Ellipsoidal Toolbox

Alex A. Kurzhanskiy
Pravin Varaiya

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-46
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-46.html

May 6, 2006

Acknowledgement

The authors thank Alexander B. Kurzhanski, Manfred Morari, Johan LÄofberg, Michal
Kvasnica and Goran Frehse for their support of this work by useful advice and encouragement.

ELLIPSOIDAL TOOLBOX[1]
Technical Report

Alex A. Kurzhanskiy and Pravin Varaiya

2006

[1]Research supported by NSF Grant CCR-00225610.

Contents

Chapter 1

Introduction

Research on dynamical and hybrid systems has produced several methods for verification and controller synthesis. A common step in these methods is the reachability analysis of the system. Reachability analysis is concerned with the computation of the reach set in a way that can effectively meet requests like the following:

1. For a given target set and time, determine whether the reach set and the target set have nonempty intersection.
2. For specified reachable state and time, find a feasible initial condition and control that steers the system from this initial condition to the given reachable state in given time.
3. Graphically display the projection of the reach set onto any specified two- or three-dimensional subspace.

Except for very specific classes of systems, exact computation of reach sets is not possible, and approximation techniques are needed. For controlled linear systems with convex bounds on the control and initial conditions, the efficiency and accuracy of these techniques depend on how they represent convex sets and how well they perform the operations of unions, intersections, geometric (Minkowski) sums and differences of convex sets. Two basic objects are used as convex approximations: polytopes of various types, including general polytopes, zonotopes, parallelotopes, rectangular polytopes; and ellipsoids.

Reachability analysis for general polytopes is implemented in the Multi Parametric Toolbox (MPT) for Matlab [12, 13]. The reach set at every time step is computed as the geometric sum of two polytopes. The procedure consists in finding the vertices of the resulting polytope and calculating their convex hull. MPT's convex hull algorithm is based on the Double Description method [28] and implemented in the CDD/CDD+ package [29]. Its complexity is V^n, where V is the number of vertices and n is the state space dimension. Hence the use of MPT is practicable for low dimensional systems. But even in low dimensional systems the number of vertices in the reach set polytope can grow very large with the number of time steps. For example, consider the system,

$$x_{k+1} = Ax_k + u_k,$$

with $A = \begin{bmatrix} \cos 1 & -\sin 1 \\ \sin 1 & \cos 1 \end{bmatrix}$, $u_k \in \{u \in \mathbf{R}^2 \mid \|u\|_\infty \leq 1\}$, and $x_0 \in \{x \in \mathbf{R}^2 \mid \|x\|_\infty \leq 1\}$. Starting with a rectangular initial set, the number of vertices of the reach set polytope is $4k + 4$ at the kth step.

In d/dt [23], the reach set is approximated by unions of rectangular polytopes [22]. The algorithm

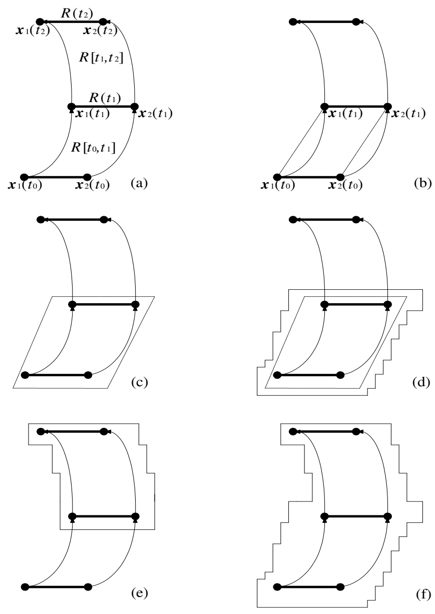

Figure 1.1: Reach set approximation by union of rectangles. Source: adapted from [22].

works as follows. First, given the set of initial conditions defined as a polytope, the evolution in time of the polytope's extreme points is computed (figure 1.1(a)). $R(t_1)$ in figure 1.1(a) is the reach set of the system at time t_1, and $R[t_0, t_1]$ is the set of all points that can be reached during $[t_0, t_1]$. Second, the algorithm computes the convex hull of vertices of both, the initial polytope and $R(t_1)$ (figure 1.1(b)). The resulting polytope is then bloated to include all the reachable states in $[t_0, t_1]$ (figure 1.1(c)). Finally, this overapproximating polytope is in its turn overapproximated by the union of rectangles (figure 1.1(d)). The same procedure is repeated for the next time interval $[t_1, t_2]$, and the union of both rectangular approximations is taken (figure 1.1(e,f)), and so on. Rectangular polytopes are easy to represent and the number of facets grows linearly with dimension, but a large number of rectangles must be used to assure the approximation is not overly conservative. Besides, the important part of this method is again the convex hull calculation whose implementation relies on the same CDD/CDD+ library. This limits the dimension of the system and time interval for which it is feasible to calculate the reach set.

Polytopes can give arbitrarily close approximations to any convex set, but the number of vertices can grow prohibitively large and, as shown in [30], the computation of a polytope by its convex hull becomes intractable for large number of vertices in high dimensions.

The method of zonotopes for approximation of reach sets [24, 25, 26] uses a special class of polytopes (see [27]) of the form,

$$Z = \{x \in \mathbf{R}^n \mid x = c + \sum_{i=1}^{p} \alpha_i g_i, \ \ -1 \leq \alpha_i \leq 1\},$$

wherein c and $g_1, ..., g_p$ are vectors in $\mathbf{R}^n$. Thus, a zonotope Z is represented by its center c and 'generator' vectors $g_1, ..., g_p$. The value p/n is called the order of the zonotope. The main benefit of zonotopes over general polytopes is that a symmetric polytope can be represented more compactly than a general polytope. The geometric sum of two zonotopes is a zonotope:

$$Z(c_1, G_1) \oplus Z(c_2, G_2) = Z(c_1 + c_2, [G_1 \; G_2]),$$

wherein G_1 and G_2 are matrices whose columns are generator vectors, and $[G_1 \; G_2]$ is their concatenation. Thus, in the reach set computation, the order of the zonotope increases by p/n with every time step. This difficulty can be averted by limiting the number of generator vectors, and overapproximating zonotopes whose number of generator vectors exceeds the limit by lower order zonotopes. The benefits of the compact zonotype representation, however, appear to diminish because in order to plot them or check if they intersect with given objects and compute those intersections, these operations are performed after converting zonotopes to polytopes.

CheckMate [21] is a Matlab toolbox that can evaluate specifications for trajectories starting from the set of initial (continuous) states corresponding to the parameter values at the vertices of the parameter set. This provides preliminary insight into whether the specifications will be true for all parameter values. The method of oriented rectangluar polytopes for external approximation of reach sets is introduced in [20]. The basic idea is to construct an oriented rectangular hull of the reach set for every time step, whose orientation is determined by the singular value decomposition of the sample covariance matrix for the states reachable from the vertices of the initial polytope. The limitation of CheckMate and the method of oriented rectangles is that only autonomous (i.e. uncontrolled) systems, or systems with fixed input are allowed, and only an external approximation of the reach set is provided.

All the methods described so far employ the notion of time step, and calculate the reach set or its approximation at each time step. This approach can be used only with discrete-time systems. By contrast, the analytic methods which we are about to discuss, provide a formula or differential equation describing the (continuous) time evolution of the reach set or its approximation.

The level set method [18, 19] deals with general nonlinear controlled systems and gives exact representation of their reach sets, but requires solving the HJB equation and finding the set of states that belong to sub-zero level set of the value function. The method [19] is impractical for systems of dimension higher than three.

Requiem [32] is a Mathematica notebook which, given a linear system, the set of initial conditions and control bounds, symbolically computes the exact reach set, using the experimental quantifier elimination package. Quantifier elimination is the removal of all quantifiers (the universal quantifier $\forall$ and the existential quantifier $\exists$) from a quantified system. Each quantified formula is substituted with quantifier-free expression with operations $+$, $\times$, $=$ and $<$. For example, consider the discrete-time system

$$x_{k+1} = Ax_k + Bu_k$$

with $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. For initial conditions $x_0 \in \{x \in \mathbf{R}^2 \mid \|x\|_\infty \leq 1\}$ and controls $u_k \in \{u \in \mathbf{R} \mid -1 \leq u \leq 1\}$, the reach set for $k \geq 0$ is given by the quantified formula

$$\{x \in \mathbf{R}^2 \mid \exists x_0, \;\; \exists k \geq 0, \;\; \exists u_i, \; 0 \leq i \leq k : \;\; x = A^k x_0 + \sum_{i=0}^{k-1} A^{k-i-1} B u_i\},$$

which is equivalent to the quantifier-free expression

$$-1 \leq [1 \;\; 0]x \leq 1 \;\wedge\; -1 \leq [0 \;\; 1]x \leq 1.$$

It is proved in [31] that for continuous-time systems, $\dot{x}(t) = Ax(t) + Bu(t)$, if A is constant and nilpotent or is diagonalizable with rational real or purely imaginary eigenvalues, and with suitable restrictions on the control and initial conditions, the quantifier elimination package returns a quantifier free formula describing the reach set. Quantifier elimination has limited applicability.

The reach set approximation via parallelotopes [35] employs the idea of parametrization described in [3] for ellipsoids. The reach set is represented as the intersection of tight external, and the union of tight internal, parallelotopes. The evolution equations for the centers and orientation matrices of both external and internal parallelotopes are provided. This method also finds controls that can drive the system to the boundary points of the reach set, similarly to [7] and [3]. It works for general linear systems. The computation to solve the evolution equation for tight approximating parallelotopes, however, is more involved than that for ellipsoids, and for discrete-time systems this method does not deal with singular state transition matrices.

Ellipsoidal Toolbox (ET) implements in MATLAB the ellipsoidal calculus [2] and its application to the reachability analysis of continuous-time [3], discrete-time [6], possibly time-varying linear systems, and linear systems with disturbances [5], for which ET calculates both open-loop and close-loop reach sets. The ellipsoidal calculus provides the following benefits:

- The complexity of the ellipsoidal representation is quadratic in the dimension of the state space, and linear in the number of time steps.
- It is possible to exactly represent the reach set of linear system through both external and internal ellipsoids.
- It is possible to single out individual external and internal approximating ellipsoids that are optimal to some given criterion (e.g. trace, volume, diameter), or combination of such criteria.
- We obtain simple analytical expressions for the control that steers the state to a desired target.

The report is organized as follows.
Chapter 2 describes the operations of the ellipsoidal calculus: affine transformation, geometric sum, geometric difference, intersections with hyperplane, ellipsoid, halfspace and polytope.
Chapter 3 presents the reachability problem and ellipsoidal methods for the reach set approximation.
Chapter 4 contains *Ellipsoidal Toolbox* installation and quick start instructions, and lists the software packages used by the toolbox.
Chapter 5 describes the implementation of methods from chapters 2 and 3 and visualization routines.
Chapter 6 describes structures and objects implemented and used in the toolbox.
Chapter 7 gives examples of how to use the toolbox.
Chapter 8 collects some conclusions and plans for future toolbox development.
The functions provided by the toolbox together with their descriptions are listed in appendix A.

Chapter 2

Ellipsoidal Calculus

2.1 Basic Notions

We start with basic definitions.

Definition 2.1.1 *Ellipsoid* $\mathcal{E}(q,Q)$ *in* $\mathbf{R}^n$ *with center* q *and shape matrix* Q *is the set*

$$\mathcal{E}(q,Q) = \{x \in \mathbf{R}^n \mid \langle (x-q), Q^{-1}(x-q)\rangle \leq 1\}, \tag{2.1}$$

wherein Q *is positive definite* ($Q = Q^T$ *and* $\langle x, Qx\rangle > 0$ *for all nonzero* $x \in \mathbf{R}^n$*).*

Here $\langle\cdot,\cdot\rangle$ denotes inner product.

Definition 2.1.2 *The support function of a set* $\mathcal{X} \subseteq \mathbf{R}^n$ *is*

$$\rho(l \mid \mathcal{X}) = \sup_{x\in\mathcal{X}} \langle l, x\rangle.$$

In particular, the support function of the ellipsoid (2.1) is

$$\rho(l \mid \mathcal{E}(q,Q)) = \langle l, q\rangle + \langle l, Ql\rangle^{1/2}. \tag{2.2}$$

Although in (2.1) Q is assumed to be positive definite, in practice we may deal with situations when Q is singular, that is, with degenerate ellipsoids flat in those directions for which the corresponding eigenvalues are zero. Therefore, it is useful to give an alternative definition of an ellipsoid using the expression (2.2).

Definition 2.1.3 *Ellipsoid* $\mathcal{E}(q,Q)$ *in* $\mathbf{R}^n$ *with center* q *and shape matrix* Q *is the set*

$$\mathcal{E}(q,Q) = \{x \in \mathbf{R}^n \mid \langle l, x\rangle \leq \langle l, q\rangle + \langle l, Ql\rangle^{1/2} \text{ for all } l \in \mathbf{R}^n\}, \tag{2.3}$$

wherein matrix Q *is positive semidefinite* ($Q = Q^T$ *and* $\langle x, Qx\rangle \geq 0$ *for all* $x \in \mathbf{R}^n$*).*

The distance from $\mathcal{E}(q,Q)$ to the fixed point a is

$$\mathbf{dist}(\mathcal{E}(q,Q),a) = \max_{\langle l,l\rangle=1} (\langle l,a\rangle - \rho(l \mid \mathcal{E}(q,Q))) = \max_{\langle l,l\rangle=1} \left(\langle l,a\rangle - \langle l,q\rangle - \langle l,Ql\rangle^{1/2}\right). \tag{2.4}$$

If $\mathbf{dist}(\mathcal{E}(q,Q),a) > 0$, a lies outside $\mathcal{E}(q,Q)$; if $\mathbf{dist}(\mathcal{E}(q,Q),a) = 0$, a is a boundary point of $\mathcal{E}(q,Q)$; if $\mathbf{dist}(\mathcal{E}(q,Q),a) < 0$, a is an internal point of $\mathcal{E}(q,Q)$.

Given two ellipsoids, $\mathcal{E}(q_1,Q_1)$ and $\mathcal{E}(q_2,Q_2)$, the distance between them is

$$\begin{aligned}\mathbf{dist}(\mathcal{E}(q_1,Q_1),\mathcal{E}(q_2,Q_2)) &= \max_{\langle l,l\rangle=1} (-\rho(-l \mid \mathcal{E}(q_1,Q_1)) - \rho(l \mid \mathcal{E}(q_2,Q_2))) &(2.5)\\ &= \max_{\langle l,l\rangle=1} \left(\langle l,q_1\rangle - \langle l,Q_1l\rangle^{1/2} - \langle l,q_2\rangle - \langle l,Q_2l\rangle^{1/2}\right). &(2.6)\end{aligned}$$

If $\mathbf{dist}(\mathcal{E}(q_1,Q_1),\mathcal{E}(q_2,Q_2)) > 0$, the ellipsoids have no common points; if $\mathbf{dist}(\mathcal{E}(q_1,Q_1),\mathcal{E}(q_2,Q_2)) = 0$, the ellipsoids have one common point - they touch; if $\mathbf{dist}(\mathcal{E}(q_1,Q_1),\mathcal{E}(q_2,Q_2)) < 0$, the ellipsoids intersect.

Checking if k nondegenerate ellipsoids $\mathcal{E}(q_1,Q_1),\cdots,\mathcal{E}(q_k,Q_k)$ have nonempty intersection, can be cast as a quadratically constrained quadratic programming (QCQP) problem:

$$\min 0$$

subject to:

$$\langle (x-q_i), Q_i^{-1}(x-q_i)\rangle - 1 \leq 0, \quad i = 1,\cdots,k.$$

If this problem is feasible, the intersection is nonempty.

Definition 2.1.4 *Given compact convex set $\mathcal{X} \subseteq \mathbf{R}^n$, its polar set, denoted $\mathcal{X}^\circ$, is*

$$\mathcal{X}^\circ = \{x \in \mathbf{R}^n \mid \langle x,y\rangle \leq 1, \; y \in \mathcal{X}\},$$

or, equivalently,

$$\mathcal{X}^\circ = \{l \in \mathbf{R}^n \mid \rho(l \mid \mathcal{X}) \leq 1\}.$$

The properties of the polar set are

- If $\mathcal{X}$ contains the origin, $(\mathcal{X}^\circ)^\circ = \mathcal{X}$;
- If $\mathcal{X}_1 \subseteq \mathcal{X}_2$, $\mathcal{X}_2^\circ \subseteq \mathcal{X}_1^\circ$;
- For any nonsingular matrix $A \in \mathbf{R}^{n\times n}$, $(A\mathcal{X})^\circ = (A^T)^{-1}\mathcal{X}^\circ$.

If a nondegenerate ellipsoid $\mathcal{E}(q,Q)$ contains the origin, its polar set is also an ellipsoid:

$$\begin{aligned}\mathcal{E}^\circ(q,Q) &= \{l \in \mathbf{R}^n \mid \langle l,q\rangle + \langle l,Ql\rangle^{1/2} \leq 1\}\\ &= \{l \in \mathbf{R}^n \mid \langle l,(Q-qq^T)^{-1}l\rangle + 2\langle l,q\rangle \leq 1\}\\ &= \{l \in \mathbf{R}^n \mid \langle (l+(Q-qq^T)^{-1}q),(Q-qq^T)(l+(Q-qq^T)^{-1}q)\rangle \leq 1 + \langle q,(Q-qq^T)^{-1}q\rangle\}.\end{aligned}$$

The special case is

$$\mathcal{E}^\circ(0,Q) = \mathcal{E}(0,Q^{-1}).$$

Definition 2.1.5 *Given k compact sets $\mathcal{X}_1, \cdots, \mathcal{X}_k \subseteq \mathbf{R}^n$, their geometric (Minkowski) sum is*

$$\mathcal{X}_1 \oplus \cdots \oplus \mathcal{X}_k = \bigcup_{x_1 \in \mathcal{X}_1} \cdots \bigcup_{x_k \in \mathcal{X}_k} \{x_1 + \cdots + x_k\}. \tag{2.7}$$

Definition 2.1.6 *Given two compact sets $\mathcal{X}_1, \mathcal{X}_2 \subseteq \mathbf{R}^n$, their geometric (Minkowski) difference is*

$$\mathcal{X}_1 \dot{-} \mathcal{X}_2 = \{x \in \mathbf{R}^n \mid x + \mathcal{X}_2 \subseteq \mathcal{X}_1\}. \tag{2.8}$$

Ellipsoidal calculus concerns the following set of operations:

- affine transformation of ellipsoid;
- geometric sum of finite number of ellipsoids;
- geometric difference of two ellipsoids;
- intersection of finite number of ellipsoids.

These operations occur in reachability calculation and verification of piecewise affine dynamical systems. The result of all of these operations, except for the affine transformation, is *not* generally an ellipsoid but some convex set, for which we can compute external and internal ellipsoidal approximations.

Additional operations implemented in the *Ellipsoidal Toolbox* include external and internal approximations of intersections of ellipsoids with hyperplanes, halfspaces and polytopes.

Definition 2.1.7 *Hyperplane $H(c, \gamma)$ in $\mathbf{R}^n$ is the set*

$$H = \{x \in \mathbf{R}^n \mid \langle c, x \rangle = \gamma\} \tag{2.9}$$

with $c \in \mathbf{R}^n$ and $\gamma \in \mathbf{R}$ fixed.

The distance from ellipsoid $\mathcal{E}(q, Q)$ to hyperplane $H(c, \gamma)$ is

$$\mathbf{dist}(\mathcal{E}(q, Q), H(c, \gamma)) = \frac{|\gamma - \langle c, q \rangle| - \langle c, Qc \rangle^{1/2}}{\langle c, c \rangle^{1/2}}. \tag{2.10}$$

If $\mathbf{dist}(\mathcal{E}(q, Q), H(c, \gamma)) > 0$, the ellipsoid and the hyperplane do not intersect; if $\mathbf{dist}(\mathcal{E}(q, Q), H(c, \gamma)) = 0$, the hyperplane is a supporting hyperplane for the ellipsoid; if $\mathbf{dist}(\mathcal{E}(q, Q), H(c, \gamma)) < 0$, the ellipsoid intersects the hyperplane. The intersection of an ellipsoid with a hyperplane is always an ellipsoid and can be computed directly.

Checking if the intersection of k nondegenerate ellipsoids $E(q_1, Q_1), \cdots, \mathcal{E}(q_k, Q_k)$ intersects hyperplane $H(c, \gamma)$, is equivalent to the feasibility check of the QCQP problem:

$$\min 0$$

subject to:

$$\begin{gathered} \langle (x - q_i), Q_i^{-1}(x - q_i) \rangle - 1 \leq 0, \qquad i = 1, \cdots, k, \\ \langle c, x \rangle - \gamma = 0. \end{gathered}$$

A hyperplane defines two (closed) *halfspaces*:

$$\mathbf{S}_1 = \{x \in \mathbf{R}^n \mid \langle c, x\rangle \leq \gamma\} \tag{2.11}$$

and

$$\mathbf{S}_2 = \{x \in \mathbf{R}^n \mid \langle c, x\rangle \geq \gamma\}. \tag{2.12}$$

To avoid confusion, however, we shall further assume that a hyperplane $H(c,\gamma)$ specifies the halfspace in the sense (2.11). In order to refer to the other halfspace, the same hyperplane should be defined as $H(-c,-\gamma)$.

The idea behind the calculation of intersection of an ellipsoid with a halfspace is to treat the halfspace as an unbounded ellipsoid, that is, as the ellipsoid with the shape matrix all but one of whose eigenvalues are ∞.

Definition 2.1.8 *Polytope $P(C,g)$ is the intersection of a finite number of closed halfspaces:*

$$P = \{x \in \mathbf{R}^n \mid Cx \leq g\},$$

wherein $C = [c_1 \ \cdots \ c_m]^T \in \mathbf{R}^{m\times n}$ and $g = [\gamma_1 \ \cdots \ \gamma_m]^T \in \mathbf{R}^m$.

The distance from ellipsoid $\mathcal{E}(q,Q)$ to the polytope $P(C,g)$ is

$$\mathbf{dist}(\mathcal{E}(q,Q), P(C,g)) = \min_{y\in P(C,g)} \mathbf{dist}(\mathcal{E}(q,Q), y), \tag{2.13}$$

where $\mathbf{dist}(\mathcal{E}(q,Q), y)$ comes from (2.4). If $\mathbf{dist}(\mathcal{E}(q,Q), P(C,g)) > 0$, the ellipsoid and the polytope do not intersect; if $\mathbf{dist}(\mathcal{E}(q,Q), P(C,g)) = 0$, the ellipsoid touches the polytope; if $\mathbf{dist}(\mathcal{E}(q,Q), P(C,g)) < 0$, the ellipsoid intersects the polytope.

Checking if the intersection of k nondegenerate ellipsoids $E(q_1,Q_1), \cdots, \mathcal{E}(q_k,Q_k)$ intersects polytope $P(C,g)$ is equivalent to the feasibility check of the QCQP problem:

$$\min 0$$

subject to:

$$\begin{aligned} \langle (x-q_i), Q_i^{-1}(x-q_i)\rangle - 1 \leq 0, &\qquad i = 1,\cdots,k, \\ \langle c_j, x\rangle - \gamma_j \leq 0, &\qquad j = 1,\cdots,m. \end{aligned}$$

2.2 Operations with Ellipsoids

2.2.1 Affine Transformation

The simplest operation with ellipsoids is an affine transformation. Let ellipsoid $\mathcal{E}(q,Q) \subseteq \mathbf{R}^n$, matrix $A \in \mathbf{R}^{m\times n}$ and vector $b \in \mathbf{R}^m$. Then

$$A\mathcal{E}(q,Q) + b = \mathcal{E}(Aq + b, AQA^T). \tag{2.14}$$

Thus, ellipsoids are preserved under affine transformation. If the rows of A are linearly independent (which implies $m \leq n$), and $b = 0$, the affine transformation is called *projection*.

2.2.2 Geometric Sum

Consider the geometric sum (2.7) in which $\mathcal{X}_1,\cdots,\mathcal{X}_k$ are nondegenerate ellipsoids $\mathcal{E}(q_1,Q_1),\cdots,$ $\mathcal{E}(q_k,Q_k)\subseteq\mathbf{R}^n$. The resulting set is not generally an ellipsoid. However, it can be tightly approximated by the parametrized families of external and internal ellipsoids.

