LMIs and Schur Formula, example: minimum distance between ellipses, generalization to **LMI sets** (intersection of ellipsoids, for simplicity)

Objective: calculate the minimum distance (and the points that achieve it) between two general LMI sets (well, the simplest example is intersection of ellipsoids).

Presentation in video:

https://personales.upv.es/asala/YT/V/distlmisetEN.html

*Let's load the necessary toolboxes (Yalmip, Sedumi in this case).

```
if ~exist("yalmiptest")
    addpath("~/bin/MATLAB/tbxmanager/")
    tbxmanager restorepath
end
```

```
Toolbox "sedumi:1.3:glnxa64" added to the Matlab path.
Toolbox "yalmip:R20230609:all" added to the Matlab path.
Toolbox "mpt:3.2.1:all" added to the Matlab path.
Toolbox "mptdoc:3.0.4:all" added to the Matlab path.
Toolbox "cddmex:1.0.1:glnxa64" added to the Matlab path.
Toolbox "fourier:1.0:glnxa64" added to the Matlab path.
Toolbox "glpkmex:1.0:glnxa64" added to the Matlab path.
Toolbox "hysdel:2.0.6:glnxa64" added to the Matlab path.
Toolbox "lcp:1.0.3:glnxa64" added to the Matlab path.
Toolbox "espresso:1.0:glnxa64" added to the Matlab path.
```

Problem statement

Let us define ellipses as $(x-c)^T \cdot P \cdot (x-c) \leq 1$; or with the equivalent inverse form (equiv. if non degenerate): $(x-c)^T \cdot Q^{-1} \cdot (x-c) \leq 1$. Both P and Q are assumed to be positive definite matrices.

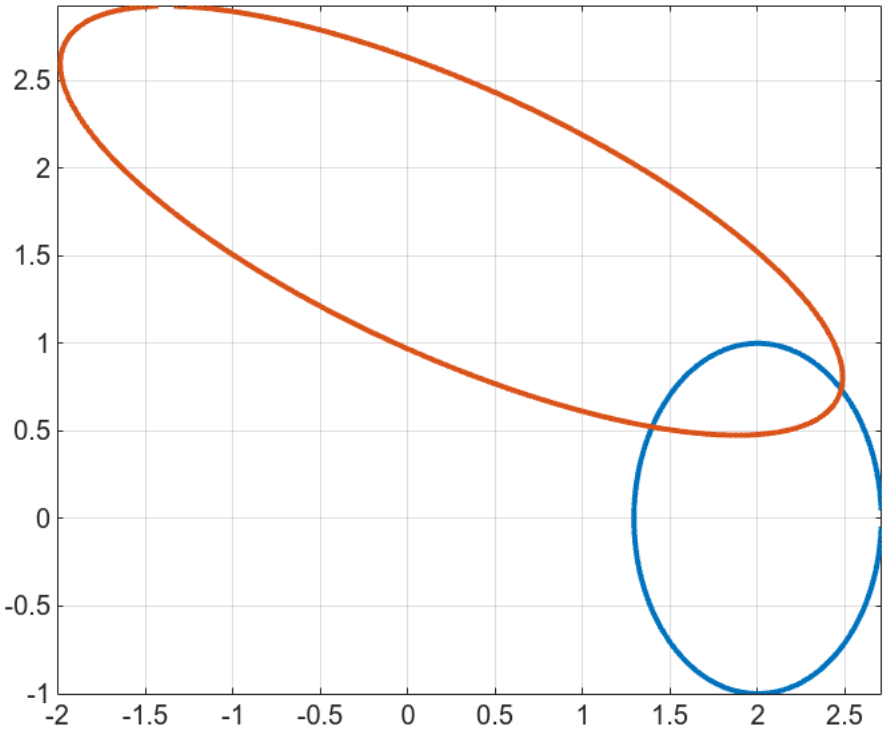
```
c{1}=[2;0];
Q{1}=[0.5 0;0 1];

c{2}=[0.25;1.7];
Q{2}=[5 -2;-2 1.5];
```

Shape matrix Q is "proportional" to the square "size" of the ellipsoid, P es "inverse proportional".

```
for k=1:2
    P{k}=inv(Q{k});
    fimplicit(@(x,y) ([x;y]-c{k})'*P{k}*([x;y]-c{k})-1,LineWidth=2), hold
on
end
```

```
Warning: MATLAB has disabled some advanced graphics rendering features by switching to software
OpenGL. For more information, click here.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
```

```
hold off, grid on, axis equal
```



