Laboratory Session 1

Basics on set operations and optimization

Ionela Prodan

2024

Contents

Throughout the Matlab/Python sessions we will make use of several toolboxes for set manipulation, visualization and the solving of optimization problems. These toolboxes are MPT3 and Yalmip in Matlab and CasADi in both Matlab and Python.

- MPT3 can be downloaded and installed from `https://www.mpt3.org/` and it covers the installation of Yalmip as well;
- CasADi can be intalled from `https://web.casadi.org/get/`
- For Python you can use the following commands: in conda prompt / conda install -c conda-forge casadi / conda update anaconda / conda install spyder=5.1.5 / in spyder : pip install casadi

1. Set manipulation and visualization

1.1. Polyhedra definitions in Matlab

The Multiparametric Toolbox (MPT) https://www.mpt3.org/ is a collection of algorithms for modeling, control, analysis and deployment of constrained optimal controllers developed under Matlab. It features a powerful geometric library that extends the application of the toolbox beyond optimal control to various problems arising in computational geometry. In this course, the MPT toolbox will be extensively used such that basic operations involving polyhedral[1] set manipulation are detailed:

- Creation of polytopes (in both half-space and generator representation).
- Basic manipulation of polytopes.
- Computational geometry tools.
- Visualization.
- Accessing internal information stored in the polytope object.

Note that the Matlab commands *methods* and *help* can be used to observe additional information about the methods of the Polyhedron class:

```
methods(Polyhedron) % returns a list of all the methods of class Polyhedron
help Polyhedron.dual % returns information about the 'dual' method of class
    Polyhedron
```

Creation of a polytope

A Polytope is defined as a bounded intersection of a finite number of hyperplanes. In general, a polytope object can be initialized in two ways:

- Providing the H-representation (i.e. intersecting half-spaces),
- By computing the generator representation (convex hull of extreme points (vertices) added to the cone of rays)

Let us now create a polytope using the H-representation:

$$P = \{x :\ Ax \leq b,\ A_e x = b_e\} \tag{1}$$

Such a polytope will be handled in MPT3 via the following commands:

```
P = Polyhedron(A,b)
Pe= Polyhedron('A',A,'b',b,'Ae',Ae,'be',be) %explicitly give the
    inequalities and equalities of the half-space representation
```

[1]We will abuse the notation and use interchangeably the ‘polytope’ and ‘polyhedron’ notions. Strictly speaking the former is the bounded case of the latter.

For practice: choose your own values for matrices A, b, A_e and b_e and observe the result. Try to obtain bounded and unbounded polyhedral objects and analyze the content of the polyhedral object.

Repeat the construction starting from the dual representation of generators:

$$P = \{x = \sum_i \alpha_i v_i + \sum_j \beta_j r_j, \sum_i \alpha_i = 1, \alpha_i, \beta_j \geq 0\} \tag{2}$$

(v_i- vertices and r_i- rays) handled in MPT 3 via the following commands:

```
P = Polyhedron(V) %input V: the vertices of the generator representation (
    row-wise)
P= Polyhedron('V',V, 'R',R) % give explicitly the vertices V and rays R of
    the generator representation
```

Note that the algorithm computes the convex hull and in the process removes the redundant vertices/rays.

In both cases you can call the method

```
plot(P)
```

which allows the graphical representation of the object (it gets more complicated in higher dimensions).

An array of polytopes can be obtained by concatenation:

```
Pall=[P1 P2]
plot(Pall)
```

Pall can be further manipulated as a union of convex bodies (all the operations which can be applied to single polytope[2]).

For practice:

- create polytopes with randomly given matrices and plot them (if randomly generated, not all the input points V will be vertices and not all half-spaces described by (A, b) are facets)
- check the numerical values characterizing the polytopes obtained (e.g., $P.A$ retrieves the left-side matrix from the H-representation)
- test the previous observations in higher dimensions

1.2. Polyhedral operations

MPT3 implements all standard geometrical operations:

set inclusion returns 1 if the logic operation is true, 0 otherwise

set intersection $P \cap Q = \{x : x \in P, x \in Q\}$

[2] An exception is the image via affine mappings which apply for one polytope but not for the concatenated structure.