Let parameter l be some nonzero vector in $\mathbf{R}^n$. Then the external approximation $\mathcal{E}(q,Q_l^+)$ and the internal approximation $\mathcal{E}(q,Q_l^-)$ of the sum $\mathcal{E}(q_1,Q_1)\oplus\cdots\oplus\mathcal{E}(q_k,Q_k)$ are *tight* along direction l, i.e.,

$$\mathcal{E}(q,Q_l^-)\subseteq\mathcal{E}(q_1,Q_1)\oplus\cdots\oplus\mathcal{E}(q_k,Q_k)\subseteq\mathcal{E}(q,Q_l^+)$$

and

$$\rho(\pm l\mid\mathcal{E}(q,Q_l^-))=\rho(\pm l\mid\mathcal{E}(q_1,Q_1)\oplus\cdots\oplus\mathcal{E}(q_k,Q_k))=\rho(\pm l\mid\mathcal{E}(q,Q_l^+)).$$

Here the center q is

$$q=q_1+\cdots+q_k, \tag{2.15}$$

the shape matrix of the external ellipsoid Q_l^+ is

$$Q_l^+=\left(\langle l,Q_1l\rangle^{1/2}+\cdots+\langle l,Q_kl\rangle^{1/2}\right)\left(\frac{1}{\langle l,Q_1l\rangle^{1/2}}Q_1+\cdots+\frac{1}{\langle l,Q_kl\rangle^{1/2}}Q_k\right), \tag{2.16}$$

and the shape matrix of the internal ellipsoid Q_l^- is

$$Q_l^-=\left(Q_1^{1/2}+S_2Q_2^{1/2}+\cdots+S_kQ_k^{1/2}\right)^T\left(Q_1^{1/2}+S_2Q_2^{1/2}+\cdots+S_kQ_k^{1/2}\right), \tag{2.17}$$

with matrices S_i, $i=2,\cdots,k$, being orthogonal ($S_iS_i^T=I$) and such that vectors $Q_1^{1/2}l,S_2Q_2^{1/2}l,\cdots,S_kQ_k^{1/2}l$ are parallel.

Varying vector l we get exact external and internal approximations,

$$\bigcup_{\langle l,l\rangle=1}\mathcal{E}(q,Q_l^-)=\mathcal{E}(q_1,Q_1)\oplus\cdots\oplus\mathcal{E}(q_k,Q_k)=\bigcap_{\langle l,l\rangle=1}\mathcal{E}(q,Q_l^+).$$

For proofs of formulas given in this section, see [2], [3].

One last comment is about how to find orthogonal matrices $S_2,\cdots,S_k$ that align vectors $Q_2^{1/2}l,\cdots,Q_k^{1/2}l$ with $Q_1^{1/2}l$. Let v and w be some unit vectors in $\mathbf{R}^n$. We have to find matrix S such that $Sv=w$. For that, we perform singular value decomposition (SVD) of vectors v and w:

$$v=U_v\Sigma_vV_v^T,\qquad w=U_w\Sigma_wV_w^T. \tag{2.18}$$

Notice that V_v and V_w are ±1 scalars. The matrix S is now easily determined:

$$SU_vV_v=U_wV_w\quad\Rightarrow\quad S=U_wV_wV_vU_v^T. \tag{2.19}$$

2.2.3 Geometric Difference

Consider the geometric difference (2.8) in whic the sets $\mathcal{X}_1$ and $\mathcal{X}_2$ are nondegenerate ellipsoids $\mathcal{E}(q_1,Q_1)$ and $\mathcal{E}(q_2,Q_2)$. We say that ellipsoid $\mathcal{E}(q_1,Q_1)$ is *bigger* than ellipsoid $\mathcal{E}(q_2,Q_2)$ if

$$\mathcal{E}(0,Q_2)\subseteq\mathcal{E}(0,Q_1).$$

If this condition is not fulfilled, the geometric difference $\mathcal{E}(q_1, Q_1)\dot{-}\mathcal{E}(q_2, Q_2)$ is an empty set:

$$\mathcal{E}(0, Q_2) \not\subseteq \mathcal{E}(0, Q_1) \quad \Rightarrow \quad \mathcal{E}(q_1, Q_1)\dot{-}\mathcal{E}(q_2, Q_2) = \emptyset.$$

If $\mathcal{E}(q_1, Q_1)$ is bigger than $\mathcal{E}(q_2, Q_2)$ and $\mathcal{E}(q_2, Q_2)$ is bigger than $\mathcal{E}(q_1, Q_1)$, in other words, if $Q_1 = Q_2$,

$$\mathcal{E}(q_1, Q_1)\dot{-}\mathcal{E}(q_2, Q_2) = \{q_1 - q_2\} \quad \text{and} \quad \mathcal{E}(q_2, Q_2)\dot{-}\mathcal{E}(q_1, Q_1) = \{q_2 - q_1\}.$$

To check if ellipsoid $\mathcal{E}(q_1, Q_1)$ is bigger than ellipsoid $\mathcal{E}(q_2, Q_2)$, we perform simultaneous diagonalization of matrices Q_1 and Q_2, that is, we find matrix T such that

$$TQ_1T^T = I \quad \text{and} \quad TQ_2T^T = D,$$

where D is some diagonal matrix. Simultaneous diagonalization of Q_1 and Q_2 is possible because both are symmetric positive definite (see [39]). To find such matrix T, we first do the SVD of Q_1:

$$Q_1 = U_1\Sigma_1V_1^T. \tag{2.20}$$

Then the SVD of matrix $\Sigma_1^{-1/2}U_1^TQ_2U_1\Sigma_1^{-1/2}$:

$$\Sigma_1^{-1/2}U_1^TQ_2U_1\Sigma_1^{-1/2} = U_2\Sigma_2V_2^T. \tag{2.21}$$

Now, T is defined as

$$T = U_2^T\Sigma_1^{-1/2}U_1^T. \tag{2.22}$$

If the biggest diagonal element (eigenvalue) of matrix $D = TQ_2T^T$ is less than or equal to 1, $\mathcal{E}(0, Q_2) \subseteq \mathcal{E}(0, Q_1)$.

Once it is established that ellipsoid $\mathcal{E}(q_1, Q_1)$ is bigger than ellipsoid $\mathcal{E}(q_2, Q_2)$, we know that their geometric difference $\mathcal{E}(q_1, Q_1)\dot{-}\mathcal{E}(q_2, Q_2)$ is a nonempty convex compact set. Although it is not generally an ellipsoid, we can find tight external and internal approximations of this set parametrized by vector $l \in \mathbf{R}^n$. Unlike geometric sum, however, ellipsoidal approximations for the geometric difference do not exist for every direction l. Vectors for which the approximations do not exist are called *bad directions*.

Given two ellipsoids $\mathcal{E}(q_1, Q_1)$ and $\mathcal{E}(q_2, Q_2)$ with $\mathcal{E}(0, Q_2) \subseteq \mathcal{E}(0, Q_1)$, l is a bad direction if

$$\frac{\langle l, Q_1l\rangle^{1/2}}{\langle l, Q_2l\rangle^{1/2}} > r,$$

in which r is a minimal root of the equation

$$\mathbf{det}(Q_1 - rQ_2) = 0.$$

To find r, compute matrix T by (2.20-2.22) and define

$$r = \frac{1}{\max(\mathbf{diag}(TQ_2T^T))}.$$

If l is *not* a bad direction, we can find tight external and internal ellipsoidal approximations $\mathcal{E}(q, Q_l^+)$ and $\mathcal{E}(q, Q_l^-)$ such that

$$\mathcal{E}(q, Q_l^-) \subseteq \mathcal{E}(q_1, Q_1)\dot{-}\mathcal{E}(q_2, Q_2) \subseteq \mathcal{E}(q, Q_l^+)$$

and

$$\rho(\pm l \mid \mathcal{E}(q, Q_l^-)) = \rho(\pm l \mid \mathcal{E}(q_1, Q_1)\dot{-}\mathcal{E}(q_2, Q_2)) = \rho(\pm l \mid \mathcal{E}(q, Q_l^+)).$$

The center q is

$$q = q_1 - q_2; \tag{2.23}$$

the shape matrix of the internal ellipsoid Q_l^- is

$$Q_l^- = \left(1 - \frac{\langle l, Q_1 l\rangle^{1/2}}{\langle l, Q_2 l\rangle^{1/2}}\right) Q_1 + \left(1 - \frac{\langle l, Q_2 l\rangle^{1/2}}{\langle l, Q_1 l\rangle^{1/2}}\right) Q_2; \tag{2.24}$$

and the shape matrix of the external ellipsoid Q_l^+ is

$$Q_l^+ = \left(Q_1^{1/2} + SQ_2^{1/2}\right)^T \left(Q_1^{1/2} + SQ_2^{1/2}\right). \tag{2.25}$$

Here S is an orthogonal matrix such that vectors $Q_1^{1/2}l$ and $SQ_2^{1/2}l$ are parallel. S is found from (2.18-2.19), with $v = Q_2^{1/2}l$ and $w = Q_1^{1/2}l$.

Running l over all unit directions that are not bad, we get

$$\bigcup_{\langle l,l\rangle=1} \mathcal{E}(q, Q_l^-) = \mathcal{E}(q_1, Q_1) \dot{-} \mathcal{E}(q_2, Q_2) = \bigcap_{\langle l,l\rangle=1} \mathcal{E}(q, Q_l^+).$$

For proofs of formulas given in this section, see [2].

2.2.4 Intersection of Ellipsoid and Hyperplane

Let nondegenerate ellipsoid $\mathcal{E}(q, Q)$ and hyperplane $H(c, \gamma)$ be such that $\mathbf{dist}(\mathcal{E}(q, Q), H(c, \gamma)) < 0$. In other words,

$$\mathcal{E}_H(w, W) = \mathcal{E}(q, Q) \cap H(c, \gamma) \neq \emptyset.$$

The intersection of ellipsoid with hyperplane, if nonempty, is always an ellipsoid. Here we show how to find it.

First of all, we transform the hyperplane $H(c, \gamma)$ into $H([1\ 0\ \cdots\ 0]^T, 0)$ by the affine transformation

$$y = Sx - \frac{\gamma}{\langle c, c\rangle^{1/2}} Sc,$$

where S is an orthogonal matrix found by (2.18-2.19) with $v = c$ and $w = [1\ 0\ \cdots\ 0]^T$. The ellipsoid in the new coordinates becomes $\mathcal{E}(q', Q')$ with

$$\begin{aligned} q' &= q - \frac{\gamma}{\langle c, c\rangle^{1/2}} Sc, \\ Q' &= SQS^T. \end{aligned}$$

Define matrix $M = Q'^{-1}$; m_{11} is its element in position $(1, 1)$, $\bar{m}$ is the first column of M without the first element, and $\bar{M}$ is the submatrix of M obtained by stripping M of its first row and first column:

$$M = \left[\begin{array}{c|c} m_{11} & \bar{m}^T \\ \hline \bar{m} & \bar{M} \end{array}\right].$$

The ellipsoid resulting from the intersection is $\mathcal{E}_H(w', W')$ with

$$\begin{aligned} w' &= q' + q'_1 \left[\begin{array}{c} -1 \\ \bar{M}^{-1}\bar{m} \end{array}\right], \\ W' &= \left(1 - q'^2_1(m_{11} - \langle \bar{m}, \bar{M}^{-1}\bar{m}\rangle)\right) \left[\begin{array}{c|c} 0 & \mathbf{0} \\ \hline \mathbf{0} & \bar{M}^{-1} \end{array}\right], \end{aligned}$$

in which q'_1 represents the first element of vector q'.

Finally, it remains to do the inverse transform of the coordinates to obtain ellipsoid $\mathcal{E}_H(w, W)$:

$$\begin{aligned} w &= S^T w' + \frac{\gamma}{\langle c, c\rangle^{1/2}} c, \\ W &= S^T W' S. \end{aligned}$$

2.2.5 Intersection of Ellipsoid and Ellipsoid

Given two nondegenerate ellipsoids $\mathcal{E}(q_1, Q_1)$ and $\mathcal{E}(q_2, Q_2)$, $\mathbf{dist}(\mathcal{E}(q_1, Q_1), \mathcal{E}(q_2, Q_2)) < 0$ implies that

$$\mathcal{E}(q_1, Q_1) \cap \mathcal{E}(q_2, Q_2) \neq \emptyset.$$

This intersection can be approximated by ellipsoids from the outside and from the inside. Trivially, both $\mathcal{E}(q_1, Q_1)$ and $\mathcal{E}(q_2, Q_2)$ are external approximations of this intersection. Here, however, we show how to find the external ellipsoidal approximation of minimal volume.

Define matrices

$$W_1 = Q_1^{-1}, \quad W_2 = Q_2^{-1}. \tag{2.26}$$

Minimal volume external ellipsoidal approximation $\mathcal{E}(q+, Q^+)$ of the intersection $\mathcal{E}(q_1, Q_1) \cap \mathcal{E}(q_2, Q_2)$ is determined from the set of equations:

$$\begin{aligned} Q^+ &= \alpha X^{-1} && (2.27) \\ X &= \pi W_1 + (1-\pi) W_2 && (2.28) \\ \alpha &= 1 - \pi(1-\pi)\langle (q_2 - q_1), W_2 X^{-1} W_1 (q_2 - q_1)\rangle && (2.29) \\ q^+ &= X^{-1}(\pi W_1 q_1 + (1-\pi) W_2 q_2) && (2.30) \\ 0 &= \alpha(\mathbf{det}(X))^2 \mathbf{trace}(X^{-1}(W_1 - W_2)) \\ &- n(\mathbf{det}(X))^2 \big(2\langle q^+, W_1 q_1 - W_2 q_2\rangle + \langle q^+, (W_2 - W_1) q^+\rangle \\ &\quad - \langle q_1, W_1 q_1\rangle + \langle q_2, W_2 q_2\rangle\big), && (2.31) \end{aligned}$$

with $0 \leq \pi \leq 1$. We substitute X, α, q^+ defined in (2.28-2.30) into (2.31) and get a polynomial of degree $2n-1$ with respect to π, which has only one root in the interval $[0, 1]$, π_0. Then, substituting $\pi = \pi_0$ into (2.27-2.30), we obtain q^+ and Q^+. Special cases are $\pi_0 = 1$, whence $\mathcal{E}(q^+, Q^+) = \mathcal{E}(q_1, Q_1)$, and $\pi_0 = 0$, whence $\mathcal{E}(q^+, Q^+) = \mathcal{E}(q_2, Q_2)$. These situations may occur if, for example, one ellipsoid is contained in the other:

$$\begin{aligned} \mathcal{E}(q_1, Q_1) \subseteq \mathcal{E}(q_2, Q_2) &\quad \Rightarrow \quad \pi_0 = 1, \\ \mathcal{E}(q_2, Q_2) \subseteq \mathcal{E}(q_1, Q_1) &\quad \Rightarrow \quad \pi_0 = 0. \end{aligned}$$

The proof that the system of equations (2.27-2.31) correctly defines the minimal volume external ellipsoidal approximationi of the intersection $\mathcal{E}(q_1, Q_1) \cap \mathcal{E}(q_2, Q_2)$ is given in [10].

To find the internal approximating ellipsoid $\mathcal{E}(q^-, Q^-) \subseteq \mathcal{E}(q_1, Q_1) \cap \mathcal{E}(q_2, Q_2)$, define

$$\beta_1 = \min_{\langle x, W_2 x\rangle = 1} \langle x, W_1 x\rangle, \tag{2.32}$$

$$\beta_2 = \min_{\langle x, W_1 x\rangle = 1} \langle x, W_2 x\rangle, \tag{2.33}$$

Notice that (2.32) and (2.33) are QCQP problems. Parameters β_1 and β_2 are invariant with respect to affine coordinate transformation and describe the position of ellipsoids $\mathcal{E}(q_1, Q_1)$, $\mathcal{E}(q_2, Q_2)$ with respect to each other:

$$\begin{aligned}
\beta_1 \geq 1,\ \beta_2 \geq 1 &\Rightarrow \mathbf{int}(\mathcal{E}(q_1, Q_1) \cap \mathcal{E}(q_2, Q_2)) = \emptyset, \\
\beta_1 \geq 1,\ \beta_2 \leq 1 &\Rightarrow \mathcal{E}(q_1, Q_1) \subseteq \mathcal{E}(q_2, Q_2), \\
\beta_1 \leq 1,\ \beta_2 \geq 1 &\Rightarrow \mathcal{E}(q_2, Q_2) \subseteq \mathcal{E}(q_1, Q_1), \\
\beta_1 < 1,\ \beta_2 < 1 &\Rightarrow \mathbf{int}(\mathcal{E}(q_1, Q_1) \cap \mathcal{E}(q_2, Q_2)) \neq \emptyset \\
& \quad\ \text{and } \mathcal{E}(q_1, Q_1) \not\subseteq \mathcal{E}(q_2, Q_2) \\
& \quad\ \text{and } \mathcal{E}(q_2, Q_2) \not\subseteq \mathcal{E}(q_1, Q_1).
\end{aligned}$$

Define parametrized family of internal ellipsoids $\mathcal{E}(q^-_{\theta_1\theta_2}, Q^-_{\theta_1\theta_2})$ with

$$q^-_{\theta_1\theta_2} = (\theta_1 W_1 + \theta_2 W_2)^{-1}(\theta_1 W_1 q_1 + \theta_2 W_2 q_2), \tag{2.34}$$

$$Q^-_{\theta_1\theta_2} = (1 - \theta_1\langle q_1, W_1 q_1\rangle - \theta_2\langle q_2, W_2 q_2\rangle + \langle q^-_{\theta_1\theta_2}, (Q^-)^{-1} q^-_{\theta_1\theta_2}\rangle)(\theta_1 W_1 + \theta_2 W_2)^{-1}. \tag{2.35}$$

The best internal ellipsoid $\mathcal{E}(q^-_{\hat{\theta}_1\hat{\theta}_2}, Q^-_{\hat{\theta}_1\hat{\theta}_2})$ in the class (2.34-2.35), namely, such that

$$\mathcal{E}(q^-_{\theta_1\theta_2}, Q^-_{\theta_1\theta_2}) \subseteq \mathcal{E}(q^-_{\hat{\theta}_1\hat{\theta}_2}, Q^-_{\hat{\theta}_1\hat{\theta}_2}) \subseteq \mathcal{E}(q_1, Q_1) \cap \mathcal{E}(q_2, Q_2)$$

for all $0 \leq \theta_1, \theta_2 \leq 1$, is specified by the parameters

$$\hat{\theta}_1 = \frac{1 - \hat{\beta}_2}{1 - \hat{\beta}_1\hat{\beta}_2}, \qquad \hat{\theta}_2 = \frac{1 - \hat{\beta}_1}{1 - \hat{\beta}_1\hat{\beta}_2}, \tag{2.36}$$

with

$$\hat{\beta}_1 = \min(1, \beta_1), \qquad \hat{\beta}_2 = \min(1, \beta_2).$$

It is the ellipsoid that we look for: $\mathcal{E}(q^-, Q^-) = \mathcal{E}(q^-_{\hat{\theta}_1\hat{\theta}_2}, Q^-_{\hat{\theta}_1\hat{\theta}_2})$. Two special cases are

$$\hat{\theta}_1 = 1,\ \hat{\theta}_2 = 0 \quad\Rightarrow\quad \mathcal{E}(q_1, Q_1) \subseteq \mathcal{E}(q_2, Q_2) \quad\Rightarrow\quad \mathcal{E}(q^-, Q^-) = \mathcal{E}(q_1, Q_1),$$

and

$$\hat{\theta}_1 = 0,\ \hat{\theta}_2 = 1 \quad\Rightarrow\quad \mathcal{E}(q_2, Q_2) \subseteq \mathcal{E}(q_1, Q_1) \quad\Rightarrow\quad \mathcal{E}(q^-, Q^-) = \mathcal{E}(q_2, Q_2).$$

The method of finding the internal ellipsoidal approximation of the intersection of two ellipsoids is described in [9].

2.2.6 Intersection of Ellipsoid and Halfspace

Finding the intersection of ellipsoid and halfspace can be reduced to finding the intersection of two ellipsoids, one of which is unbounded. Let $\mathcal{E}(q_1, Q_1)$ be a nondegenerate ellipsoid and let $H(c, \gamma)$ define the halfspace

$$\mathbf{S}(c, \gamma) = \{x \in \mathbf{R}^n \mid \langle c, x\rangle \leq \gamma\}.$$

We have to determine if the intersection $\mathcal{E}(q_1, Q_1) \cap \mathbf{S}(c, \gamma)$ is empty, and if not, find its external and internal ellipsoidal approximations, $\mathcal{E}(q^+, Q^+)$ and $\mathcal{E}(q^-, Q^-)$. Two trivial situations are:

- $\mathbf{dist}(\mathcal{E}(q_1, Q_1), H(c, \gamma)) > 0$ and $\langle c, q_1 \rangle > 0$, which implies that $\mathcal{E}(q_1, Q_1) \cap \mathbf{S}(c, \gamma) = \emptyset$;
- $\mathbf{dist}(\mathcal{E}(q_1, Q_1), H(c, \gamma)) > 0$ and $\langle c, q_1 \rangle < 0$, so that $\mathcal{E}(q_1, Q_1) \subseteq \mathbf{S}(c, \gamma)$, and then $\mathcal{E}(q^+, Q^+) = \mathcal{E}(q^-, Q^-) = \mathcal{E}(q_1, Q_1)$.

In case $\mathbf{dist}(\mathcal{E}(q_1, Q_1), H(c, \gamma) < 0$, i.e. the ellipsoid intersects the hyperplane,

$$\mathcal{E}(q_1, Q_1) \cap \mathbf{S}(c, \gamma) = \mathcal{E}(q_1, Q_1) \cap \{x \mid \langle (x - q_2), W_2(x - q_2) \rangle \leq 1\},$$

with

$$\begin{aligned} q_2 &= (\gamma + 2\sqrt{\overline{\lambda}})c, & (2.37) \\ W_2 &= \frac{1}{4\overline{\lambda}} cc^T, & (2.38) \end{aligned}$$

$\overline{\lambda}$ being the biggest eigenvalue of matrix Q_1. After defining $W_1 = Q_1^{-1}$, we obtain $\mathcal{E}(q^+, Q^+)$ from equations (2.27-2.31), and $\mathcal{E}(q^-, Q^-)$ from (2.34-2.35), (2.36).

Remark. Notice that matrix W_2 has rank 1, which makes it singular for $n > 1$. Nevertheless, expressions (2.27-2.28), (2.34-2.35) make sense because W_1 is nonsingular, $\pi_0 \neq 0$ and $\hat{\theta}_1 \neq 0$.

To find the ellipsoidal approximations $\mathcal{E}(q^+, Q^+)$ and $\mathcal{E}(q^-, Q^-)$ of the intersection of ellipsoid $\mathcal{E}(q, Q)$ and polytope $P(C, g)$, $C \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$, such that

$$\mathcal{E}(q^-, Q^-) \subseteq \mathcal{E}(q, Q) \cap P(C, g) \subseteq \mathcal{E}(q^+, Q^+),$$

we first compute

$$\mathcal{E}(q_1^-, Q_1^-) \subseteq \mathcal{E}(q, Q) \cap \mathbf{S}(c_1, \gamma_1) \subseteq \mathcal{E}(q_1^+, Q_1^+),$$

wherein $\mathbf{S}(c_1, \gamma_1)$ is the halfspace defined by the first row of matrix C, c_1, and the first element of vector g, γ_1. Then, one by one, we get

$$\begin{aligned} &\mathcal{E}(q_2^-, Q_2^-) \subseteq \mathcal{E}(q_1^-, Q_1^-) \cap \mathbf{S}(c_2, \gamma_2), \quad \mathcal{E}(q_1^+, Q_1^+) \cap \mathbf{S}(c_2, \gamma_2) \subseteq \mathcal{E}(q_2^+, Q_2^+), \\ &\mathcal{E}(q_3^-, Q_3^-) \subseteq \mathcal{E}(q_2^-, Q_2^-) \cap \mathbf{S}(c_3, \gamma_3), \quad \mathcal{E}(q_2^+, Q_2^+) \cap \mathbf{S}(c_3, \gamma_3) \subseteq \mathcal{E}(q_3^+, Q_3^+), \\ &\cdots \\ &\mathcal{E}(q_m^-, Q_m^-) \subseteq \mathcal{E}(q_{m-1}^-, Q_{m-1}^-) \cap \mathbf{S}(c_m, \gamma_m), \quad \mathcal{E}(q_{m-1}^+, Q_{m-1}^+) \cap \mathbf{S}(c_m, \gamma_m) \subseteq \mathcal{E}(q_m^+, Q_m^+), \end{aligned}$$

The resulting ellipsoidal approximations are

$$\mathcal{E}(q^+, Q^+) = \mathcal{E}(q_m^+, Q_m^+), \quad \mathcal{E}(q^-, Q^-) = \mathcal{E}(q_m^-, Q_m^-).$$

Chapter 3

Reachability

3.1 Continuous-Time Systems

Consider the system

$$\dot{x}(t) = A(t)x(t) + B(t)u(t, x(t)) + G(t)v(t), \tag{3.1}$$

in which $x \in \mathbf{R}^n$ is the state, $u \in \mathbf{R}^m$ is the control and $v \in \mathbf{R}^d$ is the disturbance. $A(t)$, $B(t)$ and $G(t)$ are continuous and take their values in $\mathbf{R}^{n\times n}$, $\mathbf{R}^{n\times m}$ and $\mathbf{R}^{n\times d}$ respectively. Control $u(t, x(t))$ and disturbance $v(t)$ are measurable functions restricted by ellipsoidal constraints: $u(t, x(t)) \in \mathcal{E}(p(t), P(t))$ and $d(t) \in \mathcal{E}(q(t), Q(t))$. The set of initial conditions is assumed to be the ellipsoid $\mathcal{E}(x_0, X_0)$.

The state transition matrix of system (3.1) is

$$\dot{\Phi}(t, t_0) = A(t)\Phi(t, t_0), \quad \Phi(t, t) = I,$$

which for constant matrix A simplifies to

$$\Phi(t, t_0) = e^{A(t-t_0)}.$$

Although (3.1) indicates that control $u(t, x(t))$ depends on the state, hence is closed-loop, the system may also be controlled by open-loop control $u(t, x(t)) = u(t)$, $t \geq t_0$. The reachable set of system (3.1) under open-loop control (OLRS) is different from the closed-loop reach set (CLRS).

