix P that satisfies

$$A_1^T P + PA_1 < 0, \qquad A_2^T P + PA_2 < 0, \qquad P > 0,$$

where $A_1 \in \mathbf{R}^{n\times n}$ and $A_2 \in \mathbf{R}^{n\times n}$ are given. We first convert these to the nonstrict LMIs

$$A_1^T P + PA_1 \leq -I, \qquad A_2^T P + PA_2 \leq -I, \qquad P \geq I,$$

and then solve these using the `cvx` code

```
cvx_begin sdp
    variable P(n,n) symmetric
    A1'*P + P*A1 <= -eye(n)
    A2'*P + P*A2 <= -eye(n)
    P >= eye(n)
cvx_end
```

Our next example is based on the bounded-real lemma: We seek a quadratic Lyapunov function $V(x) = x^T P x$ that proves the RMS gain of the linear system $\dot{x} = Ax + Bu$, $y = Cx$, is no more than γ. This reduces to checking whether there exists a $P = P^T$ that satisfies

$$P \geq 0, \qquad \left[\begin{array}{cc} A^T P + PA + C^T C & PB \\ B^T P & -\gamma^2 I \end{array}\right] \leq 0.$$

This problem can be solved by the following `cvx` code:

```
cvx_begin sdp
    variable P(n,n) symmetric
    P >= 0
    [ A'*P + P*A + C'*C           P*B; ...
        B'*P             -gamma^2*eye(m)] <= 0
cvx_end
```

Here we assume that `A`, `B`, `C`, `n`, `m`, and `gamma` are already defined. After this code segment is processed, the string `cvx_status` will tell you whether or not the LMI above is feasible. If it is, then `P` will contain a solution to the LMI.

As an extension on this problem, suppose we want to find the smallest possible value of γ, *i.e.*, the best upper bound on the RMS gain of the linear dynamical system. To do this, we solve the SDP

$$\begin{array}{ll} \text{minimize} & \rho \\ \text{subject to} & P \geq 0 \\ & \left[\begin{array}{cc} A^T P + PA + C^T C & PB \\ B^T P & -\rho I \end{array}\right] \leq 0, \end{array} \tag{1}$$

with variables $P = P^T \in \mathbf{R}^{n\times n}$ and $\rho \in \mathbf{R}$. (The actual gain is given by $\sqrt{\rho}$.)

To solve this SDP, we use the following `cvx` code:

```
cvx_begin sdp
    variable rho
    variable P(n,n) symmetric
    minimize (rho)
    P >= 0
    [ A'*P + P*A + C'*C           P*B; ...
        B'*P             -rho*eye(m)] <= 0
cvx_end
gamma = sqrt(rho);
```

Notes:

- If running a `cvx` script returns an error message, you will need to run `cvx_clear` to clear all active `cvx` data before running any `cvx` script again.
- In most cases, an LMI has many solutions (*i.e.*, assignments of variable values that satisfy the LMI). Any of these is a valid solution. If there is a solution, `cvx` will return *one* of them. The particular one returned need not be the same on different platforms (though it usually is), and can change if you change the problem specification, even in a way that clearly gives an equivalent problem (for example, by changing the order of the constraint statements). Of course, this is *not* an error.

  It can also happen that the solution of an SDP is not unique, in which case the same comments apply.
- If you get an error message that mentions the operator `cvx.times`, it's very likely that you have attempted to form an expression that is not affine in the variables. (For example, you've multiplied two affine expressions.)
- Be careful with symmetry. Remember that LMIs should be in symmetric form.