Minkowski addition $P \oplus Q = \{x + y : \ x \in P, y \in Q\}$

Pontryagin difference $P \ominus Q = \{x : \ x + y \in P, \forall y \in Q\}$

Hausdorff distance $d(P, Q) = \min_{x \in P} \max_{y \in Q} d(x, y)$

```
P1<=P2 %test for inclusion
P1==P2 %test for equality (two-way inclusion)
P1.intersect(P) %returns the result of the intersection
P1+P2 %returns the Minkowski addition
P1-P2 % returns the Pontryagin difference
P1.distance(P2) % returns the Hausdorff distance
P1\P2 % computes the set difference
```

For practice:

- generate randomly two polytopes and depict their inclusion, addition and difference (to have a non-empty difference you should have one of the polytopes included in the other)
- depict the Hausdorff distance (i.e., the segment linking the two points whose inter-distance is the smallest); to have a non-empty distance you should take the polytopes such that they do not intersect

1.3. Polyhedra visualisation in Python

The necessary libaries in Python are in the following:

```
import numpy as np
import casadi as cas
from scipy.spatial import ConvexHull
from scipy.spatial import HalfspaceIntersection
import matplotlib.pyplot as plt
import polytope as pc
```

You may find the vertices of a half-space representation Polyhedron as follows:

```
def get_vertices_Ab(A,b,feasible_point):
    Ab = np.column_stack([A,-b])
    Hs = HalfspaceIntersection(Ab, feasible_point)
    x, y = zip(*Hs.intersections)
    points = np.stack([x,y],axis=1)
    Pv = ConvexHull(points)
    vertices = np.empty((0,2))
    for n in Pv.vertices:
        vertices = np.vstack((vertices,points[n]))
    return vertices
```

You may find the vertices of a polyhedron from random points:

```
  def get_vertices_V(points):
    Pv = ConvexHull(points)
    vertices = np.empty((0,2))
    for n in Pv.vertices:
        vertices = np.vstack((vertices,points[n]))
    return vertices
```

Define the polytope from the half-space representation:

```
F = np.vstack([np.eye(2,2), -np.eye(2,2)])
g = np.ones([4,1])
Ph = pc.Polytope(F,g)
print(' Polytop half-space representation:\n', Ph)

feasible_point = np.array([0,0])
vertices = get_vertices_Ab(Ph.A,Ph.b,feasible_point)
plt.figure()
polygon = plt.Polygon(vertices, color='r', alpha=0.4, linewidth = 2, label=
    'Ph')
plt.gca().add_patch(polygon)

plt.axis([-2,2,-2,2])

plt.title('Polytope given in halfspace representation')
plt.legend()
# Show the plot
plt.grid()
plt.show()
```

Define the polytope from the vertices representation

```
V = np.array([[0.2, -0.2], [0.2, 0.4] , [-0.3, 0.4], [-0.3, -0.5]])
Pv = pc.qhull(V)
print(Pv)
Pv2_vertices = get_vertices_V(V)
print('Vertices by pc.qhull:', Pv.vertices)
print('Vertices by ConvexHull:',Pv2_vertices)
plt.figure()
polygon = plt.Polygon(Pv2_vertices, color='r', alpha=0.4, linewidth = 2,
    label='Pv')
plt.gca().add_patch(polygon)

plt.axis([-2,2,-2,2])
plt.title('Polytope given in vertices representation')
plt.legend()
# Show the plot
plt.grid()
plt.show()
```

2. Solving optimization problems

2.1. Optimization problems with Yalmip in Matlab

Yalmip toolbox [3] allows to write optimization problems (cost and constraints) in a compact manner by hiding the complexities of constraint concatenation (thus saving time and errors).