We distinguish two types of OLRS. We are given the set of initial conditions $\mathcal{E}(x_0, X_0)$, initial time t_0 and time $t > t_0$.

Definition 3.1.1 (OLRS of maxmin type) *The maxmin open-loop reach set $\mathcal{X}^+(t, t_0, \mathcal{E}(x_0, X_0))$ is the set of all states x, such that for any disturbance $v(\tau) \in \mathcal{E}(q(\tau), Q(\tau))$ there exist initial state $x^0 \in \mathcal{E}(x_0, X_0)$ and control $u(\tau) \in \mathcal{E}(p(\tau), P(\tau))$, which steers the system from $x(t_0) = x^0$ to $x(t) = x$.*

Definition 3.1.2 (OLRS of minmax type) *The minmax open-loop reach set $\mathcal{X}^-(t, t_0, \mathcal{E}(x_0, X_0))$ is the set of all states x, such that there exists control $u(\tau) \in \mathcal{E}(p(\tau), P(\tau))$ that for all disturbances $v(\tau) \in \mathcal{E}(q(\tau), Q(\tau))$ assigns initial state $x^0 \in \mathcal{E}(x_0, X_0)$ and steers the system from $x(t_0) = x^0$ to $x(t) = x$.*

Remark. The terms 'maxmin' and 'minmax' come from the fact that $\mathcal{X}^+(t, t_0, \mathcal{E}(x_0, X_0))$ is a subzero level set of the value function

$$V^-(t, x) = \max_v \min_u \{\mathbf{dist}(x(t_0), \mathcal{E}(x_0, X_0)) \mid x(t) = x)\},$$

and $\mathcal{X}^-(t, t_0, \mathcal{E}(x_0, X_0))$ is a subzero level set of the value function

$$V^+(t, x) = \min_u \max_v \{\mathbf{dist}(x(t_0), \mathcal{E}(x_0, X_0)) \mid x(t) = x)\}.$$

In maxmin case, the control is chosen for the *known* disturbance in the time interval $[t_0, t]$, and the reach set is

$$\begin{aligned}&\mathcal{X}^+(t, t_0, \mathcal{E}(x_0, X_0)) = \\ &\Big(\Phi(t, t_0)\mathcal{E}(x_0, X_0) \oplus \int_{t_0}^{t} \Phi(t, \tau)B(\tau)\mathcal{E}(p(\tau), P(\tau))d\tau\Big) \\ &\dot{-} \int_{t_0}^{t} \Phi(t, \tau)(-G(\tau))\mathcal{E}(q(\tau), Q(\tau))d\tau.\end{aligned} \tag{3.2}$$

In minmax case, the control in the time interval $[t_0, t]$ is chosen with *no* knowledge of the disturbance and the reach set is

$$\begin{aligned}&\mathcal{X}^-(t, t_0, \mathcal{E}(x_0, X_0)) = \\ &\Big(\Phi(t, t_0)\mathcal{E}(x_0, X_0) \dot{-} \int_{t_0}^{t} \Phi(t, \tau)(-G(\tau))\mathcal{E}(q(\tau), Q(\tau))d\tau\Big) \\ &\oplus \int_{t_0}^{t} \Phi(t, \tau)B(\tau)\mathcal{E}(p(\tau), P(\tau))d\tau.\end{aligned} \tag{3.3}$$

Since for any $\mathcal{W}_1, \mathcal{W}_2, \mathcal{W}_3 \subseteq \mathbf{R}^n$ it is true that

$$(\mathcal{W}_1 \dot{-} \mathcal{W}_2) \oplus \mathcal{W}_3 = (\mathcal{W}_1 \oplus \mathcal{W}_3) \dot{-} (\mathcal{W}_2 \oplus \mathcal{W}_3) \subseteq (\mathcal{W}_1 \oplus \mathcal{W}_3) \dot{-} \mathcal{W}_2, \tag{3.4}$$

the following inclusion holds,

$$\mathcal{X}^-(t, t_0, \mathcal{E}(x_0, X_0)) \subseteq \mathcal{X}^+(t, t_0, \mathcal{E}(x_0, X_0)).$$

Fixing a time instant τ_1, $t_0 < \tau_1 < t$, define *sequential* maxmin OLRS with *one correction*:

$$\mathcal{X}_1^+(t, t_0, \mathcal{E}(x_0, X_0)) = \mathcal{X}^+(t, \tau_1, \mathcal{X}^+(\tau_1, t_0, \mathcal{E}(x_0, X_0))),$$

and *sequential* minmax OLRS with *one correction*:

$$\mathcal{X}_1^-(t, t_0, \mathcal{E}(x_0, X_0)) = \mathcal{X}^-(t, \tau_1, \mathcal{X}^-(\tau_1, t_0, \mathcal{E}(x_0, X_0))).$$

Following (3.2) and (3.3), we obtain integral formulas for $\mathcal{X}_1^+(t, t_0, \mathcal{E}(x_0, X_0))$,

$$\begin{aligned}&\mathcal{X}_1^+(t, t_0, \mathcal{E}(x_0, X_0)) = \\ &\Big(\Phi(t, \tau_1)\Big(\Big(\Phi(\tau_1, t_0)\mathcal{E}(x_0, X_0) \oplus \int_{t_0}^{\tau_1} \Phi(\tau_1, \tau)B(\tau)\mathcal{E}(p(\tau), P(\tau))d\tau\Big) \\ &\dot{-} \int_{t_0}^{\tau_1} \Phi(\tau_1, \tau)(-G(\tau))\mathcal{E}(q(\tau), Q(\tau))d\tau\Big) \oplus \int_{\tau_1}^{t} \Phi(t, \tau)B(\tau)\mathcal{E}(p(\tau), P(\tau))d\tau\Big) \\ &\dot{-} \int_{\tau_1}^{t} \Phi(t, \tau)(-G(\tau))\mathcal{E}(q(\tau), Q(\tau))d\tau,\end{aligned}$$

and for $\mathcal{X}_1^-(t,t_0,\mathcal{E}(x_0,X_0))$,

$$\begin{aligned}
&\mathcal{X}_1^-(t,t_0,\mathcal{E}(x_0,X_0)) = \\
&\Big(\Phi(t,\tau_1)\Big(\Big(\Phi(\tau_1,t_0)\mathcal{E}(x_0,X_0)\dot{-}\int_{t_0}^{\tau_1}\Phi(\tau_1,\tau)(-G(\tau))\mathcal{E}(q(\tau),Q(\tau))d\tau\Big) \\
&\oplus\int_{t_0}^{\tau_1}\Phi(\tau_1,\tau)B(\tau)\mathcal{E}(p(\tau),P(\tau))d\tau\Big)\dot{-}\int_{\tau_1}^{t}\Phi(t,\tau)(-G(\tau))\mathcal{E}(q(\tau),Q(\tau))d\tau\Big) \\
&\oplus\int_{\tau_1}^{t}\Phi(t,\tau)B(\tau)\mathcal{E}(p(\tau),P(\tau))d\tau.
\end{aligned}$$

Recursively, sequential maxmin and minmax OLRS with k *corrections* for $t_0 < \tau_1 < \cdots < \tau_k < t$ can be defined,

$$\mathcal{X}_k^+(t,t_0,\mathcal{E}(x_0,X_0)) = \mathcal{X}^+(t,\tau_k,\mathcal{X}_{k-1}^+(\tau_k,t_0,\mathcal{E}(x_0,X_0))),$$

and

$$\mathcal{X}_k^-(t,t_0,\mathcal{E}(x_0,X_0)) = \mathcal{X}^-(t,\tau_k,\mathcal{X}_{k-1}^-(\tau_k,t_0,\mathcal{E}(x_0,X_0))).$$

By (3.4) the following inclusion holds,

$$\begin{aligned}
&\mathcal{X}^-(t,t_0,\mathcal{E}(x_0,X_0)) \subseteq \mathcal{X}_1^-(t,t_0,\mathcal{E}(x_0,X_0)) \subseteq \cdots \subseteq \mathcal{X}_k^-(t,t_0,\mathcal{E}(x_0,X_0)) \subseteq \\
&\mathcal{X}_k^+(t,t_0,\mathcal{E}(x_0,X_0)) \subseteq \cdots \subseteq \mathcal{X}_1^+(t,t_0,\mathcal{E}(x_0,X_0)) \subseteq \mathcal{X}^+(t,t_0,\mathcal{E}(x_0,X_0)).
\end{aligned} \tag{3.5}$$

As $k \to \infty$, we arrive at (see [4])

$$\mathcal{X}_\infty^+(t,t_0,\mathcal{E}(x_0,X_0)) = \mathcal{X}_\infty^-(t,t_0,\mathcal{E}(x_0,X_0)) = \mathcal{X}(t,t_0,\mathcal{E}(x_0,X_0)),$$

in which $\mathcal{X}(t,t_0,\mathcal{E}(x_0,X_0))$ is CLRS.

Definition 3.1.3 (CLRS) *Given initial time t_0 and the set of initial conditions $\mathcal{E}(x_0,X_0)$, the closed-loop reach set $\mathcal{X}(t,t_0,\mathcal{E}(x_0,X_0))$ of system (3.1) at time $t > t_0$ is the set of all states x for each of which there exist initial condition $x^0 \in \mathcal{E}(x_0,X_0)$ and control $u(\tau,x(\tau))$ that for every disturbance $v(\tau) \in \mathcal{E}(q(\tau),Q(\tau))$ assigns trajectory $x(\tau)$ satisfying*

$$\dot{x}(\tau) \in A(\tau)x(\tau) + B(\tau)u(\tau,x(\tau)) + G(\tau)v(\tau),$$

with $t_0 \le \tau \le t$, such that $x(t_0) = x_0$ and $x(t) = x$.

It follows from (3.5) that maxmin OLRS is an external bound and minmax OLRS is an internal bound for CLRS:

$$\mathcal{X}^-(t,t_0,\mathcal{E}(x_0,X_0)) \subseteq \mathcal{X}(t,t_0,\mathcal{E}(x_0,X_0)) \subseteq \mathcal{X}^+(t,t_0,\mathcal{E}(x_0,X_0)).$$

CLRS can be empty. This happens for example, if the set of initial conditions is reduced to a single state x_0 and control $u(t)$ is fixed, but the disturbance bound $\mathcal{E}(q(t),Q(t))$ is a nondegenerate ellipsoid for all t. A sufficient condition for $\mathcal{X}(t,t_0,\mathcal{E}(x_0,X_0) \neq \emptyset$ is

$$\mathcal{E}(0,G(\tau)Q(\tau)G^T(\tau)) \subseteq \mathcal{E}(0,B(\tau)P(\tau)B^T(\tau)) \tag{3.6}$$

for $t_0 \le \tau \le t$. To ensure $\mathcal{X}(t,t_0,\mathcal{E}(x_0,X_0)) \neq \emptyset$ when (3.6) does not hold, the initial set $\mathcal{E}(x_0,X_0)$ must be sufficiently big.

Remark. If matrix $Q(\cdot) = 0$, the system (3.1) becomes an ordinary affine system with known $v(\cdot) = q(\cdot)$. If matrix $G(\cdot) = 0$, (3.1) reduces to a linear controlled system. In the absence of disturbance ($Q(\cdot) = 0$ or $G(\cdot) = 0$), reach set $\mathcal{X}(t, t_0, \mathcal{E}(x_0, X_0))$ is the set of all states x to which system (3.1) can be steered at time t through all admissible controls $u(\tau, x(\tau)) \in \mathcal{E}(p(\tau), P(\tau))$ ($u(\tau) \in \mathcal{E}(p(\tau), P(\tau))$) starting at any $x^0 \in \mathcal{E}(x_0, X_0)$ at time t_0, $t_0 \leq \tau < t$. In this case, the reach set is always nonempty and is the same for open-loop and closed-loop controls.

The reach sets for systems with disturbances computed by *Ellipsoidal Toolbox* are CLRS. Hence, from now on, and further when describing backward reachability, by reach set we refer to CLRS.

The reach set $\mathcal{X}(t, t_0, \mathcal{E}(x_0, X_0))$ is a symmetric compact convex set. It satisfies the semigroup property:

$$\mathcal{X}(t, t_0, \mathcal{E}(x_0, X_0)) = \mathcal{X}(t, \tau, \mathcal{X}(\tau, t_0, \mathcal{E}(x_0, X_0))), \quad t_0 \leq \tau \leq t.$$

The reach set can be approximated by the parametrized families of external and internal ellipsoids, $\mathcal{E}(x_c(t), X_l^+(t))$ and $\mathcal{E}(x_c(t), X_l^-(t))$ respectively:

$$\mathcal{E}(x_c(t), X_l^-(t)) \subseteq \mathcal{X}(t, t_0, \mathcal{E}(x_0, X_0)) \subseteq \mathcal{E}(x_c(t), X_l^+(t)).$$

The trajectory of the center is governed by the equation

$$\dot{x}_c(t) = A(t)x_c(t) + B(t)p(t) + G(t)q(t), \quad x_c(t_0) = x_0. \tag{3.7}$$

The equation for the shape matrix of the external ellipsoid is

$$\begin{aligned}
\dot{X}_l^+(t) &= A(t)X_l^+(t) + X_l^+(t)A^T(t) && (3.8)\\
&+ \pi_l(t)X_l^+(t) + \frac{1}{\pi_l(t)}B(t)P(t)B^T(t) && (3.9)\\
&- X_l^{+1/2}(t)S_l(t)(G(t)Q(t)G^T(t))^{1/2} - (G(t)Q(t)G^T(t))^{1/2}S_l^T(t)X_l^{+1/2}(t), && (3.10)\\
& X_l^+(t_0) = X_0, && (3.11)
\end{aligned}$$

in which

$$\pi_l(t) = \frac{\langle l, \Phi(t_0, t)B(t)P(t)B^T(t)\Phi^T(t_0, t)l\rangle^{1/2}}{\langle l, \Phi(t_0, t)X_l^+(t)\Phi^T(t_0, t)l\rangle^{1/2}},$$

and matrix $S_l(t)$ is orthogonal ($S_l(t)S_l^T(t) = I$), determined from the equation

$$S_l(t)(G(t)Q(t)G^T(t))^{1/2}\Phi^T(t_0, t)l = \frac{\langle l, \Phi(t_0, t)G(t)Q(t)G^T(t)\Phi^T(t_0, t)l\rangle^{1/2}}{\langle l, \Phi(t_0, t)X_l^+\Phi^T(t_0, t)l\rangle^{1/2}} X_l^{+1/2}\Phi^T(t_0, t)l.$$

In the presence of disturbance if the reach set is empty, the matrix $X_l^+(t)$ becomes sign indefinite. For a system without disturbance, the part indicated by (3.10) naturally vanishes from the equation (3.8-3.11).

The equation for the shape matrix of the internal ellipsoid is

$$\begin{aligned}
\dot{X}_l^-(t) &= A(t)X_l^-(t) + X_l^-(t)A^T(t) && (3.12)\\
&+ X_l^{-1/2}(t)T_l(t)(B(t)P(t)B^T(t))^{1/2} + (B(t)P(t)B^T(t))^{1/2}T_l^T(t)X_l^{-1/2}(t) && (3.13)\\
&- \eta_l(t)X_l^-(t) - \frac{1}{\eta_l(t)}G(t)Q(t)G^T(t), && (3.14)\\
& X_l^-(t_0) = X_0, && (3.15)
\end{aligned}$$

in which

$$\eta_l(t) = \frac{\langle l, \Phi(t_0, t)G(t)Q(t)G^T(t)\Phi^T(t_0, t)l\rangle^{1/2}}{\langle l, \Phi(t_0, t)X_l^+(t)\Phi^T(t_0, t)l\rangle^{1/2}},$$

and matrix $T_l(t)$ is orthogonal, determined from the equation

$$T_l(t)(B(t)P(t)B^T(t))^{1/2}\Phi^T(t_0,t)l = \frac{\langle l, \Phi(t_0,t)B(t)P(t)B^T(t)\Phi^T(t_0,t)l\rangle^{1/2}}{\langle l, \Phi(t_0,t)X_l^-\Phi^T(t_0,t)l\rangle^{1/2}} X_l^{-1/2}\Phi^T(t_0,t)l.$$

Similarly to the external case, the part indicated by (3.14) vanishes from the equation (3.12-3.15) for a system without disturbance. Here vector $l \in \mathbf{R}^n$ is a parameter, and approximations $\mathcal{E}(x_c(t), X_l^+(t))$, $\mathcal{E}(x_c(t), X_l^-(t))$ are tight—they touch the boundary of the reach set $\mathcal{X}(t, t_0, \mathcal{E}(x_0, X_0))$ at the point determined by the direction $\Phi(t_0,t)l$:

$$\rho(\Phi(t_0,t)l \mid \mathcal{E}(x_c(t), X_l^-(t))) = \rho(\Phi(t_0,t)l \mid \mathcal{X}(t,t_0,\mathcal{E}(x_0,X_0))) = \rho(\Phi(t_0,t)l \mid \mathcal{E}(x_c(t), X_l^+(t))). \tag{3.16}$$

The boundary point where the external and internal ellipsoids touch the boundary of the reach set is given by

$$x_l^*(t) = x_c(t) + \frac{X_l^+(t)\Phi^T(t_0,t)l}{\langle l, \Phi(t_0,t)X_l^+(t)\Phi^T(t_0,t)l\rangle^{1/2}} = x_c(t) + \frac{X_l^-(t)\Phi^T(t_0,t)l}{\langle l, \Phi(t_0,t)X_l^-(t)\Phi^T(t_0,t)l\rangle^{1/2}}.$$

Points $x_l^*(t)$ form trajectories, to which we refer as *good curves*. Due to the nonsingular nature of matrix $\Phi(t_0,t)$, for every boundary point of the reach set there exists a good curve to which it belongs. To follow a good curve specified by parameter l, the system has to start at time t_0 at initial state

$$x_l^0 = x_0 + \frac{X_0 l}{\langle l, X_0 l\rangle^{1/2}}, \tag{3.17}$$

and the control must be chosen as

$$u_l(t) = p(t) + \frac{P(t)B^T(t)\Phi(t_0,t)l}{\langle l, \Phi(t_0,t)B(t)P(t)B^T(t)\Phi(t_0,t)l\rangle^{1/2}}. \tag{3.18}$$

This is an open-loop control. If there is no disturbance, it steers the system along the good curve defined by vector l. In the presence of disturbance, this control keeps the system on a good curve if and only if the disturbance plays against the control always taking its extreme values.

Expression (3.16) leads to the following fact:

$$\bigcup_{\langle l,l\rangle=1} \mathcal{E}(x_c(t), X_l^-(t)) = \mathcal{X}(t,t_0,\mathcal{E}(x_0,X_0)) = \bigcap_{\langle l,l\rangle=1} \mathcal{E}(x_c(t), X_l^+(t)). \tag{3.19}$$

In practice this means that the more values of l we use to compute $X_l^+(t)$ and $X_l^-(t)$, the better will be our approximation.

Analogous results hold for the backward reach set.

Definition 3.1.4 *Given the terminating time t_1 and target set $\mathcal{E}(y_1, Y_1)$, the backward reach set (closed-loop) $\mathcal{Y}(t_1, t, \mathcal{E}(y_1, Y_1))$ of system (3.1) at time $t < t_1$ is the set of all states y, for each of which there exists terminating state $y^1 \in \mathcal{E}(y_1, Y_1)$ and control $u(\tau, y(\tau))$ that for every disturbance $v(\tau) \in \mathcal{E}(q(\tau), Q(\tau))$ assigns trajectory $y(\tau)$ satisfying*

$$\dot{y}(\tau) \in A(\tau)y(\tau) + B(\tau)u(\tau, y(\tau)) + G(\tau)v(\tau),$$

where $t \le \tau < t_1$, $y(t) = y$ and $y(t_1) = y^1$.

The backward reach set satisfies the semigroup property:

$$\mathcal{Y}(t_1,t,\mathcal{E}(y_1,Y_1)) = \mathcal{Y}(\tau,t,\mathcal{Y}(t_1,\tau,\mathcal{E}(y_1,Y_1))), \quad t \leq \tau \leq t_1.$$

Being convex and compact, the backward reach set $\mathcal{Y}(t_1,t,\mathcal{E}(y_1,Y_1))$ can be tightly approximated by external ellipsoids $\mathcal{E}(y_c(t),Y_l^+(t))$ and internal ellipsoids $\mathcal{E}(y_c(t),Y_l^-(t))$ parametrized by the direction vector $l \in \mathbf{R}^n$. The equation for the center is given by

$$y_c(t) = Ay_c(t) + B(t)p(t) + G(t)q(t), \quad y_c(t_1) = y_1. \tag{3.20}$$

The equation for the shape matrix of the external ellipsoid is

$$\begin{aligned}
\dot{Y}_l^+(t) &= A(t)Y_l^+(t) + Y_l^+(t)A^T(t) && (3.21)\\
&- \pi_l(t)Y_l^+(t) - \frac{1}{\pi_l(t)}B(t)P(t)B^T(t) && (3.22)\\
&+ Y_l^{+1/2}(t)S_l(t)(G(t)Q(t)G^T(t))^{1/2} + (G(t)Q(t)G^T(t))^{1/2}S_l^T(t)Y_l^{+1/2}(t), && (3.23)\\
&\quad Y_l^+(t_1) = Y_1, && (3.24)
\end{aligned}$$

in which

$$\pi_l(t) = \frac{\langle l, \Phi(t_1,t)B(t)P(t)B^T(t)\Phi^T(t_1,t)l\rangle^{1/2}}{\langle l, \Phi(t_1,t)Y_l^+(t)\Phi^T(t_1,t)l\rangle^{1/2}},$$

and matrix $S_l(t)$ is orthogonal and satisfies the equation

$$S_l(t)(G(t)Q(t)G^T(t))^{1/2}\Phi^T(t_1,t)l = \frac{\langle l, \Phi(t_1,t)G(t)Q(t)G^T(t)\Phi^T(t_1,t)l\rangle^{1/2}}{\langle l, \Phi(t_1,t)Y_l^+\Phi^T(t_1,t)l\rangle^{1/2}}Y_l^{+1/2}\Phi^T(t_1,t)l.$$

The equation for the shape matrix of the internal ellipsoid is

$$\begin{aligned}
\dot{Y}_l^-(t) &= A(t)Y_l^-(t) + Y_l^-(t)A^T(t) && (3.25)\\
&- Y_l^{-1/2}(t)T_l(t)(B(t)P(t)B^T(t))^{1/2} - (B(t)P(t)B^T(t))^{1/2}T_l^T(t)Y_l^{-1/2}(t) && (3.26)\\
&+ \eta_l(t)Y_l^-(t) + \frac{1}{\eta_l(t)}G(t)Q(t)G^T(t), && (3.27)\\
&\quad Y_l^-(t_1) = Y_1, && (3.28)
\end{aligned}$$

in which

$$\eta_l(t) = \frac{\langle l, \Phi(t_1,t)G(t)Q(t)G^T(t)\Phi^T(t_1,t)l\rangle^{1/2}}{\langle l, \Phi(t_1,t)Y_l^+(t)\Phi^T(t_1,t)l\rangle^{1/2}},$$

and matrix $T_l(t)$ is orthogonal and determined from the equation

$$T_l(t)(B(t)P(t)B^T(t))^{1/2}\Phi^T(t_1,t)l = \frac{\langle l, \Phi(t_1,t)B(t)P(t)B^T(t)\Phi^T(t_1,t)l\rangle^{1/2}}{\langle l, \Phi(t_1,t)Y_l^-\Phi^T(t_1,t)l\rangle^{1/2}}Y_l^{-1/2}\Phi^T(t_1,t)l.$$

Just as in the forward reachability case, the parts (3.23) and (3.27) vanish from equations (3.21-3.24) and (3.25-3.28) in the absence of disturbance. Boundary value problems (3.20), (3.21-3.24) and (3.25-3.28) are converted to the initial value problems by the change of variables $s = -t$.

Remark. In expressions (3.9), (3.14), (3.22) and (3.27) the terms $\frac{1}{\pi_l(t)}$ and $\frac{1}{\eta_l(t)}$ may not be well defined for some vectors l, because matrices $B(t)P(t)B^T(t)$ and $G(t)Q(t)G^T(t)$ may be singular. In such cases, we take these entire expressions to equal zero.

For more detail on reach set calculation and good curves, and proofs, see [3, 7]. For more information on systems with disturbances, see [4, 5].

3.2 Discrete-Time Systems

Consider the discrete-time affine system:

$$x[k+1] = A[k]x[k] + B[k]u[k] + G[k]v[k], \tag{3.29}$$

where $x[k] \in \mathbf{R}^n$ is the state, $u[k] \in \mathbf{R}^m$ is the control bounded by the ellipsoid $\mathcal{E}(p[k], P[k])$, $v[k] \in \mathbf{R}^d$ is some fixed disturbance term, and matrices $A[k]$, $B[k]$, $G[k]$ are in $\mathbf{R}^{n\times n}$, $\mathbf{R}^{n\times m}$, $\mathbf{R}^{n\times d}$ respectively.

Remark. Notice that $v[k]$ is fixed. Currently, the *Ellipsoidal Toolbbox* does not accept discrete-time systems with unknown disturbances.