Schur's formula: $A - B^T Q^{-1} B \succ 0$ and

$$Q \succ 0 \quad \text{iff} \quad \begin{pmatrix} A & B^T \\ B & Q \end{pmatrix} \succ 0.$$

Identifying $B \equiv (x-c)$, $A = 1$, we have that $1 - (x-c)^T Q^{-1} (x-c) > 0$ and $Q \succ 0$ if and only if:

$$\begin{pmatrix} 1 & x^T - c^T \\ x - c & Q \end{pmatrix} \succ 0$$

Let's introduce a search point on each ellipse as decision variable:

$$\begin{pmatrix} 1 & x_1^T - c_1^T \\ x_1 - c_1 & Q_1 \end{pmatrix} \succ 0 \qquad \begin{pmatrix} 1 & x_2^T - c_2^T \\ x_2 - c_2 & Q_2 \end{pmatrix} \succ 0$$

```
x{1}=sdpvar(2,1);
x{2}=sdpvar(2,1);
LMIs=[];
for k=1:2
    LMIs=[LMIs; ...
        [1 x{k}'-c{k}'; x{k}-c{k}  Q{k}]>=0 ];
end
```

Distance between two points being smaller than d amounts to saying $d^2 > \|x_1 - x_2\|^2 = (x_1 - x_2)^T(x_1 - x_2)$, i.e., $d^2 - (x_1 - x_2)^T(x_1 - x_2) > 0$, thus Schur formula says:

$$\begin{pmatrix} d^2 & x_1^T - x_2^T \\ x_1 - x_2 & I_{2\times 2} \end{pmatrix} \succ 0$$

We will add that to the LMIs, changing d^2 nonlinear to "dsquared= d^2" linear in dsquared.

```
dsquared=sdpvar();
LMIs=[LMIs; ...
    [dsquared x{1}'-x{2}'; x{1}-x{2} eye(2)]>=0 ];
```

If we look for the minimum distance, we have to find the minimum dsquared for which there are feasible points x{1} and x{2}.

```
sol=optimize(LMIs,dsquared)
```

```
SeDuMi 1.3 by AdvOL, 2005-2008 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
eqs m = 5, order n = 10, dim = 28, blocks = 4
nnz(A) = 9 + 0, nnz(ADA) = 25, nnz(L) = 15
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            1.27E+01 0.000
  1 :   1.43E-01 3.50E+00 0.000 0.2759 0.9000 0.9000   1.82  1  1  1.2E+00
  2 :   5.91E-02 9.20E-01 0.000 0.2630 0.9000 0.9000   2.00  1  1  4.8E-01
  3 :   1.46E-02 1.74E-01 0.000 0.1893 0.9000 0.9000   1.31  1  1  3.5E-01
  4 :   1.27E-03 9.09E-03 0.000 0.0522 0.9900 0.9900   1.08  1  1  3.0E-01
  5 :   8.36E-07 7.89E-06 0.401 0.0009 0.9999 0.9999   1.01  1  1  2.6E-03
  6 :  -1.12E-07 1.62E-06 0.060 0.2060 0.9000 0.9000   1.00  1  1  5.3E-04
  7 :  -1.14E-07 6.56E-07 0.133 0.4037 0.9000 0.9000   1.00  3  1  2.1E-04
  8 :   3.54E-09 1.68E-07 0.000 0.2555 0.9000 0.9000   1.00  3  3  5.3E-05
  9 :  -7.30E-10 8.00E-09 0.000 0.0478 0.9900 0.9900   1.00  1  1  2.5E-06
 10 :   4.36E-11 4.68E-10 0.000 0.0585 0.9900 0.9900   1.00  3  3  1.4E-07
 11 :  -1.33E-12 3.15E-11 0.256 0.0674 0.9900 0.9900   1.00  1  4  9.7E-09
 12 :   3.36E-13 6.07E-12 0.000 0.1925 0.9000 0.9000   1.00  4  5  1.9E-09
 13 :  -4.05E-15 2.00E-13 0.000 0.0329 0.9900 0.9900   1.00  2  5  6.1E-11

iter seconds digits       c*x               b*y
 13      0.6   8.2  1.7195162733e-13 -4.0544141392e-15
|Ax-b| =   9.3e-15, [Ay-c]_+ =   1.5E-14, |x|=  1.0e+00, |y|=  2.9e+00

Detailed timing (sec)
   Pre          IPM          Post
1.202E-01    4.034E-01    4.432E-02
Max-norms: ||b||=1, ||c|| = 5,
Cholesky |add|=1, |skip| = 0, ||L.L|| = 245898.
sol = struct with fields:
    yalmipversion: '20230622'
    matlabversion: '24.1.0.2653294 (R2024a) Update 5'
       yalmiptime: 1.1751
       solvertime: 0.6205
             info: 'Successfully solved (SeDuMi)'
          problem: 0
```

```
dsquared
```

```
Linear scalar (real, 1 variable)
Current value: 4.0544e-15
Coefficients range: 1 to 1
```