Some of the common commands are exemplified here by solving:

$$\min_x \|x\|^2 \text{ subject to } A * x <= b$$

The code for solving this QP prolem is in the following:

```
x=sdpvar(2,1) % generation of symbolic decision variable
%a=binvar(2,1) % generation of binary symbolic variable
constraints=[A*x<=b];
objective=x'*x;
options=[];
sol = optimize(constraints,objective,options); % do the optimization
    problem with constraints and cost
double(x); % retrieve the numerical value of the solution
```

These commands can be used to represent more complex optimization problems. For practice we suggest the following tests:

1. Generate a polytope and use it as constraint for the state $(P.A \cdot x \leq P.b)$
2. Take a linear cost $(c^\top \cdot x)$ and solve the resulting optimization problem; depict the constraint set, the constrained optimum x^* and the cost function sub-level set (in this case the hyperplane $c^\top \cdot x = c^\top \cdot x^*$)
3. Repeat for the quadratic case where the cost is given as $\frac{1}{2}x^\top Hx + f^\top x$ with $H \succ 0$; note that here the sub-level is an ellipsoidal set centered in the un-constrained optimimum $-H^{-1}f$ since $\frac{1}{2}x^\top Hx + f^\top x = (x + H^{-1}f)^\top \frac{H}{2}(x + H^{-1}f) - f^\top H^{-1}f$

The Matlab script covering this introductory material is found in Listing 1. They are intended only for the case you encounter difficulties!

2.2. Optimization problems with CasADi in Matlab

Solve an LP problem:

```
import casadi.*
solver = casadi.Opti();
x = solver.variable(2,1);
constraints = Pcon.A * x <= Pcon.b;
objective = c' * x;
options.ipopt.print_level = 0;
options.ipopt.sb = 'yes';
options.print_time = 0;

```

```
solver.subject_to(constraints)
solver.minimize(objective)
solver.solver('ipopt', options);

timer = tic;
solution = solver.solve();
time2 = toc(timer);

xcas = solution.value(solver.x);
fprintf('LP solution using CasADi = [%f %f]',xcas(1), xcas(2));
```

For plotting the result of the LP optimization problem together with the cost and constrains visualisation you can follow this code:

```
figure
plot(Pcon, opt{:})
hold on
scatter(xcas(1), xcas(2), 50,'filled','r')

a = 2*axis;
axis(a)

plot(a([1 2]), ((c'*xcas(:) - c(1) .* a([1 2])) ./ c(2)),'k')
for i = 0:0.5:2
    if i ~= 1
        plot(a([1 2]), ((i*c'*xcas(:) - c(1) .* a([1 2])) ./ c(2)),
    linestyle = '--')
    end
end
arrow([0,0], c)

title('Linear cost constrained optimization problem')
legend('Constraints', 'Optimal solution','Linear cost', location= 'best')
```

Solve a QP optimization probelm:

```
import casadi.*
solver = casadi.Opti();
x = solver.variable(2,1);
constraints = Pcon.A * x <= Pcon.b;
objective2 = x'*H*x / 2 + f'*x;
options.ipopt.print_level = 0;
options.ipopt.sb = 'yes';
options.print_time = 0;

solver.subject_to(constraints)
solver.minimize(objective2)
solver.solver('ipopt', options);

timer = tic;
solution = solver.solve();
time2 = toc(timer)

xcas2 = solution.value(solver.x);
```

```
fprintf('QP solution using CasADi = [%f %f]',xcas2(1), xcas2(2));
```

For plotting the result of the QP optimization problem together with the cost and constrains visualisation you can follow this code:

```
    xunc = -inv(H)*f;
fprintf('Unconstrained LP solution using CasADi = [%f %f]',xunc(1), xunc(2)
    );
Cost function value comparison
Func = (xunc'*H*xunc/2+f'*xunc);
Fquad = (xcas2'*H*xcas2/2+f'*xcas2);
fprintf('Func = %f \nFquad = %f',Func, Fquad);
Generate the quadratic cost in form of an ellipse
[xxquad, yyquad] = createEllipse(H,f,Fquad);
[xxquad2, yyquad2] = createEllipse(H,f,Fquad*(-2));
[xxquad3, yyquad3] = createEllipse(H,f,Fquad*(0.5));
Plot the results
figure
hold on
h4 = plot(xxquad,yyquad,'k');
h5 = plot(xxquad2,yyquad2,Linestyle = '--');
h6 = plot(xxquad3,yyquad3,Linestyle = '--');
h2 = scatter(xcas2(1), xcas2(2), 50, 'filled','r');
h3 = scatter(xunc(1), xunc(2), 25, 'filled','g');
h1 = plot(Pcon, opt{:});

title('Quadratic cost constrained optimization problem')
legend([h1 h2 h3 h4],'Constraints', 'Constrained optimum',...
    'Unconstrained optimum','Quadratic cost', 'location','best')
```