The state transition matrix is

$$\Phi(k+1, k_0) = A[k]\Phi(k, k_0), \quad k \geq k_0, \quad \Phi(k, k) = I,$$

which for time-invariant systems simplifies as

$$\Phi(k, k_0) = A^{k-k_0}.$$

Definition 3.2.1 *Given the initial condition $x[k_0] = x^0 \in \mathcal{E}(x_0, X_0)$, the reach set $\mathcal{X}(k, k_0, x^0)$ of the system (3.29) at time step $k > k_0$ is the set of all states reachable by the system at time step k through all admissible controls $u[i] \in \mathcal{E}(p[i], P[i])$, $i = k_0, \cdots, k-1$.*

The reach set from the set of initial conditions $\mathcal{E}(x_0, X_0)$ is

$$\mathcal{X}(k, k_0, \mathcal{E}(x_0, X_0)) = \bigcup_{x^0 \in \mathcal{E}(x_0, X_0)} \mathcal{X}(k, k_0, x^0).$$

The reach set satisfies the semigroup property:

$$\mathcal{X}(k, k_0, \mathcal{E}(x_0, X_0)) = \mathcal{X}(k, i, \mathcal{X}(i, k_0, \mathcal{E}(x_0, X_0))), \quad k_0 \leq i \leq k.$$

If matrices $A[k]$ are nonsingular and the set of initial conditions $\mathcal{E}(x_0, X_0)$ is a nondegenerate ellipsoid, the reach set $\mathcal{X}(k, k_0, \mathcal{E}(x_0, X_0))$, $k > k_0$, is convex and compact, has nonempty interior, and can be approximated by the families of external and internal ellipsoids $\mathcal{E}(x_c[k], X_l^+[k])$, $\mathcal{E}(x_c[k], X_l^-[k])$, parametrized by vector $l \in \mathbf{R}^n$:

$$\mathcal{E}(x_c[k], X_l^-[k]) \subseteq \mathcal{X}(k, k_0, \mathcal{E}(x_0, X_0)) \subseteq \mathcal{E}(x_c[k], X_l^+[k]),$$

and at the same time

$$\rho(\Phi(k_0, k)l \mid \mathcal{E}(x_c[k], X_l^-[k])) = \rho(\Phi(k_0, k)l \mid \mathcal{X}(k, k_0, \mathcal{E}(x_0, X_0))) = \rho(\Phi(k_0, k)l \mid \mathcal{E}(x_c[k], X_l^+[k])).$$

The recurrence relation for the center is given by

$$x_c[k+1] = A[k]x_c[k] + B[k]p[k] + G[k]v[k], \quad x_c[k_0] = x_0. \tag{3.30}$$

The shape matrix of the external ellipsoid is determined from

$$X_l^+[k+1] = (1 + \pi_l[k])A[k]X_l^+[k]A^T[k] + (1 + \frac{1}{\pi_l[k]})B[k]P[k]B^T[k], \quad X_l^+[k_0] = X_0, \tag{3.31}$$

in which

$$\pi_l[k] = \frac{\langle l, \Phi(k_0,k+1)B[k]P[k]B^T[k]\Phi^T(k_0,k+1)l\rangle^{1/2}}{\langle l, \Phi(k_0,k)X_l^+[k]\Phi^T(k_0,k)l\rangle^{1/2}}, \quad k \geq k_0. \tag{3.32}$$

The shape matrix of the internal ellipsoid is

$$\begin{aligned} X_l^-[k+1] &= A[k]X_l^-[k]A^T[k] + B[k]P[k]B^T[k] \\ &+ A[k]X_l^{-1/2}[k]S_l[k](B[k]P[k]B^T[k])^{1/2} \\ &+ (B[k]P[k]B^T[k])^{1/2}S_l^T[k]X_l^{-1/2}[k]A^T[k], \quad X_l^-[k_0] = X_0, \end{aligned} \tag{3.33}$$

in which matrix $S_l[k]$ is orthogonal and determined from the equation

$$S_l[k](B[k]P[k]B^T[k])^{1/2}\Phi(k_0,k+1)l = \frac{\langle l, \Phi(k_0,k+1)B[k]P[k]B^T[k]\Phi^T(k_0,k+1)l\rangle^{1/2}}{\langle l, X_0 l\rangle^{1/2}} X_0^{1/2} l.$$

The point where the external and the internal ellipsoids both touch the boundary of the reach set is given by

$$x_l^*[k] = x_c[k] + \frac{X_l^+[k]\Phi^T(k_0,k)l}{\langle l, \Phi(k_0,k)X_l^+[k]\Phi^T(k_0,k)l\rangle^{1/2}} = x_c[k] + \frac{X_l^-[k]\Phi^T(k_0,k)l}{\langle l, \Phi(k_0,k)X_l^-[k]\Phi^T(k_0,k)l\rangle^{1/2}}.$$

Points $x_l^*[k]$, $k \geq k_0$, form a good curve. In order for the system to follow the good curve specified by some vector l, the initial state must be

$$x_l^0 = x_0 + \frac{X_0 l}{\langle l, X_0 l\rangle^{1/2}}, \tag{3.34}$$

and the control must be

$$u_l[k] = p[k] + \frac{P[k]B^T[k]\Phi^T(k_0,k+1)l}{\langle l, \Phi(k_0,k+1)B[k]P[k]B^T[k]\Phi(k_0,k+1)l\rangle^{1/2}}. \tag{3.35}$$

Remark. There are cases when $\pi_l[k]$ in (3.32) and $u_l[k]$ in (3.35) are not well defined due to singularity of the matrix $B[k]P[k]B^T[k]$. The way to handle these situations, as well as systems with singular $A[k]$, is described in detail in [6].

Backward reach sets for discrete-time systems can be computed only if matrices $A[k]$ are nonsingular for all k.

Definition 3.2.2 *Given the terminating position (k_1, y^1), $y[k_1] = y^1 \in \mathcal{E}(y_1, Y_1)$, the backward reach set $\mathcal{Y}(k_1, k, y^1)$ of the system (3.29) at time step $k < k_1$ is the set of all states $y[k]$ such that there exists a sequence of controls $u[i] \in \mathcal{E}(p[i], P[i])$, $i = k, \cdots, k_1 - 1$, that brings the system from $y[k]$ to $y[k_1] = y^1$ in $k_1 - k$ steps.*

The backward reach set for the set of terminating states $\mathcal{E}(y_1, Y_1)$ is

$$\mathcal{Y}(k_1, k, \mathcal{E}(y_1, Y_1)) = \bigcup_{y^1 \in \mathcal{E}(y_1, Y_1)} \mathcal{Y}(k_1, k, y^1).$$

The backward reach set satisfies the semigroup property:

$$\mathcal{Y}(k_1, k, \mathcal{E}(y_1, Y_1)) = \mathcal{Y}(i, k, \mathcal{Y}(k_1, i, \mathcal{E}(y_1, Y_1))), \quad k \leq i \leq k_1.$$

Backward reach sets can also be tightly approximated by external and internal ellipsoids, $\mathcal{E}(y_c[k], Y_l^+[k])$ and $\mathcal{E}(y_c[k], Y_l^-[k])$. The equation for the center $y_c[k]$ is

$$y_c[k] = A^{-1}[k](y_c[k+1] - B[k]p[k] - G[k]v[k]), \quad y_c[k_1] = y_1. \tag{3.36}$$

The equation for the shape matrix for the external ellipsoid is

$$\begin{aligned} Y_l^+[k] &= (1 + \pi_l[k])A^{-1}[k]Y_l^+[k+1]A^{-1T}[k] \\ &+ (1 + \frac{1}{\pi_l[k]})A^{-1}[k]B[k]P[k]B^T[k]A^{-1T}[k], \quad Y_l^+[k_1] = Y_1, \end{aligned} \tag{3.37}$$

in which

$$\pi_l[k] = \frac{\langle l, \Phi(k_1, k+1)B[k]P[k]B^T[k]\Phi^T(k_1, k+1)l\rangle^{1/2}}{\langle l, \Phi(k_1, k)Y_l^+[k+1]\Phi^T(k_1, k)l\rangle^{1/2}}, \quad k < k_1.$$

The expression for the shape matrix of the internal ellipsoid is

$$\begin{aligned} Y_l^-[k] &= A^{-1}[k]Y_l^-[k]A^{-1T}[k] + A^{-1}[k]B[k]P[k]B^T[k]A^{-1T}[k] \\ &+ A^{-1}[k]Y_l^{-1/2}[k+1]S_l[k](A^{-1}[k]B[k]P[k]B^T[k]A^{-1T}[k])^{1/2} \\ &+ (A^{-1}[k]B[k]P[k]B^T[k]A^{-1T}[k])^{1/2}S_l^T[k]Y_l^{-1/2}[k+1]A^{-1T}[k], \quad Y_l^-[k_1] = Y_1, \end{aligned} \tag{3.38}$$

in which matrix $S_l[k]$ is orthogonal and determined from the equation

$$S_l[k](A^{-1}[k]B[k]P[k]B^T[k]A^{-1T}[k])^{1/2}\Phi(k_1, k)l = \frac{\langle l, \Phi(k_1, k)A^{-1}[k]B[k]P[k]B^T[k]A^{-1T}[k]\Phi^T(k_1, k)l\rangle^{1/2}}{\langle l, Y_1 l\rangle^{1/2}} Y_1^{1/2} l.$$

Chapter 4

Installation

4.1 Additional Software

Some routines of the *Ellipsoidal Toolbox*, namely,

- `distance`
- `intersect`
- `intersection_ia`
- `isinside`

require solving semidefinite programming (SDP) problems. We use YALMIP ([14], [15]) as an interface to an external SDP solver. YALMIP supports a large variety of SDP packages. One of them, SeDuMi ([16], [17]) is distributed along with *ET*. The user is free to choose any other SDP solver so long as it is supported by YALMIP. The list of supported SDP solvers can be obtained from [15].

Both YALMIP and SeDuMi are included in the *ET* distribution, so you do not need to download them separately. However, if you have them already installed, or wish to install them independently of *ET*, you should download the lite version of *ET*.

4.2 Installation and Quick Start

1. Go to
   `http://www.eecs.berkeley.edu/~akurzhan/ellipsoids`
   and download the *Ellipsoidal Toolbox* or its lite version stripped of YALMIP and SeDuMi.
2. Unzip the distribution file into the directory where you would like the toolbox to be.
3. Read the copyright notice.

4. In MATLAB command window change the working directory to the one where you unzipped the toolbox and type
   `>> install`

5. At this point, the directory tree of the *Ellipsoidal Toolbox* is added to the MATLAB path list. In order to save the updated path list, in your MATLAB window menu go to `File` → `Set Path...` and click `Save`.

6. To get an idea of what the toolbox is about, type
   `>> ell_demo1`
   This will produce a demo of basic *ET* functionality: how to create and manipulate ellipsoids. Type
   `>> ell_demo2`
   to learn how to plot ellipsoids and hyperplanes in 2 and 3D.
   For a quick tutorial on how to use the toolbox for reachability analysis and verification, type
   `>> ell_demo3`

Chapter 5

Implementation

5.1 Operations with Ellipsoids

In the *Ellipsoidal Toolbox* we define a new class `ellipsoid` inside the MATLAB programming environment. The following three commands define the same ellipsoid $\mathcal{E}(q, Q)$, with $q \in \mathbf{R}^n$ and $Q \in \mathbf{R}^{n\times n}$ being symmetric positive semidefinite:

```
>> E = ellipsoid(q, Q);
>> E = ellipsoid(Q) + q;
>> E = sqrtm(Q)*ell_unitball(size(Q, 1)) + q;
```

For the `ellipsoid` class we overload the following functions and operators:

- `isempty(E)` - checks if `E` is an empty ellipsoid.
- `display(E)` - displays the details of ellipsoid $\mathcal{E}(q, Q)$, namely, its center q and the shape matrix Q.
- `plot(E)` - plots ellipsoid $\mathcal{E}(q, Q)$ if its dimension is not greater than 3.
- `E1 == E2` - checks if ellipsoids $\mathcal{E}(q_1, Q_1)$ and $\mathcal{E}(q_2, Q_2)$ are equal.
- `E1 ~= E2` - checks if ellipsoids $\mathcal{E}(q_1, Q_1)$ and $\mathcal{E}(q_2, Q_2)$ are not equal.
- `[ , ]` - concatenates the ellipsoids into the horizontal array, e.g. `EE = [E1 E2 E3]`.
- `[ ; ]` - concatenates the ellipsoids into the vertical array, e.g. `EE = [E1 E2; E3 E4]` defines 2×2 array of ellipsoids.
- `E1 >= E2` - checks if the ellipsoid $\mathcal{E}(q_1, Q_1)$ is bigger than the ellipsoid $\mathcal{E}(q_2, Q_2)$, or equivalently $\mathcal{E}(0, Q_1) \subseteq \mathcal{E}(0, Q_2)$.
- `E1 <= E2` - checks if $\mathcal{E}(0, Q_2) \subseteq \mathcal{E}(0, Q_1)$.
- `-E` - defines ellipsoid $\mathcal{E}(-q, Q)$.

- `E + b` - defines ellipsoid $\mathcal{E}(q+b, Q)$.
- `E - b` - defines ellipsoid $\mathcal{E}(q-b, Q)$.
- `A * E` - defines ellipsoid $\mathcal{E}(q, AQA^T)$.
- `inv(E)` - inverts the shape matrix of the ellipsoid: $\mathcal{E}(q, Q^{-1})$.

All the listed operations can be applied to a single ellipsoid as well as to a two-dimensional array of ellipsoids. For example,

```
>> E1 = ellipsoid([2; -1], [9 -5; -5 4]);  % nondegenerate ellipsoid in R^2
>> E2 = polar(E1);  % E2 is polar ellipsoid for E1
>> E3 = inv(E2);  % E3 is generated from E2 by inverting its shape matrix
>> EE = [E1 E2; E3 ellipsoid([1; 1], eye(2))];  % 2x2 array of ellipsoids
>> EE <= E1  % check if E1 is bigger than each of the ellipsoids in EE

ans =

     1     0
     1     0
```

To access individual elements of the array, the usual MATLAB subindexing is used:

```
>> A = [0 1; -2 0]; b = [3; 0];  % A - 2x2 real matrix, b - vector in R^2
>> AT = A * EE(:, 2) + b;  % affine transformation of ellipsoids in the second column of EE
```

Sometimes it may be useful to modify the shape of the ellipsoid without affecting its center. Say, we would like to bloat or squeeze the ellipsoid:

```
>> BLT = shape(E1, 2);  % bloats ellipsoid E1
>> SQZ = shape(E1, 0.5);  % squeezes ellipsoid E1
```

Since function `shape` does not change the center of the ellipsoid, it only accepts scalars or square matrices as its second input parameter.

Several functions access the internal data of the ellipsoid object:

```
>> [q, Q] = parameters(E2)  % get the center and the shape matrix of E2

q =

   -0.5000
   -0.1667

Q =

   0.9167    0.9167
   0.9167    1.5278

>> D = ellipsoid([42 -7 -2 4; -7 10 3 1; -2 3 5 -2; 4 1 -2 2]);  % define new ellipsoid
>> isdegenerate([EE(1, :) D])  % check if given ellipsoids are degenerate

ans =

     0     0     1

>> [n, r] = dimension([EE(1, :) D])  % get space dimensions and ranks of the shape matrices

n =

   2     2     4

r =

   2     2     3
```

One way to check if two ellipsoids intersect, is to compute the distance between them:

```
>> distance(EE, E3)  % distance between E3 and each of the ellipsoids in EE

ans =
     -0.7761     -2.1553
     -1.3519      0.1683
```

This result indicates that the ellipsoid E3 does not intersect with the ellipsoid EE(2, 2), with all the other ellipsoids in EE it has nonempty intersection. If the intersection of the two ellipsoids is nonempty, it can be approximated by ellipsoids from the outside as well as from the inside:

```
>> EA = intersection_ea(E1, E3);  % external approximation of intersection of E1 and E3
>> IA = intersection_ia(E1, E3);  % internal approximation of intersection of E1 and E3
```

It can be checked that resulting ellipsoid EA contains the given intersection, whereas IA is contained in this intersection:

```
>> isinside(EA, [E1 E3], 'i')  % array [E1 E3] should be treated as intersection

ans =

     1

>> isinside([E1 E3], IA)  % check if IA belongs to the intersection of E1 and E3

ans =

     1
```

Function `isinside` in general checks if the intersection of ellipsoids in the given array contains the union or intersection of ellipsoids or polytopes.

It is also possible to solve the feasibility problem, that is, to check if the intersection of more than two ellipsoids is empty:

```
>> intersect(EE, EE(1, 1), 'i')  % check if the intersection of ellipsoids in EE is empty

ans =

    -1
```

In this particular example the result -1 indicates that the intersection of ellipsoids in `EE` is empty. Function `intersect` in general checks if an ellipsoid, hyperplane or polytope intersects the union or the intersection of ellipsoids in the given array:

```
>> % check if EE(2, 2) intersects the intersection of E1, E2 and E3:
>> intersect([E1 E2 E3], EE(2, 2), 'i')

ans =

    0

>> % check if E(2, 2) intersects the union of E1, E2 and E3:
>> intersect([E1 E2 E3], EE(2, 2), 'u')

ans =

    1
```

For the ellipsoids in $\mathbf{R}$, $\mathbf{R}^2$ and $\mathbf{R}^3$ the geometric sum can be computed explicitely and plotted:

```
>> minksum(EE);  % compute and plot the geometric sum of ellipsoids in EE
```

If the dimension of the space in which the ellipsoids are defined exceeds 3, an error is returned. The result of the geometric sum operation is not generally an ellipsoid, but it can be approximated by families of external and internal ellipsoids parametrized by the direction vector:

```
>> % define the set of directions:
>> L = [1 0; 1 1; 0 1; -1 1; 1 3]';  % columns of matrix L are vectors in R^2
>>
>> EA = minksum_ea(EE, L)  % compute external ellipsoids for the directions in L

EA =
1x5 array of ellipsoids.

>> IA = minksum_ia(EE, L)  % compute internal ellipsoids for the directions in L
```

```
IA =
1x5 array of ellipsoids.

>> % intersection of external ellipsoids should always contain
>> % the union of internal ellipsoids:
>> isinside(EA, IA, 'u')

ans =

     1
```

Functions minksum_ea and minksum_ia work for ellipsoids of arbitrary dimension. They should be used for general computations whereas minksum is there merely for visualization purposes.

If the geometric difference of two ellipsoids is not an empty set, it can be computed explicitely and plotted for ellipsoids in $\mathbf{R}$, $\mathbf{R}^2$ and $\mathbf{R}^3$:

```
>> E4 = shape(EE(2, 2), 0.4);  % ellipsoid defined by squeezing the ellipsoid EE(2, 2)
>> E1 >= E4  % check if the geometric difference E1 - E4 is nonempty

ans =

     1

>> minkdiff(E1, E4);  % compute and plot this geometric difference
```

Similar to minksum, minkdiff is there for visualization purpose. It works only for dimensions 1, 2 and 3, and for higher dimensions it returns an error. For arbitrary dimensions, the geometric difference can be approximated by families of external and internal ellipsoids parametrized by the direction vector, provided this direction is not bad:

```
>> isbaddirection(E1, E4, L)  % find out which of the directions in L are bad

ans =

     1     0     0     1     0

>> % two of five directions specified by L are bad,
>> % so, only three ellipsoidal approximations can be produced for this L:
>> EA = minkdiff_ea(E1, E4, L)

EA =
1x3 array of ellipsoids.

>> IA = minkdiff_ea(E1, E4, L)

IA =
1x3 array of ellipsoids.
```

The class `hyperplane` of the *Ellipsoidal Toolbox* is used to describe hyperplanes and halfspaces. The following two commands define one and the same hyperplane but two different halfspaces:

```
>> H = hyperplane([1; 1], 1);  % defines halfspace x1 + x2 <= 1
>> H = hyperplane([-1; -1], -1);  % defines halfspace x1 + x2 >= 1
```

The following functions and operators are overloaded for the `hyperplane` class:

- `isempty(H)` - checks if `H` is an empty hyperplane.
- `display(H)` - displays the details of hyperplane $H(c, \gamma)$, namely, its normal c and the scalar γ.
- `plot(H)` - plots hyperplane $H(c, \gamma)$ if the dimension of the space in which it is defined is not greater than 3.
- `H1 == H2` - checks if hyperplanes $H(c_1, \gamma_1)$ and $H(c_2, \gamma_2)$ are equal.
- `H1 ~= H2` - checks if hyperplanes $H(c_1, \gamma_1)$ and $H(c_2, \gamma_2)$ are not equal.
- `[ , ]` - concatenates the hyperplanes into the horizontal array, e.g. `HH = [H1 H2 H3]`.
- `[ ; ]` - concatenates the hyperplanes into the vertical array, e.g. `HH = [H1 H2; H3 H4]` - defines 2×2 array of hyperplanes.
- `-H` - defines hyperplane $H(-c, -\gamma)$, which is the same as $H(c, \gamma)$ but specifies different halfspace.

There are several ways to access the internal data of the `hyperplane` object:

```
>> [c, g] = parameters(H)  % get the normal and the scalar that define hyperplane H

c =

   -1
   -1


g =

   -1

>> dimension(H)  % get the dimension of the space where H is defined

ans =

     2

>> H0 = hyperplane([1 -1; 1 1]);  % define two hyperplanes passing through the origin
>> isparallel(H, H0)  % check which of two hyperplanes in array H0 is parallel to H

ans =

     1     0
```

All the functions of *Ellipsoidal Toolbox* that accept `hyperplane` object as parameter, work with single hyperplanes as well as with hyperplane arrays. One exception is the function `parameters` that allows only single `hyperplane` object.

An array of hyperplanes can be converted to the `polytope` object of the Multi-Parametric Toolbox ([12], [13]), and back:

```
>> define array of four hyperplanes:
>> HH = hyperplane([1 1; -1 -1; 1 -1; -1 1]', [2 2 2 2]);

HH =
1x4 array of hyperplanes.

>> P  = hyperplane2polytope(HH);  % convert array of hyperplanes to polytope
>> HP = polytope2hyperplane(P);  % covert polytope to array of hyperplanes
>> HP == HH

ans =

     1     1     1     1
```

Functions `hyperplane2polytope` and `polytope2hyperplane` require the Multi-Parametric Toolbox to be installed.

We can compute distance from ellipsoids to hyperplanes and polytopes:

```
>> distance(E1, HH)  % distance from ellipsoid E1 to each of the hyperplanes in HH

ans =

     -0.5176    0.8966   -2.6841    0.1444

>> distance(EE, P)  % distance from each of the ellipsoids in EE to the polytope P

ans =

     0     0
     0     0
```

A negative distance value in the case of ellipsoid and hyperplane means that the ellipsoid intersects the hyperplane. As we see in this example, ellipsoid `E1` intersects hyperplanes `H(1)` and `H(3)` and has no common points with `H(2)` and `H(4)`. When `distance` function has a polytope as a parameter, it always returns nonnegative values to be consistent with `distance` function of the Multi-Parametric Toolbox. Here, the zero distance values mean that each ellipsoid in `EE` has nonempty intersection with polytope `P`.

It can be checked if the union or intersection of given ellipsoids intersects given hyperplanes or polytopes:

```
>> % check if the union of ellipsoids in EE intersects  hyperplanes in HH:
>> intersect(EE, HH)

ans =

     1     1     1     1

>> % check if the intersection of ellipsoids in the first column of EE
>> % intersects with hyperplanes in HH:
>> intersect(EE(:, 1), HH, 'i')

ans =

     0     0     1     0

>> % check if the intersection of ellipsoids E1, E2 and E3
>> % intersects with polytope P:
>> intersect([E1 E2 E3], P, 'i')

ans =

     1
```

The intersection of ellipsoid and hyperplane can be computed exactly:

```
>> % compute the intersections of ellipsoids in the first column of EE
>> % with hyperplane H(3):
>> I = hpintersection(EE(:, 1), H(3))

I =
2x1 array of ellipsoids.

>> isdegenerate(I)  % resulting ellipsoids should lose rank

ans =

     1
     1
```

Functions `intersection_ea` and `intersection_ia` can be used with `hyperplane` objects, which in this case define halfspaces and `polytope` objects:

```
>> % compute external and internal ellipsoidal approximations
>> % of the intersections of ellipsoids in the first column of EE
>> % with the halfspace x1 - x2 <= 2:
>> EA1 = intersection_ea(EE(:, 1), H(3))  % get external ellipsoids

EA1 =
2x1 array of ellipsoids.
```

```
>> IA1 = intersection_ia(EE(:, 1), H(3))  % get internal ellipsoids

IA1 =
2x1 array of ellipsoids.

>> % compute external and internal ellipsoidal approximations
>> % of the intersections of ellipsoids in the first column of EE
>> % with the halfspace x1 - x2 >= 2:
>> EA2 = intersection_ea(EE(:, 1), -H(3));  % get external ellipsoids
>> IA2 = intersection_ia(EE(:, 1), -H(3));  % get internal ellipsoids

>> % compute ellipsoidal approximations of the intersection
>> % of ellipsoid E1 and polytope P:
>> EA = intersection_ea(E1, P);  % get external ellipsoid
>> IA = intersection_ia(E1, P);  % get internal ellipsoid
```

Function `isinside` can be used to check if a polytope or union of polytopes is contained in the intersection of given ellipsoids:

```
>> Q = 0.5*P + [1; 1];  % polytope Q is obtained by affine transformation of P
>>
>> % check if the intersection of ellipsoids in the first column of EE
>> % contains the union of polytopes P and Q:
>> isinside(EE(:, 1), [P Q])  % equivalent to: isinside(EE(:, 1), P | Q)

ans =

     0

>> % check if ellipsoid EE(2, 2) contains the intersection of P and Q:
>> isinside(EE(2, 2), [P Q], 'i')  % equivalent to: isinside(EE(2, 2), P & Q)

ans =

     1
```

Functions `distance`, `intersect`, `intersection_ia` and `isinside` use the YALMIP interface ([14], [15]) to the external optimization package. The default optimization package included in the distribution of the *Ellipsoidal Toolbox* is SeDuMi ([16],[17]). The user, however, is free to choose any other optimization tool that solves second order cone programming (SOCP) problems, as long as this tool is supported by YALMIP. QCQP is a special case of SOCP.