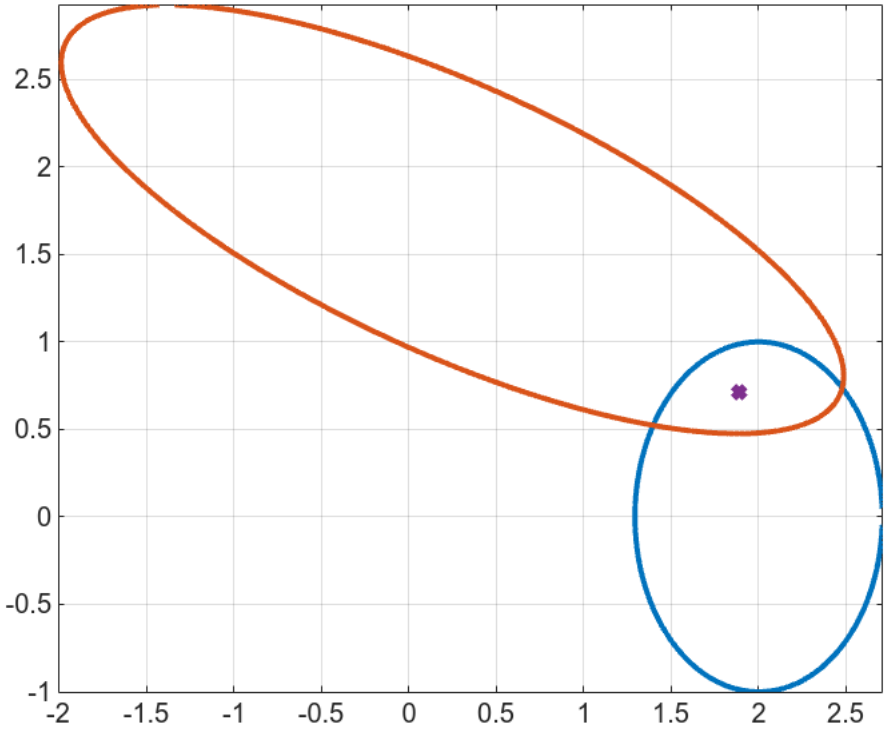
```
sqrt(value(dsquared))
```

```
ans = 6.3674e-08
```

```
solx{1}=value(x{1});
solx{2}=value(x{2});
for k=1:2
    fimplicit(@(x,y) ([x;y]-c{k})'*P{k}*([x;y]-c{k})-1,LineWidth=2), hold
on
end
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
```

```
for k=1:2
    plot(solx{k}(1),solx{k}(2),'x',LineWidth=3)
end
hold off, grid on, axis equal
```



Generalization: minimum distance between "LMI sets"

If the sets are representable with LMI constraints, we can calculate their minimum distance... Intersections of ellipses, direct sums, convex hulls, and other convex sets with polynomial boundaries [https://homepages.laas.fr/henrion/courses/edsys04/edsys04_3.pdf Didier Henrion, LAAS, Tolouse]

Well, some of these sets are "lifted LMI sets", lower-dimensional projections of "true" LMI sets. Details out of the scope of this introductory material.

Let's introduce a search point on each ellipse as decision variable and let's add that the set "1" is the INTERSECTION of the previous ellipse and another one:

$$\begin{pmatrix} 1 & x_1^T - c_1^T \\ x_1 - c_1 & Q_1 \end{pmatrix} \succ 0 \qquad \begin{pmatrix} 1 & x_2^T - c_2^T \\ x_2 - c_2 & Q_2 \end{pmatrix} \succ 0$$

$$\begin{pmatrix} 1 & x_1^T - c_3^T \\ x_1 - c_3 & Q_3 \end{pmatrix} \succ 0$$

So very minor changes to the code solve the problem: just add the last LMI to the constraints and repeat optimization.

```
c{3}=[3;-.2];
Q{3}=[1 0.2;0.2 0.5]; P{3}=inv(Q{3});
Set1Func=@(xi) max((xi-c{1})'*P{1}*(xi-c{1}), (xi-c{3})'*P{3}*(xi-c{3}))-1
```

```
Set1Func = function_handle with value:
    @(xi)max((xi-c{1})'*P{1}*(xi-c{1}),(xi-c{3})'*P{3}*(xi-c{3}))-1
```

```
fimplicit(@(x,y) ([x;y]-c{1})'*P{1}*([x;y]-c{1})-1,':'), hold on
```