2.3. Optimization problems with CasADi in Python

We propose you the implementation of an LP optimization problem. You can follow this example and propose yourself the implementation of a QP optimization problem.

```
    c = np.random.rand(2,1) -0.5
solver = cas.Opti()
x = solver.variable(2,1)
#constraints = cas.mtimes(myPcon['A'], x) <= myPcon['b']
constraints = cas.mtimes(Pcon.A, x) <= Pcon.b
objective = cas.mtimes(c.T,x)
options = {'ipopt': {'print_level': 0, 'sb': 'yes'}, 'print_time': 0}

solver.subject_to(constraints)
solver.minimize(objective)
solver.solver('ipopt', options)
%%time
solution = solver.solve()

xcas = solution.value(solver.x)
print('LP solution using CasADi = ',xcas)

Plot the results
```

```

myPcon['V']
array([[-0.3,  0.4],
       [-0.3, -0.5],
       [ 0.2, -0.2],
       [ 0.2,  0.4]])
plt.figure()
polygon = plt.Polygon(myPcon['V'], color='b', alpha=0.4, linewidth = 2,
    label='Constraints')
plt.gca().add_patch(polygon)
plt.scatter(xcas[0], xcas[1], s=40, c='r', label='Optimal solution')

a = np.array([i*2 for i in plt.axis()])
plt.axis(a)

x = a[0:2]
y = (np.dot(c.T, xcas) - c[0] * a[0:2]) / c[1]
plt.plot(x, y, label='Linear cost function', color='black')

# Plot additional lines for different values of 'i'
for i in np.arange(0, 2.1, 0.5):
    if i !=1 :
        y_i = (i * np.dot(c.T ,xcas) - c[0] * a[0:2]) / c[1]
        plt.plot(x, y_i, linestyle = '--')

plt.title('Linear Cost Constrained Optimization Problem')
plt.legend()

# Show the plot
plt.grid()
plt.show()
```

3. Exercises (optional)

The next exercises are intended to end up with the construction of "your own" invariant set approximations (recall theoretical definitions in [1] and their use in control theory as for example MPC, collision and obstacle avoidance for multi-agent systems and the like). They should be constructed for any n-dimensional discrete-time LTI system but for exemplification we propose the next benchmarks:

- Double integrator
- Complex eigenvalues (real-valued matrix but complex conjugate eigenvalues)
- 3-dimensional state space model
- 2-dimensional state space model

The numerical data is found in Listing 2–5.

Exercise 3.1. Consider a LTI dynamical system afected by bounded additive disturbances (characterized by the matrices A, B and $\bar{\delta}$). Construct the ultimate bounds set [2] given by

$$\Omega_{UB}(\epsilon) = \left\{x : \ |V^{-1}x| \leq (I - |\Lambda|)^{-1}|V^{-1}B|\bar{\delta} + \epsilon\right\} \tag{3}$$

where V, Λ are computed by executing $[V, \Lambda] = jordan(A)$. Show graphically that the set is robust positively invariant. [3]

References

[1] F. Blanchini. "Set invariance in control–a survey". In: *Automatica* 35.11 (1999), pp. 1747–1767.

[2] E. Kofman, H. Haimovich, and M.M. Seron. "A systematic method to obtain ultimate bounds for perturbed systems". In: *International Journal of Control* 80.2 (2007), pp. 167–178.

[3] Johan Löfberg. "YALMIP: A toolbox for modeling and optimization in MATLAB". In: *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*. IEEE. 2004, pp. 284–289.

[3]A solution is depicted in Listing 6.