5.2 Reachability

To compute the reach sets of the systems described in chapter 3, we define two new classes in the *Ellipsoidal Toolbox*: class `linsys` for the system description, and class `reach` for the reach set data.

We start by explaining how to define a system using `linsys` object. For example, description of the system

$$\left[\begin{array}{c}\dot{x}_1\\ \dot{x}_2\end{array}\right] = \left[\begin{array}{cc}0 & 1\\ 0 & 0\end{array}\right]\left[\begin{array}{c}x_1\\ x_2\end{array}\right] + \left[\begin{array}{c}u_1(t)\\ u_2(t)\end{array}\right], \quad u(t)\in\mathcal{E}(p(t),P)$$

with

$$p(t) = \left[\begin{array}{c}\sin(t)\\ \cos(t)\end{array}\right], \quad P = \left[\begin{array}{cc}9 & 0\\ 0 & 2\end{array}\right],$$

is done by the following sequence of commands:

```
>> A = [0 1; 0 0]; B = eye(2);  % matrices A and B, B is identity
>> U.center = {'sin(t)'; 'cos(t)'};  % center of the ellipsoid depends on t
>> U.shape = [9 0; 0 2];  % shape matrix of the ellipsoid is static
>> sys = linsys(A, B, U);  % create linear system object
```

If matrices A or B depend on time, say $A(t) = \left[\begin{array}{cc}0 & 1-\cos(2t)\\ -\frac{1}{t} & 0\end{array}\right]$, then matrix `A` should be symbolic:

```
>> At = {'0' '1 - cos(2*t)'; '-1/t' '0'};  % A(t) - time-variant
>> sys_t = linsys(At, B, U);
```

To describe the system with disturbance

$$\left[\begin{array}{c}\dot{x}_1\\ \dot{x}_2\end{array}\right] = \left[\begin{array}{cc}0 & 1\\ 0 & 0\end{array}\right]\left[\begin{array}{c}x_1\\ x_2\end{array}\right] + \left[\begin{array}{c}u_1(t)\\ u_2(t)\end{array}\right] + \left[\begin{array}{c}0\\ 1\end{array}\right]v(t),$$

with bounds on control as before, and disturbance being $-1 \leq v(t) \leq 1$, we type:

```
>> G = [0; 1];  % matrix G
>> V = ellipsoid(1);  % disturbance bounds: unit ball in R
>> sys_d = linsys(A, B, U, G, V);
```

Control and disturbance bounds `U` and `V` can have different types. If the bound is constant, it should be described by `ellipsoid` object. If the bound depends on time, then it is represented by a structure with fields `center` and `shape`, one or both of which are symbolic. In system `sys`, the control bound `U` is defined as such a structure. Finally, if the control or disturbance is known and fixed, it should be defined as a vector, of type `double` if constant, or symbolic, if it depends on time.

To declare a discrete-time system

$$\left[\begin{array}{c}x_1[k+1]\\ x_2[k+1]\end{array}\right] = \left[\begin{array}{cc}0 & 1\\ -1 & -0.5\end{array}\right]\left[\begin{array}{c}x_1[k]\\ x_2[k]\end{array}\right] + \left[\begin{array}{c}0\\ 1\end{array}\right]u[k], \quad -1\leq u[k]\leq 1,$$

we use the same `linsys` constructor:

```
>> Ad = [0 1; -1 -0.5]; Bd = [0; 1];  % matrices A and B
>> Ud  = ellipsoid(1);  % control bounds: unit ball in R
>> dtsys = linsys(Ad, Bd, Ud, [], [], [], [], 'd');  % discrete-time system
```

Once the linsys object is created, we need to specify the set of initial conditions, the time interval and values of the direction vector, for which the reach set approximations must be computed:

```
>> X0 = ell_unitball(2)  % set of initial conditions
>> T = [0 10]  % time interval
>> L = [1 0; 0 1]';  % columns of L specify the directions
```

The reach set approximation is computed by calling the constructor of the reach object:

```
>> options.save_all = 1  % turn on save_all option (its default value is 0)
>> rs = reach(sys, X0, L, T, options);  % reach set of continuos-time system
```

The options parameter in the reach() call is optional. We shall soon explain why we used it here. At this point, variable rs contains the reach set approximations for the specified continuous-time system, time interval and set of initial conditions computed for given directions. By default, both external and internal approximations are computed. To compute only external or only internal approximations, options structure must contain another field - approximation. For only external approximations, this field must be set to 0, for only internal approximations, it must be set to 1. The reach set approximation data can be extracted in the form of arrays of ellipsoids:

```
>> EA = get_ea(rs)  % external approximating ellipsoids

EA =
4x200 array of ellipsoids.

>> [IA, tt] = get_ia(rs);  % internal approximating ellipsoids
```

Ellipsoidal arrays EA and IA have 4 rows because we computed the reach set approximations for 4 directions. Each row of ellipsoids corresponds to one direction. The number of columns in EA and IA is defined by the time_grid parameter of the global ellOptions structure (see chapter 6 for details). It represents the number of time values in our time interval, at which the approximations are evaluated. These time values are returned in the optinal output parameter, array tt, whose length is the same as the number of columns in EA and IA. Intersection of ellipsoids in a particular column of EA gives external ellipsoidal approximation of the reach set at corresponding time. Internal ellipsoidal approximation of this set at this time is given by the union of ellipsoids in the same column of IA.

We may be interested in the reachability data of our system in some particular time interval, smaller than the one for which the reach set was computed, say $3 \leq t \leq 5$. This data can be extracted and returned in the form of reach object by the cut function:

```
>> ct = cut(rs, [3 5]);  % reach set for the time interval [3, 5]
```

To obtain a snap shot of the reach set at given time, the same function cut is used:

```
>> ct = cut(rs, 5);  % reach set at time t = 5
```

It can be checked if the external or internal reach set approximation intersects with given ellipsoids, hyperplanes or polytopes:

```
>> E = ellipsoid([-17; 0], [4 -1; -1 1]);  % define ellipsoid
>> HH = hyperplane([1 1; -1 -1; 1 -1; -1 1]', [2 2 2 2]);  % define 4 hyperplanes
>> P = hyperplane2polytope(HH) + [2; 10];  % define polytope
>> % check if ellipsoid E intersects with external approximation:
>> intersect(ct, E, 'e')

ans =

     1

>> % check if ellipsoid E intersects with internal approximation:
>> intersect(ct, E, 'i')

ans =

     0

>> % check if hyperplanes in HH intersect with internal approximation:
>> intersect(ct, HH, 'i')

ans =

     1     1     1     1

>> % check if polytope P intersects with external approximation:
>> intersect(ct, P)

ans =

     0
```

If a given set intersects with the internal approximation of the reach set, then this set intersects with the actual reach set. If the given set does not intersect with external approximation, this set does not intersect the actual reach set. There are situations, however, when the given set intersects with the external approximation but does not intersect with the internal one. In our example above, ellipsoid `E` is such a case: the quality of the approximation does not allow us to determine whether or not `E` intersects with the actual reach set. To improve the quality of approximation, `refine` function should be used:

```
>> L1 = [1; -1];  % define new directions, in this case one, but could be more
>> rs = refine(rs, L1);  % compute approximations for the new directions
>> ct = cut(rs, 5);  % snap shot of the reach set at time t = 5
>> intersect(ct, E, 'i')  % check if E intersects the internal approximation

ans =

     1
```

Now we are sure that ellipsoid `E` intersects with the actual reach set. Recall that when we computed reach set `rs` the first time, we did it with the option `save_all` set to 1. This option indicated to the `reach` constructor that it should save all intermediate calculations of data in the `reach` object `rs`. These data include evaluations of matrices A, B, G at specific time values (in case these matrices depend on time) together with the control and disturbance bounds, the state transition matrix and its inverse evaluated at these time values. By default, `save_all` option is set to 0, and all these intermediate data are not retained, which significantly reduces the memory used by the `reach` object `rs`. However, to use the `refine` function, the reach set object must contain all calculated data, otherwise, an error is returned.

Having a reach set object resulting from the `reach`, `cut` or `refine` operations, we can obtain the trajectory of the center of the reach set and the good curves along which the actual reach set is touched by its ellipsoidal approximations:

```
>> [ctr, tt] = get_center(rs);  % trajectory of the center
>> gc = get_goodcurves(rs)  % get good curves

gc =
    [2x200 double]    [2x200 double]    [2x200 double]    [2x200 double]    [2x200 double]
```

Variable `ctr` here is a matrix whose columns are the points ofthe reach set center trajectory evaluated at time values returned in the array `tt`. Variable `gc` contains 4 matrices each of which corresponds to a good curve (columns of such matrix are points of the good curve evaluated at time values in `tt`). The analytic expression for the control driving the system along a good curve is given by formula (3.18).

We computed the reach set up to time 10. It is possible to continue the reach set computation for a longer time horizon using the reach set data at time 10 as initial condition. It is also possible that the dynamics and inputs of the system change at certain time, and from that point on the system evolves according to the new system of differential equations. For example, starting at time 10, our reach set may evolve in time according to the time-variant system `sys_t` defined above. Switched systems are a special case of this situation. To compute the further evolution in time of the existing reach set, function `evolve` should be used:

```
>> rs2 = evolve(rs, 15);  % reach set from time 10 to 15 with the same dynamics
>> rs2 = evolve(rs, 15, sys_t);  % reach set from time 10 to 15 with new dynamics
>>
>> % not only the dynamics, but the inputs can change as well,
>> % from time 15 to 20 disturbance is added to the system:
>> rs3 = evolve(rs2, 20 sys_d);  % sys_d - system with disturbance defined above
```

Function `evolve` can be viewed as an implementation of the semigroup property.

To compute the backward reach set for some specified target set, we declare the time interval so that the terminating time comes first:

```
>> Y = ellipsoid([8; 2], [4 1; 1 2]);  % target set in the form of ellipsoid
>> Tb = [10 5];  % backward time interval
>> brs = reach(sys, Y, L, Tb, options);  % backward reach set
>> brs = refine(brs, L1);  % refine the approximation
>> brs2 = evolve(brs, 0);  % further evolution in backward time from 5 to 0
```

Reach set and backward reach set computation for discrete-time systems and manipulations with the resulting reach set object are performed using the same functions as for continuous-time systems:

```
>> T = [0 100];  % represents 100 time steps from 1 to 100
>> dtrs = reach(dtsys, X0, L, T);  % reach set for 100 time steps
>> dtrs2 = evolve(dtrs, 200);  % compute next 100 time steps
>>
>> Tb = [50 0];  % backward time interval
>> dtbrs = reach(dtsys, Y, L, Tb, options)  % backward reach set
>> dtbrs = refine(dtbrs, L1);  % refine the approximation
>> [EA, tt] = get_ea(dtbrs);  % get external approximating ellipsoids and time values
>> IA = get_ia(dtbrs)  % get internal approximating ellipsoids

IA =
5x51 array of ellipsoids.
```

Number of columns in the ellipsoidal arrays `EA` and `IA` is 51 because the backward reach set is computed for 50 time steps, and the first column of these arrays contains 5 ellipsoids `Y` - the terminating condition.

When dealing with discrete-time systems, all functions that accept time or time interval as an input parameter, round the time values and treat them as integers.

5.3 Visualization

Ellipsoidal Toolbox has several plotting routines:

- `ellipsoid/plot` - plots one or more ellipsoids, or arrays of ellipsoids, defined in $\mathbf{R}$, $\mathbf{R}^2$ or $\mathbf{R}^3$.
- `ellipsoid/minkdiff` - plots geometric difference (if it is not an empty set) of two ellipsoids defined in $\mathbf{R}$, $\mathbf{R}^2$ or $\mathbf{R}^3$.
- `ellipsoid/minksum` - plots geometric sum of finite number of ellipsoids defined in $\mathbf{R}$, $\mathbf{R}^2$ or $\mathbf{R}^3$.
- `hyperplane/plot` - plots one or more hyperplanes, or arrays of hyperplanes, defined in $\mathbf{R}^2$ or $\mathbf{R}^3$.
- `reach/plot_ea` - plots external approximation of the reach set whose dimension is 2 or 3.
- `reach/plot_ia` - plots internal approximation of the reach set whose dimension is 2 or 3.

All these functions allow the user to specify the color of the plotted objects, line width for 1D and 2D plots, and transparency level of the 3D objects. Hyperplanes are displayed as line segments in 2D and square facets in 3D. In the `hyperplane/plot` method it is possible to specify the center of the line segment or facet and its size.

Ellipsoids of dimensions higher than three must be projected onto a two- or three-dimensional subspace before being plotted. This is done by means of `projection` function:

```
>> % create two 4-dimensional ellipsoids:
>> E1 = ellipsoid([14 -4 2 -5; -4 6 0 1; 2 0 6 -1; -5 1 -1 2]);
>> E2 = inv(E1);
>>
>> % specify 3-dimensional subspace by its basis:
>> BB = [1 0 0 0; 0 0 1 0; 0 1 0 1]';  % columns of BB must be orthogonal
>>
>> % get 3-dimensional projections of E1 and E2:
>> PP = projection([E1 E2], B)  % array PP contains projections of E1 and E2

PP =
1x2 array of ellipsoids.

>> plot(PP);  % plot ellipsoids in PP
```

Since the operation of projection is linear, the projection of the geometric sum of ellipsoids equals the geometric sum of the projected ellipsoids. The same is true for the geometric difference of two ellipsoids.

Function `projection` exists also for the `reach` objects:

```
>> A = [0 1 0 0; -1 0 1 0; 0 0 0 1; 0 0 -1 0];
>> B = [0; 0; 0; 1];
>> U = ellipsoid(1);
>> sys = linsys(A, B, U);  % 4-dimensional system
>> L  = [1 1 0 1; 0 -1 1 0; -1 1 1 1; 0 0 -1 1]'; % matrix of directions
>> rs = reach(sys, ell_unitball(4), L, 5);  % reach set from time 0 to 5
>> BB = [1 0 0 1; 0 1 1 0]';  % basis of 2-dimensional subspace
>> ps = projection(rs, BB);  % project reach set rs onto basis BB
>> plot_ea(ps);  % plot external approximation
>> hold on;
>> plot_ia(ps);  % plot internal approximation
```

The quality of the ellipsoid and reach set plots is controlled by the parameters `plot2d_grid` and `plot3d_grid` of the global `ellOptions` structure (see chapter 6).

Chapter 6

Structures and Objects

6.1 ellOptions

Functions of the *Ellipsoidal Toolbox* can be called with user-specified values of certain global parameters. These parameters are stored in the global structure `ellOptions`, which is kept in the MATLAB workspace as global variable. This structure is initialized with default values of parameters upon the first call to almost any function of the *ET*. Before execution, the *ET* routine checks if global structure `ellOptions` already exists, and if not, it calls the function `ellipsoids_init` that performs the initialization.

Here we list the fields of `ellOptions` structure along with their default values.

- `version = '1.02'` - current version of *ET*. A this time this parameter is harmless in the sense that it is not used by any of the routines. In the future, however, it may be used for version compatibility.
- `verbose = 1` - if set to 0, makes all the calls to *ET* routines silent, and no information except errors is displayed. Otherwise, it is assumed to be 1. Currently, there are no other levels of verbosity.
- `abs_tol = 1e-9` - absolute tolerance.
- `rel_tol = 1e-7` - relative tolerance.
- `time_grid = 200` - density of the time grid for the continuous time reach set computation. This parameter directly affects the number of ellipsoids to be stored in the `reach` object.
- `ode_solver = 1` - specifies the ODE solver for continuous time reach set computation: 1 = 'RK45', 2 = 'RK23', 3 = 'Adams'.
- `norm_control = 'on'` - switches on and off the norm control in the ODE solver. When turned on, it slows down the computation, but improves the accuracy.
- `ode_solver_options = 0` - when set to 0, calls the ODE solver without any additional options like norm control. It makes the computation faster but less accurate. Otherwise, it is assumed to be 1, and only in this case the previous option makes a difference.

- `nlcp_solver = 0` - specifies which gradient method implementation to use. If set to 0, it is `ell_nlfnlc`, which comes with *ET*; if set to 1, it is `fmincon`, which is part of MATLAB Optimization Toolbox.
- `plot2d_grid = 200` - specifies number of points used to plot a 2D ellipsoid. This parameter also affects the quality of 2D reach tube and reach set plots.
- `plot3d_grid = 200` - the number of points used to plot a 3D ellipsoid is calculated as $\frac{\texttt{plot3d\_grid}^2}{2}$. This parameter also affects the quality of 3D reach set plots.
- `sdpsettings` - the settings used by YALMIP optimization toolbox.

These parameters can be modified by editing *ellipsoids/ellipsoids_init.m* file. After you finished editing, for changes to take effect, type

```
>> clear global ellOptions;
```

If you would like to change certain parameters temporarily, without modifying the *ellipsoids_init.m* file, say, turn the verbosity off, you should type

```
>> global ellOptions;
>> ellOptions.verbose = 0;
```

and proceed with your work. Before you modify `ellOptions` structure this way however, make sure it is initialized.

6.2 ellipsoid

The main object of the *Ellipsoidal Toolbox*, `ellipsoid`, is very simple. In accordance with definition 2.1.3, it contains two fields:

- `center` - n-dimensional vector specifying the center of the ellipsoid;
- `shape` - $(n \times n)$-dimensional symmetric positive semidefinite matrix.

These fields cannot be accessed by the user directly. Their values can be obtained through `ellipsoid/parameters` function, but they cannot be modified except by some allowed operation with the `ellipsoid` object. For the list of `ellipsoid` methods, see section 8.1.

6.3 hyperplane

According to definition 2.1.7, the hyperplane object contains two fields:

- `normal` - n-dimensional vector specifying the normal to the hyperplane (c in 2.9);
- `shift` - the scalar (γ in 2.9).

These fields cannot be accessed by the user directly. Their values can be obtained through `hyperplane/parameters` function, but they cannot be modified other than by some allowed operation with the `hyperplane` object. For the list of `hyperplane` methods, see section 8.2.

In some *ET* functions, for example, in `ellipsoid/intersection_ea` and `ellipsoid/intersection_ia`, the hyperplane specifies the halfspace. It is assumed that the halfspace is

$$\{x \in \mathbf{R}^n \mid \langle \texttt{normal}, x\rangle \leq \texttt{shift}\}.$$

6.4 linsys

Ellipsoidal Toolbox supports both types of linear (affine) dynamical systems: continuous-time,

$$\begin{aligned} \dot{x}(t) &= A(t)x(t) + B(t)u(t) + G(t)v(t), \\ y(t) &= C(t)x(t) + D(t)u(t) + w(t); \end{aligned}$$

and discrete-time,

$$\begin{aligned} x[k+1] &= A[k]x[k] + B[k]u[k] + G[k]v[k], \\ y[k] &= C[k]x[k] + D[k]u[k] + w[k]. \end{aligned}$$

Both can be time-invariant (have constant matrices A, B, G, C, D) or time-variant.
The `linsys` object contains the fields:

- `A` - $(n \times n)$-dimensional matrix A of type `double` if constant, or `cell` to symbolically represent $A(t)$ or $A[k]$.
- `B` - $(n \times m)$-dimensional matrix B, `double` if constant, `cell` if symbolic.
- `control` - ellipsoidal bounds on control u, either an `ellipsoid` object of dimension m, or structure `U` with fields `U.center` and `U.shape` to represent the ellipsoid that depends on time. For example,
```
>> U.center = [0; 1];
>> U.shape = {'4' 'cos(t)'; 'cos(t)' '1'};
```
defines ellipsoid $\mathcal{E}(p, P(t))$ with $p = \left[\begin{array}{c} 0 \\ 1 \end{array}\right]$ and $P(t) = \left[\begin{array}{cc} 4 & \cos(t) \\ \cos(t) & 1 \end{array}\right]$.
- `G` - $(n \times d)$-dimensional matrix G of type `double` if constant, or `cell` if symbolic. Can be empty if the system has no disturbance or affine term.
- `disturbance` - ellipsoidal bounds on disturbance v, either an `ellipsoid` object of dimension d, or structure `V` with fields `V.center` and `V.shape` for symbolic representation of ellipsoid, similar to the `control` field. This field can be also a single d-dimensional vector - constant or symbolic - to represent an affine term.
- `C` - $(r \times n)$-dimensional matrix C of type `double` if constant, or `cell` if symbolic.
- `D` - $(r \times m)$-dimensional matrix D of type `double` if constant, or `cell` if symbolic. Can be empty.

- `noise` - ellipsoidal bounds on the noise w, either an `ellipsoid` object of dimension r, or structure `W` with fields `W.center` and `W.shape` for symbolic representation of ellipsoid, similar to the `control` and `disturbance` fields.
  This field can be also a single r-dimensional vector, constant or symbolic, to represent an affine term.
- `lti` - 1 if the system is time-invariant, 0 - otherwise.
- `dt` - 1 if the system is discrete-time, 0 - otherwise.
- `constantbounds` - indicates if the bounds on control, disturbance and noise are constant.

The fields of `linsys` object can be accessed but cannot be modified directly by the user. The only way to modify these fields is through `linsys/linsys` constructor. For the list of `linsys` methods, see section 8.3.

6.5 reach

The `reach` object represents the reach (or backward reach) set of an affine system. It contains the fields:

- `system` - the description of the system, for which the reach set is computed, in the form of `linsys` object.
- `t0` - initial time value.
- `X0` - the set of initial (or terminating, in case of backward reachability) conditions in the form of `ellipsoid` object.
- `initial_directions` - matrix whose columns represent the values of direction vector l (see section 3.2), for which the ellipsoidal approximations of the reach set are computed.
- `time_values` - time interval, for which the reach set is computed, is split into the number of segments specified by the global `ellOptions.time_grid` parameter. This field contains the values of the time grid. If the last value of this array is less than the value of `t0`, then the reach set is in fact backward reach set.
- `center_values` - matrix whose columns are values of the reach set center trajectory evaluated at times specified by `time_values`.
- `l_values` - array of directions vectors evaluated at times specified by `time_values`.
- `ea_values` - array of the shape matrices of the external ellipsoids evaluated at times specified by `time_values`.
- `ia_values` - array of the shape matrices of the internal ellipsoids evaluated at times specified by `time_values`.
- `projection_basis` - if the reach set is projected onto the given orthonormal basis, the columns of this field are the basis vectors, otherwise, this field is empty.

- `calc_data` - this field is empty unless the reach set is computed with the `save_all` option set to 1. This field then contains the intermediate calculation data, which can be used for the approximation refinement. For more detail, see description of the function `refine` in section 8.4.

These fields can be accessed and modified only through `reach` methods. For the list of `reach` methods, see section 8.4.

Chapter 7

Examples

7.1 Ellipsoids vs. Polytopes

Depending on the particular dynamical system, certain methods of reach set computation may be more suitable than others. Even for a simple 2-dimensional discrete-time linear time-invariant system, application of ellipsoidal methods may be more effective than using polytopes.

Consider the system from chapter 1:

$$\left[\begin{array}{c} x_1[k+1] \\ x_2[k+1] \end{array}\right] = \left[\begin{array}{cc} \cos(1) & \sin(1) \\ -\sin(1) & \cos(1) \end{array}\right] \left[\begin{array}{c} x_1[k] \\ x_2[k] \end{array}\right] + \left[\begin{array}{c} u_1[k] \\ u_2[k] \end{array}\right], \quad x[0] \in \mathcal{X}_0, \quad u[k] \in U, \quad k \geq 0,$$

where $\mathcal{X}_0$ is the set of initial conditions, and U is the control set.

Let $\mathcal{X}_0$ and U be unit boxes in $\mathbf{R}^2$, and compute the reach set using the polytope method implemented in MPT ([13]). With every time step the number of vertices of the reach set polytope increases by 4. The complexity of the convex hull computation increases exponentially with number of vertices. In figure 7.1, the time required to compute the reach set for different time steps using polytopes is shown in red.

To compute the reach set of the system using *Ellipsoidal Toolbox*, we assume $\mathcal{X}_0$ and U to be unit balls in $\mathbf{R}^2$, fix any number of initial direction values that corresponds to the number of ellipsoidal approximations, and obtain external and internal ellipsoidal approximations of the reach set:

```
>> A = [cos(1) sin(1); -sin(1) cos(1)];
>> U = ell_unitball(2);  % control bounds
>>
>> % define linear discrete-time system:
>> lsys = linsys(A, eye(2), U, [], [], [], [], 'd');
>>
>> X0 = ell_unitball(2);  % set of initial conditions
>> L0 = [cos(0:0.1:pi); sin(0:0.1:pi)];  % 32 initial directions
>> N  = 100;  % number of time steps
>>
>> % compute the reach set:
>> rs = reach(lsys, X0, L0, N);
```

In figure 7.1, the time required to compute both external and internal ellipsoidal approximations, with 32 ellipsoids each, for different number of time steps is shown in blue.