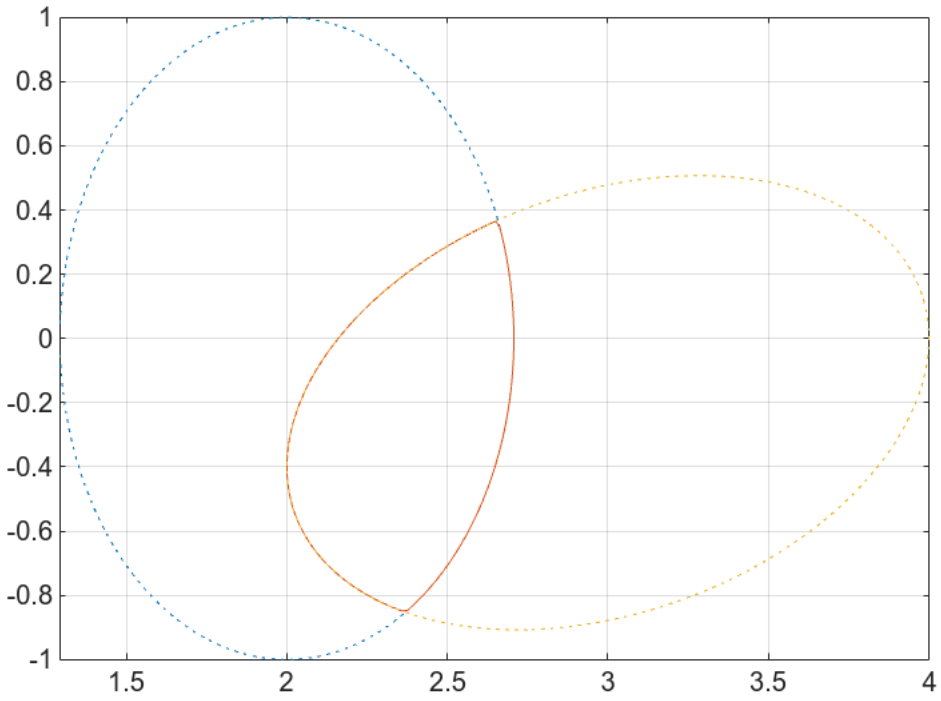
```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
```

```
fimplicit(@(x,y) Set1Func([x;y])), grid on, axis equal
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
```

```
fimplicit(@(x,y) ([x;y]-c{3})'*P{3}*([x;y]-c{3})-1,':'), hold off
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
```



We add the extra LMI here:

```
LMIs=[LMIs; [1 x{1}'-c{3}'; x{1}-c{3}  Q{3}]>=0 ];
```