A. Annexes

A.1. Matlab script for the introductory material

```
%% initialization data
close all
clc
clear all

opt={'Color', 'b', 'Alpha', 0.4}; %prettify the output

%% define a polytope from the half-space representation
F = [eye(2); -eye(2)];
g = [1; 1; 1; 1];
Feq=ones(1,2);
geq=0;

Ph = Polyhedron(F,g);
Pheq=Polyhedron('A',F,'b',g, 'Ae',Feq,'be',geq) %explicitly give the
    inequalities and equalities of the half-space representation
plot(Ph,opt{:})
title('polytopes given in H-representation')
hold on
plot(Pheq,opt{:})
hold off
%% define a polytope starting from the vertex and rays representation
figure
V = [0.2 -0.2; 0.2 0.4; -0.3 0.4; -0.3 -0.5];
Pv = Polyhedron(V); % a bounded polyhedron since it only has vertices

R=[1 1; 1 3];
Pvr=Polyhedron('V',V,'R',R); % unbounded polyhedron with vertices and rays

plot(Pv,opt{:})
hold on
plot(Pvr,opt{:})
title('polytopes given in generator representation')

%% construct an array of polytopes
figure
Ph2=Ph+[5;2];
Pall=[Ph Ph2]
plot(Pall,opt{:})
title('plot union of polytopes')
%% retrieve information from the Polyhedron object

% get the extreme points
vert = Ph.V
% get the constraints of the half-space representation
Pvr.A
Pvr.b
%% generate polyhedron from random distribution of vertices
Points=2*rand(30,2)-1
```

```
Prandom = Polyhedron(Points)
figure
plot(Prandom,opt{:})
hold on
plot (Points(:,1),Points(:,2),'*');

%% polytope operations
P1=Polyhedron(rand(5,2)-0.5);
P2=Polyhedron(rand(4,2)-0.5)+[1;1];

P1<=P2 %test for inclusion
P1==P2 %test for equality (two-way inclusion)

figure
plot(P1.intersect(P2),opt{:}) %returns the intersection between two
    polytopes
title('polytope intersection')

figure
plot(P1+P2,opt{:}) % Minkowski addition
title('Minkowski addition')

figure
plot(P1-P2,opt{:}) % Pontryagin difference
title('Pontryagin difference')

dist=P1.distance(P2); % obtain the Hausdorff distance

% plot the two polytopes and the segment denoting the Hausdorff distance
figure
plot([P1 P2],opt{:})
hold on
scatter([dist.x(1) dist.y(1)],[dist.x(2) dist.y(2)],'filed','r')
plot([dist.x(1) dist.y(1)],[dist.x(2) dist.y(2)],'r','Linewidth',2) %plot
    the distance between the two polytopes
title('Hausdorff distance')
hold off

%% Yalmip linear cost constrained optimization problem

yalmip('solver','sedumi')

Pcon=Polyhedron(1-2*rand(5,2)); %define a polytope with 5 vertices taken
    randomly (hopefully bounded)
c=rand(2,1)-0.5;
% construct the yalmip problem
x=sdpvar(2,1); % real variable
constraints=[Pcon.A*x<=Pcon.b]; % force x to stay inside the polyhedron
    described by the halfspace representation
objective=c'*x; % define the linear cost
options=[];
sol = optimize(constraints,objective,options); % solve the optimization
    problem
xsol=value(x); %recover the value of the optimum
```

```
%plot the polyhedron, the optimum and the line c'*x=c'*xsol
figure
hold on
plot(Pcon,opt{:}) % draw the constraint set
a=2*axis; % get the bounds of the figure to draw correctly the hyperplane
axis(a)
plot(a([1 2]),(c'*xsol-c(1).*a([1 2]))./c(2)) % draw the segment defined by
     c'*x=c'*xsol on the interval a(1)--a(2)
scatter(xsol(1),xsol(2),'filled','r') % draw the constrained optimum
title('linear cost constrained optimization problem')
%% repeat the procedure but this time for a quadratic cost (x'*H*x/2 + f'*x
    )

H=rand(2);
H=H*H'; % it has to be positive definite
f=rand(2,1)-0.5;

constraints=[Pcon.A*x<=Pcon.b];
objective=x'*H*x/2 + f'*x;
sol = optimize(constraints,objective,options);
xsol=double(x);

xunc=-inv(H)*f; % the unconstrained solution of a quadratic problem
figure
hold on
scatter(xsol(1),xsol(2),'filled','r') % draw the constrained optimum
scatter(xunc(1),xunc(2),'filled','r') % draw the unconstrained optimum
plot(Pcon,opt{:}) % draw the constraint set
hold on

a=2*axis; % get the bounds of the figure to draw correctly the hyperplane
axis(a)

scaling=(xsol+inv(H)*f)'*(H/2)*(xsol+inv(H)*f);
plot(ellipsoid(xunc,inv(H/2)*scaling))
title('quadratic cost constrained optimization problem')
```