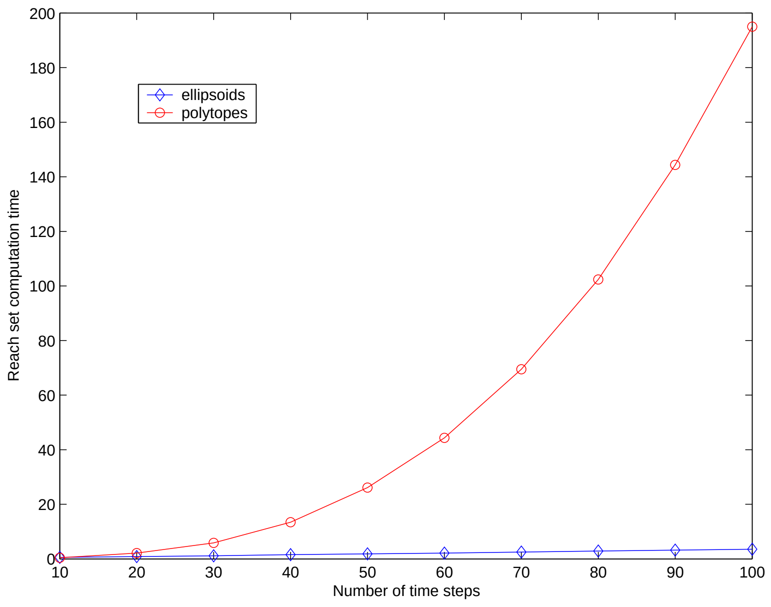

Figure 7.1: Reach set computation performance comparison: ellipsoids (blue) vs. polytopes (red).

Figure 7.1 illustrates the fact that the complexity of polytope method grows exponentially with number of time steps, whereas the complexity of ellipsoidal method grows linearly.

7.2 System with Disturbance

The mechanical system presented in figure 7.2, is described by the following system of equations:

$$m_1\ddot{x}_1 + (k_1 + k_2)x_1 - k_2x_2 = u_1, \tag{7.1}$$
$$m_2\ddot{x}_2 - k_2x_1 + (k_1 + k_2)x_2 = u_2. \tag{7.2}$$

Here u_1 and u_2 are the forces applied to masses m_1 and m_2, and we shall assume $[u_1 \;\; u_2]^T \in \mathcal{E}(0, I)$. The initial conditions can be taken as $x_1(0) = 0$, $x_2(0) = 2$. Defining $x_3 = \dot{x}_1$ and $x_4 = \dot{x}_2$, we can rewrite (7.1-7.2) as a linear system in standard form:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{k_1+k_2}{m_1} & \frac{k_2}{m_1} & 0 & 0 \\ \frac{k_2}{m_2} & -\frac{k_1+k_2}{m_2} & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{m_1} & 0 \\ 0 & \frac{1}{m_2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{7.3}$$

Now we can compute the reach set of system (7.1-7.2) for given time by computing the reach set of the linear system (7.3) and taking its projection onto (x_1, x_2) subspace.

```
>> % specify parameters k1, k2 and masses m1, m2:
```

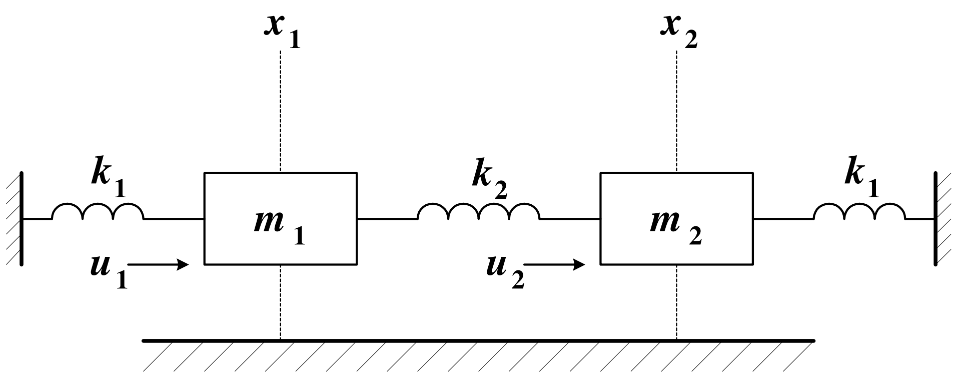

Figure 7.2: Spring-mass system.

```
>> k1 = 24;  k2 = 32;
>> m1 = 1.5; m2 = 1;
>>
>> % define matrices A, B, and control bounds U:
>> A = [0 0 1 0; 0 0 0 1; -(k1+k2)/m1 k2/m1 0 0; k2/m2 -(k1+k2)/m2 0 0];
>> B = [0 0; 0 0; 1/m1 0; 0 1/m2];
>> U = ell_unitball(2);
>>
>> lsys = linsys(A, B, U);  % linear system
>> T = [0 4];  % time interval
>>
>> % initial conditions:
>> X0 = [0 2 0 0]' + ellipsoid([0.01 0 0 0; 0 0.01 0 0; 0 0 eps 0; 0 0 0 eps]);
>>
>> % initial directions (some random vectors in R^4):
>> L0 = [1 0 1 0; 1 -1 0 0; 0 -1 0 1; 1 1 -1 1 -1 1 1 0; -2 0 1 1]';
>>
>> rs = reach(lsys, X0, L0, T);  % reach set
>>
>> BB = [1 0 0 0; 0 1 0 0]';  % orthogonal basis of (x1, x2) subspace
>> ps = projection(rs, BB);  % reach set projection
>>
>> % plot projection of reach set external approximation:
>> subplot(2, 2, 1);
>> plot_ea(ps, 'g');  % plot the whole reach tube
>> subplot(2, 2, 2);
>> plot_ea(cut(ps, 4), 'g');  % plot reach set approximation at time t = 4
```

Figure 7.3(a) shows the reach set of the system (7.1-7.2) evolving in time from $t = 0$ to $t = 4$. Figure 7.3(b) presents a snapshot of this reach set at time $t = 4$.

So far we considered an ideal system without any disturbance, such as friction. We introduce disturbance to (7.1-7.2) by adding extra terms, v_1 and v_2,

$$m_1\ddot{x}_1 + (k_1 + k_2)x_1 - k_2x_2 = u_1 + v_1, \tag{7.4}$$

$$m_2\ddot{x}_2 - k_2x_1 + (k_1 + k_2)x_2 = u_2 + v_2, \tag{7.5}$$

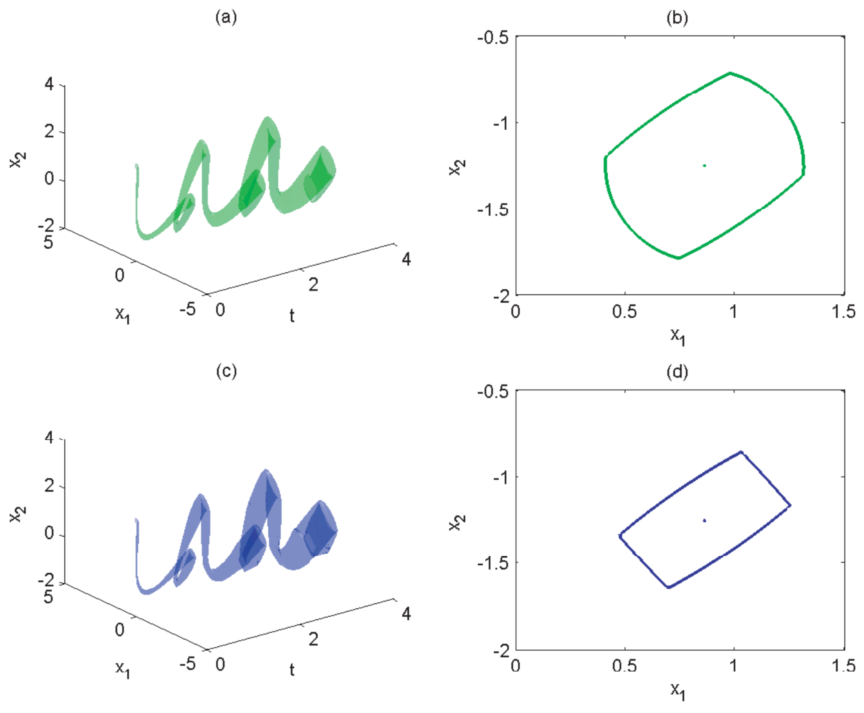

Figure 7.3: Spring-mass system without disturbance: (a) reach tube for time $t \in [0, 4]$; (b) reach set at time $t = 4$. Spring-mass system with disturbance: (c) reach tube for time $t \in [0, 4]$; (d) reach set at time $t = 4$.

which results in equation (7.3) getting an extra term

$$\left[\begin{array}{cc} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{array}\right]\left[\begin{array}{c} v_1 \\ v_2 \end{array}\right].$$

Assuming that $[v_1 \;\; v_2]^T$ is unknown but bounded by ellipsoid $\mathcal{E}(0, \frac{1}{4}I)$, we can compute the closed-loop reach set of the system with disturbance.

```
>> % define disturbance:
>> G = [0 0; 0 0; 1 0; 0 1];
>> V = 0.5*ell_unitball(2);
>>
>> lsysd = linsys(A, B, U, G, V);  % linear system with disturbance
>>
>> rsd = reach(lsysd, X0, L0, T);  % reach set
>> psd = projection(rsd, BB);  % reach set projection onto (x1, x2)
>>
>> % plot projection of reach set external approximation:
>> subplot(2, 2, 3);
>> plot_ea(ps);  % plot the whole reach tube
>> subplot(2, 2, 4);
>> plot_ea(cut(ps, 4));  % plot reach set approximation at time t = 4
```

Figure 7.3(c) shows the reach set of the system (7.4-7.5) evolving in time from $t = 0$ to $t = 4$. Figure 7.3(d) presents a snapshot of this reach set at time $t = 4$.

7.3 Switched System

By *switched systems* we mean systems whose dynamics changes at known times. Consider the RLC circuit shown in figure 7.4. It has two inputs—the voltage (v) and current (i) sources. Define

- x_1 - voltage across capacitor C_1, so $C_1\dot{x}_1$ is the corresponding current;
- x_2 - voltage across capacitor C_2, so the corresponding current is $C_2\dot{x}_2$.
- x_3 - current through the inductor L, so the voltage across the inductor is $L\dot{x}_3$.

Applying Kirchoff current and voltage laws we arrive at the linear system,

$$\left[\begin{array}{c} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{array}\right] = \left[\begin{array}{ccc} -\frac{1}{R_1C_1} & 0 & -\frac{1}{C_1} \\ 0 & 0 & \frac{1}{C_2} \\ \frac{1}{L} & -\frac{1}{L} & -\frac{R_2}{L} \end{array}\right]\left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] + \left[\begin{array}{cc} \frac{1}{R_1C_1} & \frac{1}{C_1} \\ 0 & 0 \\ 0 & 0 \end{array}\right]\left[\begin{array}{c} v \\ i \end{array}\right]. \tag{7.6}$$

The parameters R_1, R_2, C_1, C_2 and L, as well as the inputs, may depend on time. Suppose, for time $0 \leq t < 2$, $R_1 = 2$ Ohm, $R_2 = 1$ Ohm, $C_1 = 3$ F, $C_2 = 7$ F, $L = 2$ H, both inputs, v and i are

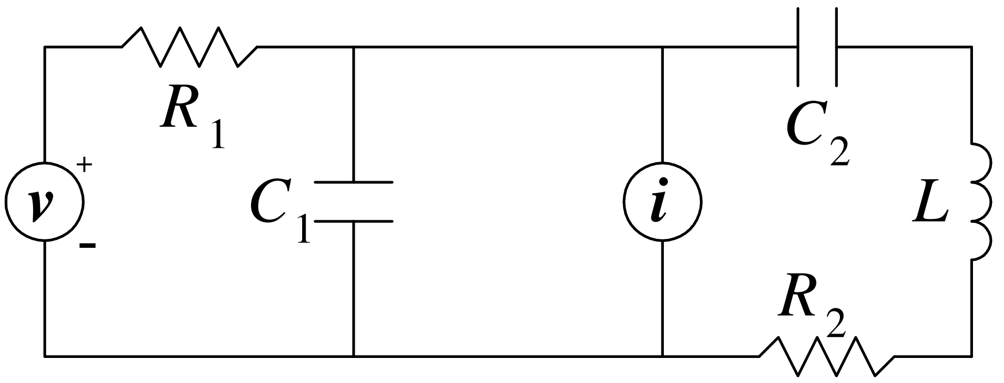

Figure 7.4: RLC circuit with two inputs.

present and bounded by ellipsoid $\mathcal{E}(0, I)$; and for time $t \geq 2$, $R_1 = R_2 = 2$ Ohm, $C_1 = C_2 = 3$ F, $L = 6$ H, the current source is turned off, and $|v| \leq 1$. Then, system (7.6) can be rewritten as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{cases} \begin{bmatrix} -\frac{1}{6} & 0 & -\frac{1}{3} \\ 0 & 0 & \frac{1}{7} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} \frac{1}{6} & \frac{1}{3} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v \\ i \end{bmatrix}, & 0 \leq t < 2, \\ \begin{bmatrix} -\frac{1}{6} & 0 & -\frac{1}{3} \\ 0 & 0 & \frac{1}{3} \\ \frac{1}{6} & -\frac{1}{6} & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} \frac{1}{6} \\ 0 \\ 0 \end{bmatrix} v, & 2 \leq t. \end{cases} \tag{7.7}$$

We can compute the reach set of (7.7) for some time $t > 2$, say, $t = 3$.

```
>> % define system 1:
>> A1 = [-1/6 0 -1/3; 0 0 1/7; 1/2 -1/2 -1/2];
>> B1 = [1/6 1/3; 0 0; 0 0];
>> U1 = ellipsoid(eye(2));
>> s1 = linsys(A1, B1, U1);
>>
>> % define system 2:
>> A2 = [-1/6 0 -1/3; 0 0 1/3; 1/6 -1/6 -1/3];
>> B2 = [1/6; 0; 0];
>> U2 = ellipsoid(1);
>> s2 = linsys(A2, B2, U2);
>>
>> X0 = ellipsoid(0.01*eye(3));  % set of initial states
>> L0 = eye(3);  % 3 initial directions
>> TS = 2;  % time of switch
>> T = 3;  % terminating time
>>
>> % compute the reach set:
>> rs1 = reach(s1, X0, L0, TS);  % reach set of the first system
>> % computation of the second reach set starts
>> % where the first left off
>> rs2 = evolve(rs1, T, s2);
>>
>> % obtain projections onto (x1, x2) subspace:
```

```
>> BB = [1 0 0; 0 1 0]';  % (x1, x2) subspace basis
>> ps1 = projection(rs1, BB);
>> ps2 = projection(rs2, BB);
>>
>> % plot the results:
>> subplot(2, 2, 1);
>> plot_ea(ps1, 'r');  % external apprx. of reach set 1 (red)
>> hold on;
>> plot_ia(ps1, 'g');  % internal apprx. of reach set 1 (green)
>> plot_ea(ps2, 'y');  % external apprx. of reach set 2 (yellow)
>> plot_ia(ps2, 'b');  % internal apprx. of reach set 2 (blue)
>>
>> % plot the 3-dimensional reach set at time t = 3:
>> subplot(2, 2, 2);
>> plot_ea(cut(rs2, 3), 'y');
>> hold on;
>> plot_ia(cut(rs2, 3), 'b');
```

Figure 7.5(a) shows how the reach set projection onto (x_1, x_2) of system (7.7) evolves in time from $t = 0$ to $t = 3$. The external reach set approximation for the first dynamics is in red, the internal approximation is in green. The dynamics switches at $t = 2$. The external reach set approximation for the second dynamics is in yellow, its internal approximation is in blue. The full three-dimensional external (yellow) and internal (blue) approximations of the reach set are shown in figure 7.5(b).

To find out where the system should start at time $t = 0$ in order to reach a neighborhood `M` of the origin at time $t = 3$, we compute the backward reach set from $t = 3$ to $t = 0$.

```
>> M  = ellipsoid(0.01*eye(3));  % terminating set
>> TT = 3;  % terminating time
>>
>> % compute backward reach set:
>> % compute the reach set:
>> brs2 = reach(s2, M, L0, [TT TS]);  % second system comes first
>> brs1 = evolve(brs2, 0, s1);  % then the first system
>>
>> % obtain projections onto (x1, x2) subspace:
>> bps1 = projection(brs1, BB);
>> bps2 = projection(brs2, BB);
>>
>> % plot the results:
>> subplot(2, 2, 3);
>> plot_ea(bps1, 'r');  % external apprx. of backward reach set 1 (red)
>> hold on;
>> plot_ia(bps1, 'g');  % internal apprx. of backward reach set 1 (green)
>> plot_ea(bps2, 'y');  % external apprx. of backward reach set 2 (yellow)
>> plot_ia(bps2, 'b');  % internal apprx. of backward reach set 2 (blue)
>>
>> % plot the 3-dimensional backward reach set at time t = 0:
>> subplot(2, 2, 2);
>> plot_ea(cut(brs1, 0), 'r');
```

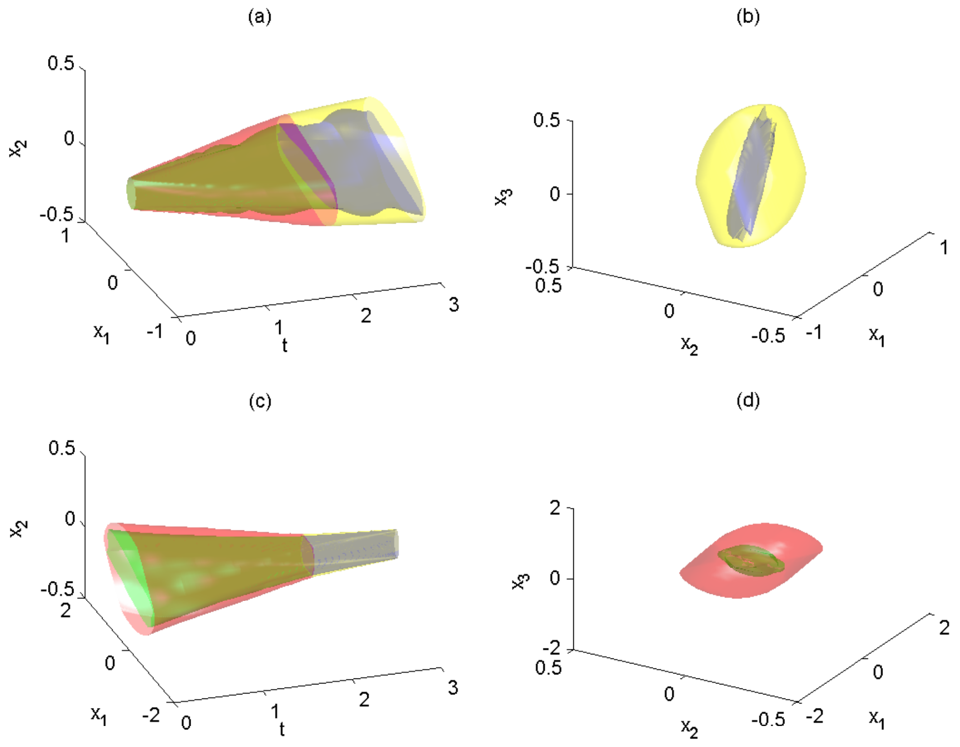

Figure 7.5: Forward and backward reach sets of the switched system (external and internal approximations). The dynamics switches at $t = 2$.
(a) Forward reach set for the time interval $0 \leq t \leq 3$ projected onto (x_1, x_2) subspace.
(b) Forward reach set at $t = 3$ in $\mathbf{R}^3$.
(c) Backward reach set evolving from $t = 3$ to $t = 0$ projected onto (x_1, x_2) subspace.
(d) Backward reach set at $t = 0$ in $\mathbf{R}^3$.

```
>> hold on;
>> plot_ia(cut(brs1, 0), 'g');
```

Figure 7.5(c) presents the evolution of the reach set projection onto (x_1, x_2) in backward time. Again, external and internal approximations corresponding to the first dynamics are shown in red and green, and to the second dynamics in yellow and blue. The full dimensional backward reach set external and internal approximations of system (7.7) at time $t = 0$ is shown in figure 7.5(d).

7.4 Hybrid System

There is no explicit implementation of the reachability analysis for hybrid systems in the *Ellipsoidal Toolbox*. Nonetheless, the operations of intersection available in the toolbox allow us to work with certain class of hybrid systems, namely, hybrid systems with affine continuous dynamics whose guards are ellipsoids, hyperplanes, halfspaces or polytopes.

We consider the *switching-mode model* of highway traffic presented in [36]. The highway segment is divided into N cells as shown in figure 7.6. In this particular case, $N = 4$. The traffic density in cell i is x_i vehicles per mile, $i = 1, 2, 3, 4$.

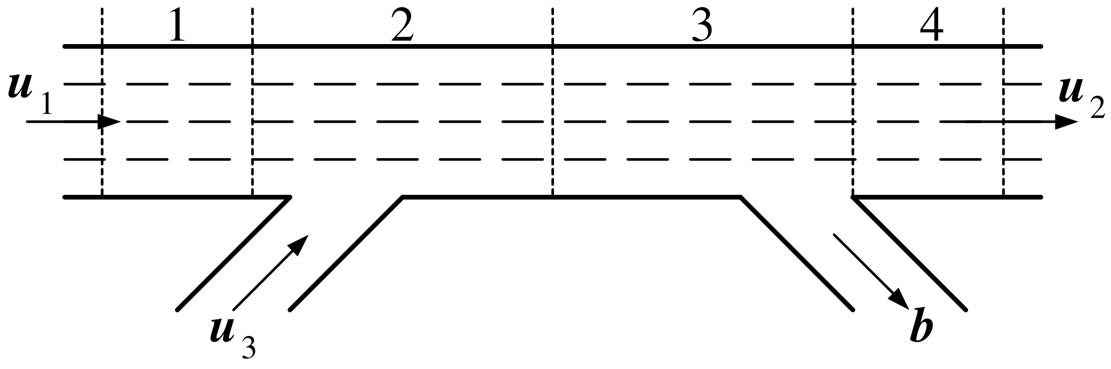

Figure 7.6: Highway model. Adapted from [36].

Define

- v_i - average speed in mph, in the i-th cell, $i = 1, 2, 3, 4$;
- w_i - backward congestion wave propagation speed in mph, in the i-th highway cell, $i = 1, 2, 3, 4$;
- x_{Mi} - maximum allowed density in the i-th cell; when this velue is reached, there is a traffic jam, $i = 1, 2, 3, 4$;
- d_i - length of i-th cell in miles, $i = 1, 2, 3, 4$;
- T_s - sampling time in hours;
- b - split ratio for the off-ramp;

- u_1 - traffic flow coming into the highway segment, in vehicles per hour (vph);
- u_2 - traffic flow coming out of the highway segment (vph);
- u_3 - on-ramp traffic flow (vph).

Highway traffic operates in two modes: *free-flow* in normal operation; and *congested* mode, when there is a jam. Traffic flow in free-flow mode is described by

$$\begin{bmatrix} x_1[t+1] \\ x_2[t+1] \\ x_3[t+1] \\ x_4[t+1] \end{bmatrix} = \begin{bmatrix} 1-\frac{v_1T_s}{d_1} & 0 & 0 & 0 \\ \frac{v_1T_s}{d_2} & 1-\frac{v_2T_s}{d_2} & 0 & 0 \\ 0 & \frac{v_2T_s}{d_3} & 1-\frac{v_3T_s}{d_3} & 0 \\ 0 & 0 & (1-b)\frac{v_3T_s}{d_4} & 1-\frac{v_4T_s}{d_4} \end{bmatrix} \begin{bmatrix} x_1[t] \\ x_2[t] \\ x_3[t] \\ x_4[t] \end{bmatrix} + \begin{bmatrix} \frac{v_1T_s}{d_1} & 0 & 0 \\ 0 & 0 & \frac{v_2T_s}{d_2} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}. \tag{7.8}$$

The equation for the congested mode is

$$\begin{bmatrix} x_1[t+1] \\ x_2[t+1] \\ x_3[t+1] \\ x_4[t+1] \end{bmatrix} = \begin{bmatrix} 1-\frac{w_1T_s}{d_1} & \frac{w_2T_s}{d_1} & 0 & 0 \\ 0 & 1-\frac{w_2T_s}{d_2} & \frac{w_3T_s}{d_2} & 0 \\ 0 & 0 & 1-\frac{w_3T_s}{d_3} & \frac{1}{1-b}\frac{w_4T_s}{d_3} \\ 0 & 0 & 0 & 1-\frac{w_4T_s}{d_4} \end{bmatrix} \begin{bmatrix} x_1[t] \\ x_2[t] \\ x_3[t] \\ x_4[t] \end{bmatrix} + \begin{bmatrix} 0 & 0 & \frac{w_1T_s}{d_1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -\frac{w_4T_s}{d_4} & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} + \begin{bmatrix} \frac{w_1T_s}{d_1} & -\frac{w_2T_s}{d_1} & 0 & 0 \\ 0 & \frac{w_2T_s}{d_2} & -\frac{w_3T_s}{d_2} & 0 \\ 0 & 0 & \frac{w_3T_s}{d_3} & -\frac{1}{1-b}\frac{w_4T_s}{d_3} \\ 0 & 0 & 0 & \frac{w_4T_s}{d_4} \end{bmatrix} \begin{bmatrix} x_{M1} \\ x_{M2} \\ x_{M3} \\ x_{M4} \end{bmatrix}. \tag{7.9}$$

The switch from the free-flow to the congested mode occurs when the density x_2 reaches x_{M2}. In other words, the hyperplane $H([0\ 1\ 0\ 0]^T, x_{M2})$ is the guard.