And, redoing optimization, we get:

```
sol3=optimize(LMIs,dsquared)
```

```
SeDuMi 1.3 by AdvOL, 2005-2008 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
eqs m = 5, order n = 13, dim = 37, blocks = 5
nnz(A) = 11 + 0, nnz(ADA) = 25, nnz(L) = 15
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            1.16E+01 0.000
  1 :   3.09E-01 3.85E+00 0.000 0.3316 0.9000 0.9000   2.13  1  1  1.2E+00
  2 :   1.29E-02 7.43E-01 0.000 0.1928 0.9000 0.9000   2.04  1  1  4.4E-01
  3 :  -8.72E-02 5.64E-02 0.000 0.0758 0.9900 0.9900   1.18  1  1  2.1E-01
  4 :  -1.00E-01 1.14E-02 0.000 0.2022 0.9000 0.9000   1.01  1  1  3.7E-02
  5 :  -1.03E-01 2.20E-03 0.000 0.1929 0.9000 0.9000   1.01  1  1  7.1E-03
  6 :  -1.04E-01 1.40E-04 0.000 0.0637 0.9900 0.9900   1.00  1  1  4.7E-04
  7 :  -1.04E-01 1.22E-05 0.148 0.0870 0.9900 0.9900   1.00  1  1  4.2E-05
  8 :  -1.04E-01 7.87E-07 0.024 0.0647 0.9900 0.9900   1.00  1  1  2.9E-06
  9 :  -1.04E-01 7.25E-08 0.099 0.0921 0.9900 0.9900   1.00  1  1  2.6E-07
 10 :  -1.04E-01 1.67E-08 0.000 0.2304 0.9000 0.9000   1.00  2  2  6.1E-08
 11 :  -1.04E-01 7.17E-10 0.000 0.0429 0.9900 0.9900   1.00  2  2  2.6E-09
 12 :  -1.04E-01 1.78E-10 0.000 0.2487 0.9000 0.9000   1.00  2  3  6.6E-10

iter seconds digits       c*x               b*y
 12      0.1   8.7 -1.0353545475e-01 -1.0353545494e-01
|Ax-b| =   5.2e-12, [Ay-c]_+ =   3.6E-11, |x|=  1.5e+00, |y|=  3.5e+00

Detailed timing (sec)
   Pre          IPM          Post
1.653E-02    6.956E-02    3.621E-03
Max-norms: ||b||=1, ||c|| = 6,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 9869.78.
```

```
sol3 = struct with fields:
    yalmipversion: '20230622'
    matlabversion: '24.1.0.2653294 (R2024a) Update 5'
       yalmiptime: 0.3221
       solvertime: 0.0918
             info: 'Successfully solved (SeDuMi)'
          problem: 0
```

```
value(dsquared)
```

```
ans = 0.1035
```

```
sqrt(value(dsquared))
```

```
ans = 0.3218
```

```
solx{1}=value(x{1});
solx{2}=value(x{2});

fimplicit(@(x,y) ([x;y]-c{1})'*P{1}*([x;y]-c{1})-1,':'), hold on
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
```

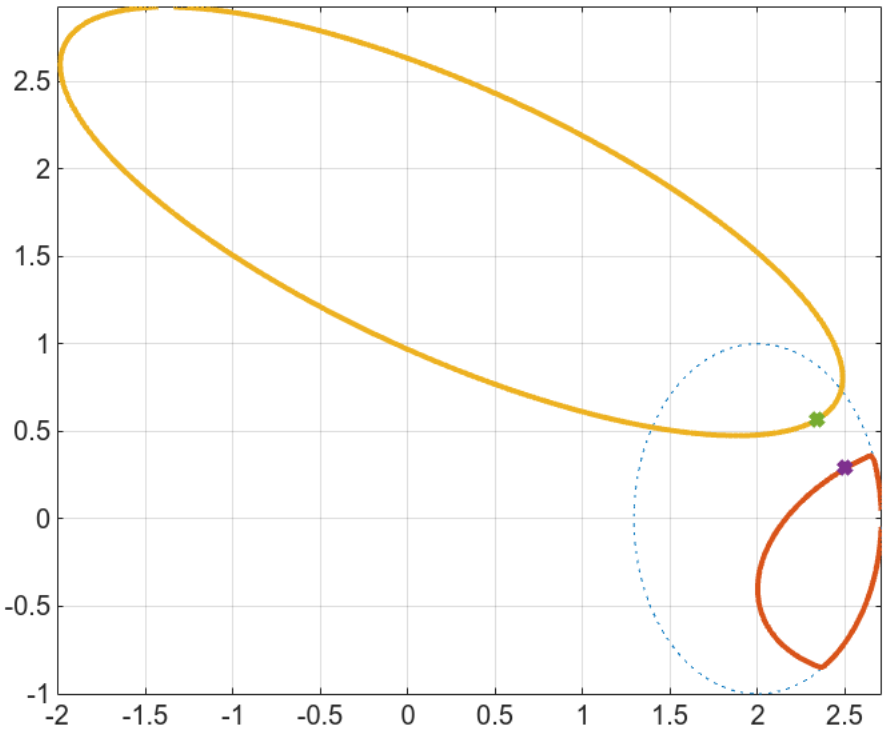
```
fimplicit(@(x,y) Set1Func([x;y]),LineWidth=2)
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
```

```
fimplicit(@(x,y) ([x;y]-c{2})'*P{2}*([x;y]-c{2})-1,LineWidth=2), hold on
```

```
Warning: Function behaves unexpectedly on array inputs. To improve performance, properly
vectorize your function to return an output with the same size and shape as the input arguments.
```

```
for k=1:2
    plot(solx{k}(1),solx{k}(2),'x',LineWidth=3)
end
hold off, grid on, axis equal
```


2.5
2
1.5
1
0.5
0
-0.5
-1
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5