Listing 1: Matlab script for the intro material

A.2. Numerical data used in the exercises

```
% discrete model in R2 with real eigenvalues

% x^+=Ax+B*d with |d|<= delta
A=[0 1; -2 -3];
B=eye(2);
% the matrices are for the continuous case, we discretize with Te
Te=0.1;
A=eye(2)+Te*A;
B=Te*B;
delta=[1 1]';
```

Listing 2: Discrete dynamics in $\mathbb{R}^2$ with real eigenvalues

```
% discrete model in R2 with complex conjugated eigenvalues

% x^+=Ax+B*d with |d|<= delta
A=[0 1; -2 -2];
B=eye(2);
% the matrices are for the continuous case, we discretize with Te
Te=0.1;
A=eye(2)+Te*A;
B=Te*B;
delta=[1 1]';
```

Listing 3: Discrete dynamics in $\mathbb{R}^2$ with complex conjugated eigenvalues

```
% discrete model in R3 with real eigenvalues

% x^+=Ax+B*d with |d|<= delta
A=[   0.9000          0          0;...
    -4.3600     0.2800     1.5600;...
     2.2800     0.2600     0.0200];
B=eye(3);
delta=[1 1 1]';
```

Listing 4: Discrete dynamics in $\mathbb{R}^3$ with real eigenvalues

```
% discrete model in R3 with real eigenvalues

% x^+=Ax+B*d with |d|<= delta
A=[0 0.6; 0.3 -0.5];
B=[0;1];
delta=1;
```

Listing 5: Discrete dynamics in $\mathbb{R}^2$ with real eigenvalues

A.3. Solutions to the exercises

```
% matlab session 1 - exercise 1
clear all
close all
clc

opt={'Color', 'b', 'Alpha', 0.4}; %prettify the output
opt2={'Color', 'r', 'Alpha', 0.4};

%% compute ultimate bounds sets for the case of real eigenvalues
% dynamics given in data_1.m and data_3.m have both real eigenvalues and
% can be used here

run data_1
```

```
[n,m]=size(B);
[V,D]=jordan(A);

% compute the ultimate bounds sets
epsilon=0;
Pub=Polyhedron([inv(V); -inv(V)],repmat(inv(eye(n)-abs(D))*abs(inv(V)*B)*
    delta+epsilon,2,1));

% plot the UB set and its iteration in one step under the dynamics
Delta=Polyhedron([eye(m); -eye(m)],[delta;delta]);
plot(Pub,opt{:})
hold on
plot(A*Pub+B*Delta,opt2{:})
title('UB set - 1st example - real eigenvalues')


run data_3

[n,m]=size(B);
[V,D]=jordan(A);

% compute the ultimate bounds sets
epsilon=0;
Pub=Polyhedron([inv(V); -inv(V)],repmat(inv(eye(n)-abs(D))*abs(inv(V)*B)*
    delta+epsilon,2,1));

% plot the UB set and its iteration in one step under the dynamics
figure

Delta=Polyhedron([eye(m); -eye(m)],[delta;delta]);
plot(Pub,opt{:})
hold on
plot(A*Pub+B*Delta,opt2{:})
title('UB set - 3rd example - real eigenvalues')

% check numerically the positive invaraince via set inclusion
if A*Pub+B*Delta<=Pub
    sprintf('The set is robust positive invariant')
end
```

Listing 6: Matlab code for Exercise 3.1