We indicate how to implement the reach set computation of this hybrid system. We first define the two linear systems and the guard.

```
>> % assign parameter values:
>> v1 = 65; v2 = 60; v3 = 63; v4 = 65;  % mph
>> w1 = 10; w2 = 10; w3 = 10; w4 = 10;  % mph
>> d1 = 2; d2 = 3; d3 = 4; d4 = 2;  % miles
>> Ts = 2/3600;  % sampling time in hours
>> xM1 = 200; xM2 = 200; xM3 = 200; xM4 = 200;  % vehicles per lane
>> b = 0.4;
>>
>> A1 = [(1-(v1*Ts/d1)) 0 0 0
       (v1*Ts/d2) (1-(v2*Ts/d2)) 0 0
```

```
         0 (v2*Ts/d3) (1-(v3*Ts/d3)) 0
         0 0 ((1-b)*(v3*Ts/d4)) (1-(v4*Ts/d4))];
>> B1 = [v1*Ts/d1 0 0; 0 0 v2*Ts/d2; 0 0 0; 0 0 0];
>> U1 = ellipsoid([180; 150; 50], [100 0 0; 0 100 0; 0 0 25]);
>>
>> A2 = [(1-(w1*Ts/d1)) (w2*Ts/d1) 0 0
         0 (1-(w2*Ts/d2)) (w3*Ts/d2) 0
         0 0 (1-(w3*Ts/d3)) ((1/(1-b))*(w4*Ts/d3))
         0 0 0 (1-(w4*Ts/d4))];
>> B2 = [0 0 w1*Ts/d1; 0 0 0; 0 0 0; 0 -w4*Ts/d4 0];
>> U2 = U1;
>> G2 = [(w1*Ts/d1) (-w2*Ts/d1) 0 0
         0 (w2*Ts/d2) (-w3*Ts/d2) 0
         0 0 (w3*Ts/d3) ((-1/(1-b))*(w4*Ts/d3))
         0 0 0 (w4*Ts/d4)];
>> V2 = [xM1; xM2; xM3; xM4];
>>
>> % define linear systems:
>> s1 = linsys(A1, B1, U1, [], [], [], [], 'd');  % free-flow mode
>> s2 = linsys(A2, B2, U2, G2, V2, [], [], 'd');  % congestion mode
>>
>> % define guard:
>> GRD = hyperplane([0; 1; 0; 0], xM2);
```

We assume that initially the system is in free-flow mode. Given a set of initial conditions, we compute the reach set according to dynamics (7.8) for certain number of time steps. We will consider the external approximation of the reach set by a single ellipsoid.

```
>> initial conditions:
>> X0 = [170; 180; 175; 170] + 10*ell_unitball(4);
>
>> L0 = [1; 0; 0; 0];  % single initial direction
>> N = 100;  % number of time steps
>>
>> ffrs = reach(s1, X0, L0, N);  % free-flow reach set
>> EA = get_ea(ffrs);  % 101x1 array of external ellipsoids
```

Having obtained the ellipsoidal array EA representing the reach set evolving in time, we determine the ellipsoids in the array that intersect the guard.

```
>> I = hpintersection(EA, GRD);  % some of the intersections are empty
>> D = find(~isempty(I));  % determine nonempty intersections
>> min(D)

ans =

      19

>> max(D)
```

```
ans =

        69
```

Analyzing the values in array `D`, we conclude that the free-flow reach set has nonempty intersection with hyperplane `GRD` at $t = 18$ for the first time, and at $t = 68$ for the last time. Between $t = 18$ and $t = 68$ it crosses the guard. Figure 7.7(a) shows the free-flow reach set projection onto (x_1, x_2, x_3) subspace for $t = 10$, before the guard crossing; figure 7.7(b) for $t = 50$, during the guard crossing; and figure 7.7(c) for $t = 80$, after the guard was crossed.

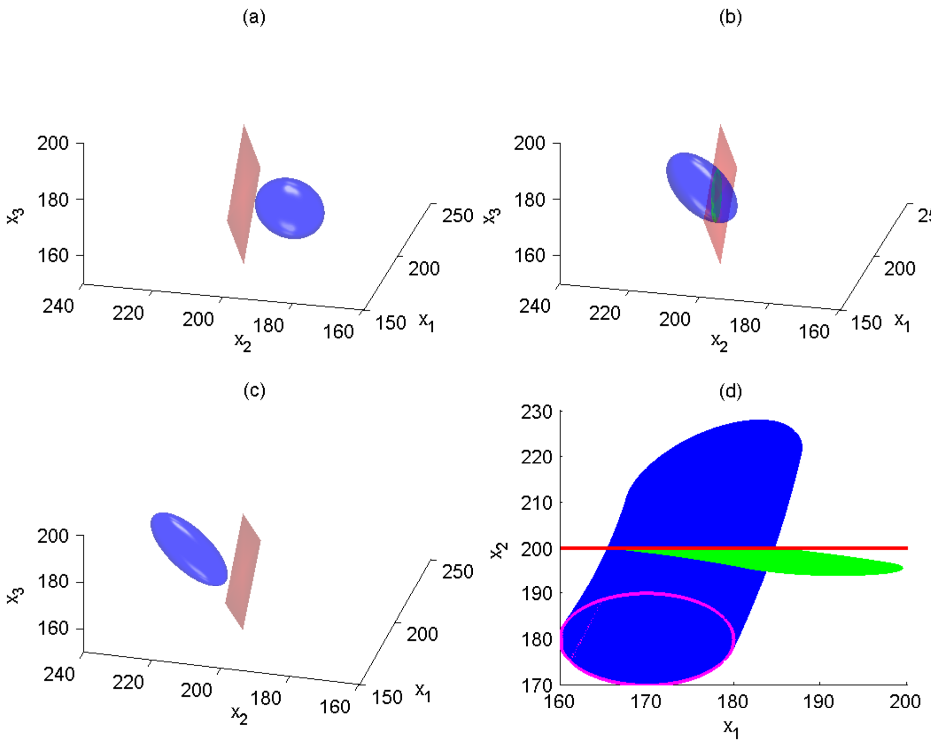

Figure 7.7: Reach set of the free-flow system is blue, reach set of the congested system is green, the guard is red.
(a) Reach set of the free-flow system at $t = 10$, before reaching the guard (projection onto (x_1, x_2, x_3)).
(b) Reach set of the free-flow system at $t = 50$, crossing the guard. (projection onto (x_1, x_2, x_3)).
(c) Reach set of the free-flow system at $t = 80$, after the guard is crossed. (projection onto (x_1, x_2, x_3)).
(d) Reach set trace from $t = 0$ to $t = 100$, free-flow system in blue, congested system in green; bounds of initial conditions are marked with magenta (projection onto (x_1, x_2)).

For each time step that the intersection of the free-flow reach set and the guard is nonempty, we establish a new initial time and a set of initial conditions for the reach set computation according to dynamics (7.9). The initial time is the array index minus one, and the set of initial conditions is the intersection of the free-flow reach set with the guard.

```
>> crs = [];
>> for i = 1:size(D, 2)
     rs = reach(s2, I(D(i)), L0, [(D(i)-1) N]);
     crs = [crs rs];
   end
```

The union of reach sets in array `crs` forms the reach set for the congested dynamics.

A summary of the reach set computation of the linear hybrid system (7.8-7.9) for $N = 100$ time steps with one guard crossing is given in figure 7.7(d), which shows the projection of the reach set trace onto (x_1, x_2) subspace. The system starts evolving in time in free-flow mode from a set of initial conditions at $t = 0$, whose boundary is shown in magenta. The free-flow reach set evolving from $t = 0$ to $t = 100$ is shown in blue. Between $t = 18$ and $t = 68$ the free-flow reach set crosses the guard. The guard is shown in red. For each nonempty intersection of the free-flow reach set and the guard, the congested mode reach set starts evolving in time until $t = 100$. All the congested mode reach sets are shown in green. Observe that in the congested mode, the density x_2 in the congested part decreases slightly, while the density x_1 upstream of the congested part increases. The blue set above the guard is not actually reached, because the state evolves according to the green region.

Chapter 8

Summary and Outlook

Although some of the operations with ellipsoids are present in the commercial Geometric Bounding Toolbox [33, 34], the ellipsoid-related functionality of that toolbox is rather limited.

Ellipsoidal Toolbox is the first free MATLAB package that implements ellipsoidal calculus and uses ellipsoidal methods for reachability analysis of continuous- and discrete-time affine systems, continuous-time linear systems with disturbances and switched systems, whose dynamics changes at known times. The reach set computation for hybrid systems whose guards are hyperplanes or polyhedra is not implemented explicitly, but the tool for such computation exists, namely, the operations of intersection of ellipsoid with hyperplane and ellipsoid with halfspace.

In future versions of the toolbox we intend to implement additional operations with ellipsoids:

- external and internal ellipsoidal approximations of the intersection of N ellipsoids (currently, these operations are implemented only for $N = 2$);
- external and internal ellipsoidal approximations for the sets

$$\begin{array}{ccc}(\mathcal{E}_1 \oplus \mathcal{E}_2)\dot{-}\mathcal{E}_3 & \text{and} & \mathcal{E}_1 \oplus (\mathcal{E}_2\dot{-}\mathcal{E}_3),\\ (\mathcal{E}_1 \oplus \mathcal{E}_2)\cap\mathcal{E}_3 & \text{and} & (\mathcal{E}_1\dot{-}\mathcal{E}_2)\cap\mathcal{E}_3;\end{array}$$

- ordering of ellipsoids: partial ordering of their shape matrices;
- ellipsoidal approximations of non-ellipsoidal convex sets, e.g. polytopes;
- reachability analysis for state-constrained and obstacle problems; and
- ellipsoidal methods for stochastic control and estimation.

Acknowledgement

The authors would like to thank Alexander B. Kurzhanski, Manfred Morari, Johan Löfberg, Michal Kvasnica and Goran Frehse for their support of this work by useful advice and encouragement.

Bibliography

[1] Ellipsoidal Toolbox homepage: *www.eecs.berkeley.edu/~akurzhan/ellipsoids*

[2] A.B.Kurzhanski, I.Vályi (1997). *Ellipsoidal Calculus for Estimation and Control.* Birkhäuser, Boston, ser. SCFA.

[3] A.B.Kurzhanski, P.Varaiya (2000). *On ellipsoidal techniques for reachability analysis.* In: *Optimization Methods and Software*, Vol.17, pp.177-237, Taylor & Francis.

[4] A.B.Kurzhanski, P.Varaiya (2002). *On reachability under uncertainty*, In *SIAM Journal on Control and Optimization*, Vol.41(1), pp.181-216.

[5] A.B.Kurzhanski, P.Varaiya (2001). *Reachability analysis for uncertain systems - the ellipsoidal technique.* In: *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications and Algorithms*, Vol.9, pp.347-367.

[6] A.A.Kurzhanskiy, P.Varaiya (2005). *Ellipsoidal Techniques for Reachability Analysis of Discrete-Time Linear Systems.* Submitted to IEEE Transactions on Automatic Control.

[7] P.Varaiya (1998). *Reach set computation using optimal control.* In: *Proc. of KIT Workshop on Verification on Hybrid Systems.* Verimag, Grenoble.

[8] P.Varaiya. *Lecture notes on optimization.* (*www.eecs.berkeley.edu/~varaiya/papers_ps.dir/NOO.pdf*).

[9] A.Yu.Vazhentsev (1999). *On Internal Ellipsoidal Approximations for Problems of Control Synthesis with Bounded Coordinates.* In: *Izvestia Rossiiskoi Akademii Nauk. Teoriya i Systemy Upravleniya.*

[10] L.Ros, A.Sabater, F.Thomas (2002). *An Ellipsoidal Calculus Based on Propagation and Fusion.* In: *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, Vol.32, N.4.

[11] Robust Ellipsoidal Calculus homepage: *www-iri.upc.es/people/ros/ellipsoids.html*

[12] M.Kvasnica, P.Grieder, M.Baotić, M.Morari (2004). *Multi-Parametric Toolbox (MPT).* In: R.Alur and G.J.Pappas, editors, *Hybrid Systems: Computation and Control*, Vol.2993 of *Lecture Notes in Computer Science*, pp.448-462. Springer-Verlag.

[13] Multi-Parametric Toolbox homepage: *control.ee.ethz.ch/~mpt*

[14] J.Löfberg (2004). *A Toolbox for Modeling and Optimization in MATLAB.* In: Proceedings of CACSD Conference, Taipei, Taiwan.

[15] YALMIP homepage: *control.ee.ethz.ch/~joloef/yalmip.php.*

[16] J.F.Sturm (1999). *Using SeDuMi 1.02, A MATLAB Toolbox for Optimization over Symmetric Cones.* In: *Optimization Methods and Software*, Vol.11-12, pp.625-653, Taylor & Francis.

[17] SeDuMi homepage: *sedumi.mcmaster.ca*

[18] I.Mitchell, C.Tomlin (2000). *Level set methods for computation in hybrid systems*, In: N.Lynch and B.H.Krogh, editors, *Hybrid Systems: Computation and Control*, Vol.1790 of *Lecture Notes in Computer Science*. Springer-Verlag.

[19] Level Set Toolbox homepage: *www.cs.ubc.ca/˜mitchell/ToolboxLS*

[20] O.Stursberg, B.H.Krogh (2003). *Efficient representation and computation of reachable sets for hybrid systems*, In: O.Maler and A.Pnueli, editors, *Hybrid Systems: Computation and Control*, Vol.2623 of *Lecture Notes in Computer Science*, pp.482-497. Springer-Verlag.

[21] CheckMate homepage: *www.ece.cmu.edu/˜webk/checkmate*

[22] E.Asarin, O.Bournez, T.Dang, O.Maler (2000). *Approximate reachability analysis of piecewise linear dynamical systems*, In: N.Lynch and B.H.Krogh, editors, *Hybrid Systems: Computation and Control*, Vol.1790 of *Lecture Notes in Computer Science*, pp.21-31. Springer-Verlag.

[23] *d/dt* homepage: *www-verimag.imag.fr/˜tdang/ddt.html*

[24] A.Girard (2005). *Reachability of uncertain linear systems using zonotopes.* In: M.Morari and L.Thiele, editors, *Hybrid Systems: Computation and Control*, Vol.3414 of *Lecture Notes in Computer Science*, pp 291-305. Springer-Verlag.

[25] A.Girard, C.Le Guernic, O.Maler (2006). *Computation of reachable sets of linear time-invariant systems with inputs*, In: J.Hespanha and A.Tiwari, editors, *Hybrid Systems: Computation and Control*, Vol.3927 of *Lecture Notes in Computer Science*, pp.257-271. Springer-Verlag.

[26] MATISSE homepage: *www.seas.upenn.edu/˜agirard/Software/MATISSE*

[27] Zonotope methods on Wolfgang Kühn homepage: *www.decatur.de*

[28] T.S.Motzkin, H.Raiffa, G.L.Thompson, R.M.Thrall (1953). *The Double Description Method.* In: H.W.Kuhn and A.W.Tucker, editors, *Conttributions to theory of games*, Vol.2. Princeton University Press.

[29] CDD/CDD+ homepage: *www.cs.mcgill.ca/˜fukuda/soft/cdd_home/cdd.html*

[30] D.Avis, D.Bremner, R.Seidel (1997). *How good are convex hull algorithms?* In: *Computational Geometry: Theory and Applications* No.7, pp.265-301. ELSEVIER.

[31] G.Lafferriere, G.J.Pappas, S.Yovine (2001). *Symbolic Reachability Computation for Families of Linear Vector Fields.* In: *Journal of Symbolic Computation* No.32, pp.231-253, Academic Press.

[32] Requiem homepage: *www.seas.upenn.edu/˜hybrid/requiem/requiem.html*

[33] S.M.Veres, A.V.Kuntsevich, I.Vályi, S.Hermsmeyer, D.S.Wall (2001). *Geometric Bounding toolbox for MATLAB.* In: *MATLAB/Simulink Connections Catalogue.* Natick, MA: MathWorks Inc.

[34] Geometric Bounding Toolbox homepage: *www.sysbrain.com/gbt*

[35] E.K.Kostousova (2001). *Control synthesis via parallelotopes: optimization and parallel computations.* In: *Optimization Methods and Software*, Vol.14, N.4, pp.267-310, Taylor & Francis.

[36] L.Muñoz, X.Sun, R.Horowitz, L.Alvarez (2003). *Traffic density estimation with the cell transmission model*, In: *Proceedings of the American Control Conference*, Denver, Colorado, USA, pp.3750-3755.

[37] S.Boyd, L.El Ghaoui, E.Feron, V.Balakrishnan (1994). *Linear Matrix Inequalities in System and Control Theory.* SIAM.

[38] S.Boyd, L.Vandenberghe (2004). *Convex Optimization.* Cambridge University Press.

[39] F.R.Gantmacher (1960). *Matrix theory, I-II.* Chelsea Pub., NY.

[40] G.H.Golub, C.F.Van Loan (1996). *Matrix Computations*, 3rd Edn. The Johns Hopkins University Press.

[41] R.T.Rockafellar (1999). *Convex Analysis*, 2nd Edn. Princeton University Press.

Appendix A

Function Reference

A.1 ellipsoid Methods

`dimension` - returns the dimension of the space in which the ellipsoid is defined and the rank of its shape matrix.

Parameters:

- `E` - single ellipsoid or array of ellipsoids.

Returns:

- `n` - dimension of the space.
- `r` - rank of the ellipsoid's shape matrix. This output parameter is optional.

Example:

```
>> E1 = ellipsoid;
>> A = [3 1; 0 1; -2 1]; E2 = ellipsoid([1; -1; 1], A*A');
>> E3 = ellipsoid(eye(2));
>> E4 = ellipsoid(0);
>> [n, r] = dimension([E1 E2; E3 E4])

n =

   0     3
   2     1

r =

   0     2
   2     0
```

`display` - displays the details of the ellipsoid object.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

This function is rarely used explicitely. It is called automatically for the object `E` when semicolon does not terminate the statement.

`distance` - computes the distance from the given ellipsoid to the specified object - vector, ellipsoid, hyperplane or polytope.

Parameters:

- `E` - ellipsoid or array of ellipsoids.
- `X` - in case of vectors, it is a matrix, whose columns represent the vectors to which the distance is measured, number of columns must match the number of ellipsoids in the array `E`;
  in case of ellipsoids, it is array of ellipsoids whose size must match the size of `E`;
  in case of hyperplanes, it is array of hyperplanes whose size must match the size of `E`;
  in case of polytopes, it is polytope array whose length must match the number of ellipsoids in the array `E`.
- `F` - this flag if set to 1 indicates that the distance should be computed in the metric of ellipsoids in `E`. This parameter is optional, its default value is 0, and the distance is computed in the Euclidean metric. This parameter makes no difference for the distance computation between ellipsoids and polytopes.

Returns:

- `D` - distance or array of distances.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> V = [1 1; 1 -1; -1 1; -1 -1]';
>> distance(E, V)

ans =

    1.0000   -1.0000    1.6668    1.0855
```

`ellipsoid` - constructor for the ellipsoid object. If called without parameters, returns empty ellipsoid.

Parameters:

- `q` - center of the ellipsoid, vector in $\mathbf{R}^n$. This parameter is optional, and if omitted, it is assumed that the ellipsoid is centered at the origin.
- `Q` - shape matrix of the ellipsoid, matrix in $\mathbf{R}^{n\times n}$, symmetric positive semidefinite.

Returns:

- `E` - object of type `ellipsoid class`.

Example:

```
>> E = ellipsoid([1 0 -1 6]', 9*eye(4));
```

creates a ball of radius 3 in $\mathbf{R}^4$ centered at $[1\ 0\ -1\ 6]^T$.

`eq` - overloaded operator `'=='`, it checks if two ellipsoids are equal.

Parameters:

- `E1` - first ellipsoid.
- `E2` - second ellipsoid.

Returns 1 if two ellipsoids are equal, 0 - otherwise.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> E == [E ellipsoid(eye(2))]

ans =

     1     0
```

`ge`, `gt` - checks if the first ellipsoid is bigger than the second one.

Parameters:

- `E1` - first ellipsoid.
- `E2` - second ellipsoid.

Returns 1 if `E1` contains `E2` when both have the same center, 0 - otherwise.

Example:

```
>> E > E

ans =

     1
```

illustrates the fact that an ellipsoid is always bigger than itself.

`hpintersection` - computes the ellipsoid which results from intersection of given ellipsoid with given hyperplane.

Parameters:

- `E` - ellipsoid or array of ellipsoids.
- `H` - hyperplane or array of hyperplanes.

If `E` and `H` are arrays, then their sizes must match.

Returns:

- `I` - ellipsoid or array of ellipsoids that are intersections of ellipsoids in `E` with hyperplanes in `H`.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> H = [hyperplane([0 -1; -1 0]', 1); hyperplane([0 1; 1 0]', 1)];
>> I = hpintersection(E, H)

I =
2x2 array of ellipsoids.
```

`intersect` - checks if the union or intersection of ellipsoids intersects given ellipsoid, hyperplane or polytope.

Parameters:

- `E` - ellipsoid or array of ellipsoids whose union or intersection is considered.
- `X` - can be single or array of ellipsoids, hyperplanes or polytopes.
- `s` - if `'u'`, `E` should be treated as union of ellipsoids, if `'i'`, `E` should be treated as intersection of ellipsoids. This parameter is optional, its default value is `'u'`.

Returns -1 in case parameter `s` is set to `'i'` and the intersection of ellipsoids in `E` is empty,
0 if the union or intersection of ellipsoids in `E` does not intersect the object in `X`,
1 if the union or intersection of ellipsoids in `E` and the object in `X` have nonempty intersection.

Example:

```
>> E1 = ellipsoid([-2; -1], [4 -1; -1 1]);
>> E2 = E1 + [5; 5];
>> H  = hyperplane([1; -1]);
>> intersect([E1 E2], H)

ans =

     1

>> intersect([E1 E2], H, 'i')

ans =

    -1
```

Here two ellipsoids `E1` and `E2` do not intersect but both are intersected by hyperplane `H`.

`intersection_ea` - computes the external ellipsoidal approximation of the intersection of the ellipsoid with given ellipsoid, halfspace or polytope.

Parameters:

- `E` - ellipsoid or array of ellipsoids.
- `X` - can be single or array of ellipsoids, hyperplanes or polytopes.

If `E` and `X` are arrays, then their sizes must match.

Returns:

- `EA` - ellipsoid or array of ellipsoids that externally approximate the intersection.

Example:

```
>> E1 = ellipsoid([-2; -1], [4 -1; -1 1]);
>> E2 = E1 + [5; 5];
>> B  = ell_unitball(2);
>> EA = intersection_ea([E1 E2], B)

EA =
1x2 array of ellipsoids.
```

`intersection_ia` - computes the internal ellipsoidal approximation of the intersection of the ellipsoid with given ellipsoid, halfspace or polytope.

Parameters:

- `E` - ellipsoid or array of ellipsoids.
- `X` - can be single or array of ellipsoids, hyperplanes or polytopes.

If `E` and `X` are arrays, then their sizes must match.

Returns:

- `IA` - ellipsoid or array of ellipsoids that internally approximate the intersection.

Example:

```
>> E1 = ellipsoid([-2; -1], [4 -1; -1 1]);
>> E2 = E1 + [5; 5];
>> B  = ell_unitball(2);
>> IA = intersection_ia([E1 E2], B)

IA =
1x2 array of ellipsoids.
```

`inv` - inverts the shape matrix of the ellipsoid if it is nonsingular.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns:

- `I` - ellipsoid or array of ellipsoids with inverted shape matrices.

`isbaddirection` - checks if ellipsoidal approximations of the geometric difference of two ellipsoids can be computed for given directions.

Parameters:

- `E1` - first ellipsoid.
- `E2` - second ellipsoid.
- `L` - matrix whose columns are direction vectors that need to be checked.

Returns 1 if given direction is bad and ellipsoidal approximation cannot be computed for it, 0 - otherwise.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> B = 3*ell_unitball(2);
>> L = [1 0; 1 1; 0 1; -1 1]';
>> isbaddirection(B, E, L)

ans =

     0     1     1     0}
```

means that for vectors $[1\ 1]^T$ and $[0\ 1]^T$ the ellipsoidal approximation of the geometric difference `B`$\dot{-}$`E` cannot be computed.

`isdegenerate` - checks if given ellipsoid is degenerate.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns 1 if the ellipsoid is degenerate, 0 - otherwise.

`isempty` - checks if given ellipsoid is an empty object.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns 1 if empty, 0 - otherwise.

