Lab #1

1. Install and play with CVX[1]. Be sure to learn how to solve a least-squares problem.

2. *Reformulating constaints in* `cvx`. Each of the following `cvx` code fragments describes a convex constraint on the scalar variables `x`, `y`, and `z`, but violates the `cvx` rule set, and so is invalid. Briefly explain why each fragment is invalid. Then, rewrite each one in an equivalent form that conforms to the `cvx` rule set. In your reformulations, you can use linear equality and inequality constraints, and inequalities constructed using `cvx` functions. You can also introduce additional variables, or use LMIs. Be sure to explain (briefly) why your reformulation is equivalent to the original constraint, if it is not obvious.

   Check your reformulations by creating a small problem that includes these constraints, and solving it using `cvx`. Your test problem doesn't have to be feasible; it's enough to verify that `cvx` processes your constraints without error.

   Remark. This *looks* like a problem about 'how to use `cvx` software', or 'tricks for using `cvx`'. But it really checks whether you understand the various composition rules, convex analysis, and constraint reformulation rules.

   (a) `norm( [ x + 2*y , x - y ] ) == 0`

   (b) `square( square( x + y ) ) <= x - y`

   (c) `1/x + 1/y <= 1; x >= 0; y >= 0`

   (d) `norm([ max( x , 1 ) , max( y , 2 ) ]) <= 3*x + y`

   (e) `x*y >= 1; x >= 0; y >= 0`

   (f) `( x + y )^2 / sqrt( y ) <= x - y + 5`

   (g) `x^3 + y^3 <= 1; x>=0; y>=0`

   (h) `x+z <= 1+sqrt(x*y-z^2); x>=0; y>=0`

   Solution:

   (a) The lefthand side is correctly identified as convex, but equality constraints are only valid with affine left and right hand sides. Since the norm of a vector is zero if and only if the vector is zero, we can express the constraint as `x+2*y==0; x-y==0`. We can also write it as `norm([x+2*y,x-y]) <= 0`.

[1] At `http://www.stanford.edu/~boyd/cvx/`. If you do not have access to matlab, check out the Python version, CVXOPT.

(b) The problem is that `square()` can only accept affine arguments, because it is convex, but not increasing. To correct this use `square_pos()` instead:

```
square_pos( square( x + y ) ) <= x - y
```

We can also reformulate this constraint by introducing an additional variable.

```
variable t
square( x+y ) <= t;
square( t ) <= x - y
```

Note that, in general, decomposing the objective by introducing new variables doesn't need to work. It works in this case because the outer `square` function is convex and monotonic over $\mathbf{R}_+$.

(c) $1/x$ isn't convex, unless you restrict the domain to $\mathbf{R}_{++}$. We can write this one as `inv_pos(x)+inv_pos(y)<=1`.

(d) The problem is that `norm()` can only accept affine argument since it is convex but not increasing. One way to correct this is to introduce new variables `u` and `v`:

```
norm( [ u , v ] ) <= 3*x + y;
max( x , 1 ) <= u;
max( y , 2 ) <= v;
```

Decomposing the objective by introducing new variables work here because `norm` is convex and monotonic over $\mathbf{R}_+^2$, and in particular over $(1,\infty)\times(2,\infty)$.

(e) xy isn't concave, so this isn't going to work as stated. But we can express the constraint as `x>=inv_pos(y)`. (You can switch around `x` and `y` here.) Another solution is to write the constraint as `geomean([x,y])>=1`. We can also give an LMI representation:

```
[ x 1; 1 y ] == semidefinite(2)
```

(f) This fails when we attempt to divide a convex function by a concave one. We can write this as

```
quad_over_lin(x+y,sqrt(y)) <= x-y+5
```

This works because `quad_over_lin` is monotone descreasing in the second argument, so it can accept a concave function here, and `sqrt` is concave.

(g) The function x^3+y^3 is convex for $x\geq 0$, $y\geq 0$. But x^3 isn't convex for $x<0$, so `cvx` is going to reject this statement. One way to rewrite this constraint is

```
quad_pos_over_lin(square(x),x) + quad_pos_over_lin(square(y),y) <= 1
```

This works because `quad_pos_over_lin` is convex and increasing in its first argument, hence accepts a convex function in its first argument. (The function `quad_over_lin`, however, is not increasing in its first argument, and so won't work.)

We can also give an LMI representation, introducing some new variables:

```
    variables s t u v
    [s t; t x] == semidefinite(2); % s>=t^2/x; x>=0
    [u v; v y] == semidefinite(2); % u>=v^2/y; y>=0
    [t x; x 1] == semidefinite(2); % t>=x^2
    [v y; y 1] == semidefinite(2); % v>=y^2
    s+u<= 1;
```

We can check our reformulations by writing this following feasibility problem in `cvx` (which is obviously infeasible)

```
cvx_begin
    variables x y u v
    x + 2*y == 0;
    x - y == 0;
    square_pos( square ( x + y ) ) <= x - y;
    inv_pos(x) + inv_pos(y)  <= 1;
    norm( [ u ; v ] ) <= 3*x + y;
    max( x , 1 ) <= u;
    max( y , 2 ) <= v;
    x >= inv_pos(y);
    x >= 0;
    y >= 0;
    quad_over_lin(x + y , sqrt(y)) <= x - y + 5;
    quad_pos_over_lin(square(x),x) + quad_pos_over_lin(square(y),y) <= 1;
cvx_end
```