`isinside` - checks if the intersection of ellipsoids contains the union or intersection of given ellipsoids or polytopes.

Parameters:

- `E` - ellipsoid or array of ellipsoids whose intersection contains or not the union or intersection of other given objects.
- `X` - can be array of ellipsoids or polytopes whose union or intersection belongs or not to the intersection `E`.
- `s` - if `'u'`, `X` should be treated as union, if `'i'`, `X` should be treated as intersection. This parameter is optional, its default value is `'u'`.

Returns -1 if parameter `s` is set to `'i'` and the intersection of ellipsoids or polytopes in `X` is empty,
0 - if the intersection `E` does not cover `X`,
1 - if `E` contains `X`.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> B = ell_unitball(2);
>> isinside(E, [E B], 'i')

ans =

     1
```

illustrates the fact that any ellipsoid contains its intersection with another ellipsoid.

`isinternal` - checks if the union or intersection of ellipsoids contains given vectors.

Parameters:

- `E` - ellipsoid or array of ellipsoids whose intersection contains or not given vectors or polytopes.
- `X` - can be matrix whose columns represent vectors to be checked.
- `s` - if `'u'`, `E` should be treated as union, if `'i'`, `E` should be treated as intersection. This parameter is optional, its default value is `'u'`.

Returns 1 if vector in `X` belongs to union or intersection `E`, 0 - otherwise.

Example:

```
>> E1 = ellipsoid([-2; -1], [4 -1; -1 1]);
>> E2 = E1 + [5; 5];
>> isinternal([E1 E2], [2 7; -1 4], 'i')

ans =

     0     0

>> isinternal([E1 E2], [2 7; -1 4])

ans =

     1     1
```

Here the intersection of `E1` and `E2` is empty, and thus, cannot contain any points. The union, on the other hand, contains centers of both ellipsoids.

`le`, `lt` - checks if the second ellipsoid is bigger than the first one.

Parameters:

- `E1` - first ellipsoid.
- `E2` - second ellipsoid.

Returns 1 if `E2` contains `E1` when both have the same center, 0 - otherwise.

This operation is the mirror of `ge`, `gt`.

`maxeig` - returns the biggest eigenvalue of the ellipsoid.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns:

- `a` - largest eigenvalue or array of largest eigenvalues of ellipsoids in `E`.

`mineig` - returns the smallest eigenvalue of the ellipsoid.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns:

- `a` - smallest eigenvalue or array of smallest eigenvalues of ellipsoids in `E`.

`minkdiff` - computes and plots the geometric difference of two ellipsoids in 2D and 3D.

Parameters:

- `E1` - first ellipsoid.
- `E2` - second ellipsoid.
- `o` - options structure whose fields describe how the geometric difference must be plotted. The fields of this structure are
  - `show_all` - if set to 1, also displays the ellipsoids `E1` and `E2`;
  - `newfigure` - if set to 1, each plot command will open a new figure window;
  - `fill` - if set to 1, the resulting set and the ellipsoids, if plotted in 2D, will be filled with color;
  - `color` - specifies the color of the plot in the RGB format: `[x y z]`;
  - `shade` - the level of transparency for 3D plots, takes values between 0 and 1 (0 - transparent, 1 - opaque).

  This parameter is optional.

Returns:

- `x` - center of the resulting set.
- `X` - matrix whose columns represent the boundary points of the resulting set. The number of points is defined by parameters `plot2d_grid` for 2D plots and `plot3d_grid` for 3D plots of the global `ellOptions` structure.

Both output parameters are optional.

`minkdiff_ea` - computes external ellipsoidal approximations of the geometric difference of two ellipsoids of arbitrary dimension for given directions, if these directions are not bad.

Parameters:

- `E1` - first ellipsoid.
- `E2` - second ellipsoid.
- `L` - matrix whose columns specify the directions for which the approximations should be computed.

Returns:

- `EA` - array of computed external ellipsoids. Can be empty, if all the directions specified in `L` are bad.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> B = 3*ell_unitball(2);
>> L = [1 0; 1 1; 0 1; -1 1]';
>> EA = minkdiff_ea(B, E, L)

EA =
1x2 array of ellipsoids.
```

The resulting array `EA` contains only two ellipsoids because two of the four directions specified in `L` are bad.

`minkdiff_ia` - computes internal ellipsoidal approximations of the geometric difference of two ellipsoids of arbitrary dimension for given directions, if these directions are not bad.

Parameters:

- `E1` - first ellipsoid.
- `E2` - second ellipsoid.
- `L` - matrix whose columns specify the directions for which the approximations should be computed.

Returns:

- `EA` - array of computed internal ellipsoids. Can be empty, if all the directions specified in `L` are bad.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> B = 3*ell_unitball(2);
>> L = [1 0; 1 1; 0 1; -1 1]';
>> IA = minkdiff_ia(B, E, L)

IA =
1x2 array of ellipsoids.
```

The resulting array `IA` contains only two ellipsoids because two of the four directions specified in `L` are bad.

`minksum` - computes and plots the geometric sum of finite number of ellipsoids in 2D and 3D.

Parameters:

- `E` - array of ellipsoids whose geometric sum needs to be computed.
- `o` - options structure whose fields describe how the geometric sum must be plotted. The fields of this structure are
  - `show_all` - if set to 1, also displays the ellipsoids in the array `E`;
  - `newfigure` - if set to 1, each plot command will open a new figure window;
  - `fill` - if set to 1, the resulting set and the ellipsoids, if plotted in 2D, will be filled with color;
  - `color` - specifies the color of the plot in the RGB format: `[x y z]`;
  - `shade` - the level of transparency for 3D plots, takes values between 0 and 1 (0 - transparent, 1 - opaque).

  This parameter is optional.

Returns:

- `x` - center of the resulting set.
- `X` - matrix whose columns represent the boundary points of the resulting set. The number of points is defined by parameters `plot2d_grid` for 2D plots and `plot3d_grid` for 3D plots of the global `ellOptions` structure.

Both output parameters are optional.

`minksum_ea` - computes external ellipsoidal approximations of the geometric sum of finite number of ellipsoids of arbitrary dimension for given directions.

Parameters:

- `E` - array of ellipsoids whose geometric sum needs to be approximated.
- `L` - matrix whose columns specify the directions for which the approximations should be computed.

Returns:

- `EA` - array of computed external ellipsoids whose length is the same as the number of columns of matrix `L`.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> B = ell_unitball(2);
>> L = [1 0; 1 1; 0 1; -1 1]';
>> EA = minkdiff_ea([E B inv(E)] L)

EA =
1x4 array of ellipsoids.
```

`minksum_ia` - computes internal ellipsoidal approximations of the geometric sum of finite number of ellipsoids of arbitrary dimension for given directions.

Parameters:

- `E` - array of ellipsoids whose geometric sum needs to be approximated.
- `L` - matrix whose columns specify the directions for which the approximations should be computed.

Returns:

- `IA` - array of computed internal ellipsoids whose length is the same as the number of columns of matrix `L`.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> B = ell_unitball(2);
>> L = [1 0; 1 1; 0 1; -1 1]';
>> IA = minkdiff_ia([E B inv(E)] L)

IA =
1x4 array of ellipsoids.
```

`minus` - overloaded operator `'-'`.

Parameters:

- `E` - ellipsoid or array of ellipsoids defined in $\mathbf{R}^n$.
- `b` - vector in $\mathbf{R}^n$ or matrix in $\mathbf{R}^{n\times m}$, where m equals the number of ellipsoids in the array `E`.

Returns:

- `E1` - ellipsoid or array of ellipsoids with same shapes as `E`, but with centers shifted by vectors in `-b`.

Example:

```
>> E  = [ellipsoid([-2; -1], [4 -1; -1 1]) ell_unitball(2)];
>> E1 = E - [1; 1];
>> E1(1)

ans =

Center:
    -3
    -2

Shape:
     4    -1
    -1     1

Nondegenerate ellipsoid in R^2.

>> E1(2)

ans =

Center:
    -1
    -1

Shape:
     1     0
     0     1

Nondegenerate ellipsoid in R^2.
```

`move2origin` - moves given ellipsoids to the origin.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns:

- `O` - the same ellipsoid or array of ellipsoids as `E`, but all centered at the origin.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> O = move2origin(E)

O =

Center:
     0
     0

Shape:
     4    -1
    -1     1

Nondegenerate ellipsoid in R^2.
```

`mtimes` - overloaded operator `'*'`.

Parameters:

- `A` - scalar, or matrix in $\mathbf{R}^{m\times n}$.
- `E` - ellipsoid or array of ellipsoids defined in $\mathbf{R}^n$.

Returns:

- `E1` - ellipsoid or array of ellipsoids resulting from linear transformation of `E`.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> A = [0 1; -1 0];
>> A*E

ans =

Center:
    -1
     2

Shape:
     1     1
     1     4

Nondegenerate ellipsoid in R^2.
```

`ne` - overloaded operator `'~='`, it checks if two ellipsoids are not equal.

Parameters:

- `E1` - first ellipsoid.
- `E2` - second ellipsoid.

Returns 1 if two ellipsoids are not equal, 0 - otherwise.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> E \~{ }= [E ellipsoid(eye(2))]

ans =

     0     1
```

`parameters` - returns parameters of the ellipsoid, its center and shape matrix.

Parameters:

- `E` - single ellipsoid.

Returns:

- `q` - center of the ellipsoid. It is optional output parameter.
- `Q` - shape matrix of the ellipsoid.

`plot` - plots ellipsoids in 1D, 2D and 3D.

Parameters:

- `E` - ellipsoid or array of ellipsoids.
- `c` - specifies the color:
  - `'b'` - blue;
  - `'c'` - cyan;
  - `'g'` - green;
  - `'k'` - black;
  - `'m'` - magenta;
  - `'r'` - red;
  - `'w'` - white;
  - `'y'` - yellow.

  This parameter is optional.
- `o` - options structure whose fields describe how the ellipsoids must be plotted. The fields of this structure are
  - `newfigure` - if set to 1, each plot command will open a new figure window;
  - `fill` - if set to 1, the resulting set and the ellipsoids, if plotted in 2D, will be filled with color;
  - `width` - specifies line width for 1D and 2D plots;
  - `color` - specifies the color of the plot in the RGB format: `[x y z]`;
  - `shade` - the level of transparency for 3D plots, takes values between 0 and 1 (0 - transparent, 1 - opaque).

  This parameter is optional.

Returns: None.

`plus` - overloaded operator `'+'`.

Parameters:

- `E` - ellipsoid or array of ellipsoids defined in $\mathbf{R}^n$.
- `b` - vector in $\mathbf{R}^n$ or matrix in $\mathbf{R}^{n\times m}$, where m equals the number of ellipsoids in the array `E`.

Returns:

- `E1` - ellipsoid or array of ellipsoids with same shapes as `E`, but with centers shifted by vectors in `b`.

Example:

```
>> E  = [ellipsoid([-2; -1], [4 -1; -1 1]) ell_unitball(2)];
>> E1 = E + [1; 1];
>> E1(1)

ans =

Center:
    -1
     0

Shape:
     4    -1
    -1     1

Nondegenerate ellipsoid in R^2.

>> E1(2)

ans =

Center:
     1
     1

Shape:
     1     0
     0     1

Nondegenerate ellipsoid in R^2.
```

`polar` - computes polars for ellipsoids which contain the origin.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns:

- `P` - polar ellipsoid or array of polar ellipsoids for those in `E`.

For ellipsoids that do not contain the origin, this function returns empty ellipsoid.

Example:

```
>> E = ellipsoid([4 -1; -1 1]);
>> polar(E) == inv(E)

ans =

     1
```

illustrates the fact that polar set of an ellipsoid centered at the origin equals to inverse of this ellipsoid.

`projection` - computes projection of ellipsoids onto given orthogonal basis.

Parameters:

- `E` - ellipsoid or array of ellipsoids defined in $\mathbf{R}^n$.
- `B` - matrix in $\mathbf{R}^{n\times m}$ whose columns are orthogonal.

Returns:

- `P` - array of projected ellipsoids.

Example:

```
>> E = ellipsoid([-2; -1; 4], [4 -1 0; -1 1 0; 0 0 9]);
>> B = [0 1 0; 0 0 1]';
>> P = projection(E, B)

P =

Center:
    -1
     4

Shape:
     1     0
     0     9

Nondegenerate ellipsoid in R^2.
```

`rho` - computes the support function of the ellipsoids for given directions and corresponding boundary points.

Parameters:

- `E` - ellipsoid or array of ellipsoids defined in $\mathbf{R}^n$.
- `L` - vector in $\mathbf{R}^n$ if `E` is an array of ellipsoids, otherwise, if `E` is a single ellipsoid, it can be matrix in $\mathbf{R}^{n\times m}$ whose columns represent the directions for which the support function needs to be computed.

Returns:

- `R` - array of support function values.
- `X` - matrix whose columns are boundary points corresponding to the directions in `L`. This output parameter is optional.

`shape` - has the same functionality as `mtimes` but modifies only the shape matrix of the ellipsoid leaving its center as is.

Parameters:

- `A` - scalar, or matrix in $\mathbf{R}^{m \times n}$.
- `E` - ellipsoid or array of ellipsoids defined in $\mathbf{R}^n$.

Returns:

- `E1` - ellipsoid or array of ellipsoids resulting from modification of the shape matrices of ellipsoids in `E`.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> A = [0 1; -1 0];
>> shape(E, A)

ans =

Center:
    -2
    -1

Shape:
     1     1
     1     4

Nondegenerate ellipsoid in R^2.
```

`uminus` - overloaded operation unitary minus.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns:

- `E1` - array of the same ellipsoids as in `E`, whose centers are multilied by -1.

Example:

```
>> E = -ellipsoid([-2; -1], [4 -1; -1 1])

E =

Center:
     2
     1

Shape:
     4    -1
    -1     1

Nondegenerate ellipsoid in R^2.
```

`volume` - computes volume of given ellipsoids.

Parameters:

- `E` - ellipsoid or array of ellipsoids.

Returns:

- `V` - array of volume values whose size matches the size of `E`.

Example:

```
>> E = ellipsoid([-2; -1], [4 -1; -1 1]);
>> B = ell_unitball(2);
>> V = volume([E B])

V =

    5.4414    3.1416
```

A.2 hyperplane Methods

`contains` - checks if the hyperplanes contain given vectors.

Parameters:

- `H` - hyperplane or array of hyperplanes.
- `X` - matrix whose columns represent the vectors needed to be checked. The number of columns must match the number of hyperplanes in `H`.

Returns 1 if the vector in `X` belongs to the hyperplane in `H`, 0 - otherwise.

Example:

```
>> H = hyperplane([-1; 1]);
>> X = [100 -1 2; 100 1 2];
>> contains(H, X)

ans =

     1     0     1
```

`dimension` - returns the dimension of the space in which the hyperplane is defined.

Parameters:

- `H` - hyperplane or array of hyperplanes.

Returns:

- `D` - array of dimension values of the same size as `H`.

Example:

```
>> H1 = hyperplane([-1; 1]);
>> H2 = hyperplane([-1; 1; 8; -2; 3], 7);
>> H3 = hyperplane([1; 2; 0], -1);
>> D  = dimension([H1 H2 H3]);

D =

   2     5     3
```

`display` - displays the details of the hyperplane object.

Parameters:

- `H` - hyperplane or array of hyperplanes.

This function is rarely used explicitely. It is called automatically for the object `H` when semicolon does not terminate the statement.

`eq` - overloaded operator `'=='`. it checks if two hyperplanes are equal.

Parameters:

- `H1` - first hyperplane.
- `H2` - second hyperplane.

Returns 1 if two hyperplanes are equal, 0 - otherwise.

Example:

```
>> H1 = hyperplane([-1; 1]);
>> H2 = hyperplane([-1; 1; 8; -2; 3], 7);
>> H3 = hyperplane([1; 2; 0], -1);
>> H2 == [H1 H2 H3]

ans =

     0     1     0
```

`hyperplane` - constructor for the hyperplane object. If called without parameters, returns empty hyperplane.

Parameters:

- `V` - vector in $\mathbf{R}^n$ or matrix in $\mathbf{R}^{n \times m}$ whose columns define the normals to the hyperplanes that need to be created.
- `C` - scalar value, or, in case `V` has m columns, it can be array with m values.

Returns:

- `H` - hyperplane, or, if `V` has more than one column, array of hyperplanes.

Example:

```
>> V = [1 1 1; 1 1 1];
>> C = [1 -5 0];
>> H = hyperplane(V, C);
```

defines three parallel hyperplanes and returns them in the array `H`.

`isempty` - checks if the hyperplane object is empty.

Parameters:

- `H` - hyperplane or array of hyperplanes.

Returns: 1 if the hyperplane object is empty, 0 - otherwise.

`isparallel` - checks if the hyperplanes are parallel.

Parameters:

- `H1` - first hyperplane.
- `H2` - second hyperplane.

Returns 1 if two hyperplanes are equal, 0 - otherwise.

Example:

```
>> H = hyperplane([-1 1; 1 1; 1 1], [2 1 0]);
>> isparallel(H, H(2))

ans =

     0     1     1
```

`ne` - overloaded operator `'~='`, it checks if two hyperplanes are not equal.

Parameters:

- `H1` - first hyperplane.
- `H2` - second hyperplane.

Returns 1 if two hyperplanes are not equal, 0 - otherwise.

Example:

```
>> H1 = hyperplane([-1; 1]);
>> H2 = hyperplane([-1; 1; 8; -2; 3], 7);
>> H3 = hyperplane([1; 2; 0], -1);
>> [H2 H1 H3] == [H1 H2 H3]

ans =

     0     0     1
```

`parameters` - returns parameters of the hyperplane object, its normal and scalar.

Parameters:

- `H` - hyperplane object.

Parameters:

- `v` - normal vector.
- `c` - scalar. This output parameter is optional.

`plot` - plots hyperplanes in 2D and 3D.

Parameters:

- `H` - hyperplane or array of hyperplanes.
- `c` - specifies the color:
  - `'b'` - blue;
  - `'c'` - cyan;
  - `'g'` - green;
  - `'k'` - black;
  - `'m'` - magenta;
  - `'r'` - red;
  - `'w'` - white;
  - `'y'` - yellow.

  This parameter is optional.
- `o` - options structure whose fields describe how the hyperplanes must be plotted. The fields of this structure are
  - `newfigure` - if set to 1, each plot command will open a new figure window;
  - `size` - length of the line segment in 2D or square diagonal in 3D.
  - `center` - center of the line segment in 2D or square diagonal in 3D.
  - `width` - specifies line width for 2D plots;
  - `color` - specifies the color of the plot in the RGB format: `[x y z]`;
  - `shade` - the level of transparency for 3D plots, takes values between 0 and 1 (0 - transparent, 1 - opaque).

  This parameter is optional.

Returns: None.

`uminus` - overloaded operator unitary minus. It does not change the hyperplane, and affects only the halfspace this hyperplane defines.

Parameters:

- `H` - hyperplane or array of hyperplanes.

Returns:

- `H1` - array of the same hyperplanes as in `H` whose normals and scalars are multiplied by -1.

Example:

```
>> H = -hyperplane([-1; 1], 1)

H =

Normal:
     1
    -1

Shift:
    -1

Hyperplane in R^2.
```

A.3 linsys Methods

`dimension` - returns dimensions of state, input, output and disturbance input spaces.

Parameters:

- `LSYS` - linear system or array of linear systems.

Returns:

- `N` - state space dimension.
- `I` - dimension of the input space.
- `O` - dimension of the output space.
- `D` - dimension of the disturbance input space.

`display` - displays the details of the linear system object.

Parameters:

- `LSYS` - linear system or array of linear systems.

This function is rarely used explicitely. It is called automatically for the object `LSYS` when semicolon does not terminate the statement.

`hasdisturbance` - checks if given linear system is the system with disturbance.

Parameters:

- `LSYS` - linear system or array of linear systems.

Returns 1 if it is system with disturbance, 0 - otherwise.

`hasnoise` - checks if given linear system has noise at the output.

Parameters:

- `LSYS` - linear system or array of linear systems.

Returns 1 if it is system with noise, 0 - otherwise.

`isdiscrete` - checks if given linear system is discrete-time.

Parameters:

- `LSYS` - linear system or array of linear systems.

Returns 1 if the system is discrete-time, 0 - otherwise.

`isempty` - checks if given linear system is an empty object.

Parameters:

- `LSYS` - linear system or array of linear systems.

Returns 1 if `LSYS` is an empty object, 0 - otherwise.

`islti` - checks if given linear system is time invariant.

Parameters:

- `LSYS` - linear system or array of linear systems.

Returns 1 if the system is time invariant, 0 - otherwise.

`linsys` - constructor for the linear system object. If called without parameters, creates an empty object.

Parameters:

- `A` - matrix $A \in \mathbf{R}^{n\times n}$, can be symbolic if it depends on time (t - in continuous case, k - in discrete case).
- `B` - matrix $B \in \mathbf{R}^{n\times m}$, can by symbolic if it depends on time.
- `U` - defines control bounds: it can be ellipsoid object with dimension m; or structure with fields `center` - symbolic vector, and `shape` - symbolic matrix, if the ellipsoidal bounds depend on time; or, if the control is fixed, it can be single vector of type *double* or symbolic.
- `G` - matrix $G \in \mathbf{R}^{n\times d}$, can by symbolic if it depends on time. This parameter is optional.
- `V` - defines disturbance bounds: it can be ellipsoid object with dimension d; or structure with fields `center` - symbolic vector, and `shape` - symbolic matrix, if the ellipsoidal bounds depend on time; or, if the control is fixed, it can be single vector of type *double* or symbolic. This parameter is optional.
- `C` - matrix $C \in \mathbf{R}^{o\times n}$, can by symbolic if it depends on time. This parameter is optional.
- `W` - defines noise bounds: it can be ellipsoid object with dimension o; or structure with fields `center` - symbolic vector, and `shape` - symbolic matrix, if the ellipsoidal bounds depend on time; or, if the control is fixed, it can be single vector of type *double* or symbolic. This parameter is optional.
- `s` - if set to `'d'`, indicates that the system is discrete-time.

Returns:

- `LSYS` - linear system object.

Example:

```
>> A = {'0' '1 + cos(pi*k/2)'; '-2' '0'};
>> B = [0; 1];
>> U = ellipsoid(4);
>> G = [1; 0];
>> V = 1/(k+1);
>> C = [1 0];
>> lsys = linsys(A, B, U, G, V, C, [], 'd');
```

defines the following affine discrete-time system:

$$\begin{aligned}
\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \end{bmatrix} &= \begin{bmatrix} 0 & 1+\cos(\frac{\pi k}{2}) \\ -2 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u[k] + \begin{bmatrix} \frac{1}{k+1} \\ 0 \end{bmatrix}, \quad -2 \leq u[k] \leq 2 \\
y[k] &= [1 \;\; 0] \begin{bmatrix} x_1[k] \\ x_2[k] \end{bmatrix}, \quad k \geq 0.
\end{aligned}$$

A.4 reach Methods

`cut` - extracts a segment of the reach tube from the given start time to the given end time.

Parameters:

- `RS` - reach set object.
- `T` - time interval of interest in the form `[t1 t2]`. If `T` is a single scalar, then the reach set for this particular time value is returned.

Returns:

- `CRS` - the resulting reach set object.

`dimension` - returns the dimension of the reach set and the dimension of the state space for which the reach set was originally computed.

Parameters:

- `RS` - reach set object.

Returns:

- `d` - dimension of the reach set.
- `n` - dimension of the state space for which the reach set was originally computed. Values of `d` and `n` can be different if the reach set object is a result of `projection` operation. This parameter is optional.

`display` - displays the details of the reach set object.

Parameters:

- `RS` - reach set object.

This function is rarely used explicitely. It is called automatically for the object `LSYS` when semicolon does not terminate the statement.

`evolve` - computes further evolution in time of the already existing reach set.

Parameters:

- `CRS` - existing reach set.
- `T` - new time horizon.
- `LSYS` - linear system object that describes new dynamics. This parameter is optional.

Returns:

- `RS` - reach set object.

`get_center` - returns trajectory of the center of the reach set.

Parameters:

- `RS` - reach set object.

Returns:

- `X` - matrix whose columns represent the position of the center at times specified in the second output parameter, `T`.
- `T` - array of time values at which the trajectory of the reach set center is evaluated. This output parameter is optional.

The number of columns in `X` and values in `T` is specified by the parameter `time_grid` of the `ellOptions` structure.

`get_directions` - returns the trajectories of the direction vectors for which the ellipsoidal approximations were computed.

Parameters:

- `RS` - reach set object.

Returns:

- `X` - array of cells where each cell represents a trajectory of the direction vector, for which the ellipsoidal approximations of the reach set were computed.
- `T` - array of time values at which these trajectories are evaluated. This output parameter is optional.

The number of columns in cells of `X` and values in `T` is specified by the parameter `time_grid` of the `ellOptions` structure.

`get_ea` - returns array of ellipsoids that represent the external approximation of the reach set.

Parameters:

- `RS` - reach set object.

Returns:

- `E` - array of ellipsoids that externally approximate the reach set. The intersection of ellipsoids in the given column of this array is the external approximation of the reach set at the time spacified by the corresponding value in the array `T`.
- `T` - array of time values at which the approximations are evaluated. This output parameter is optional.

The number of columns in `E` and values in `T` is specified by the parameter `time_grid` of the `ellOptions` structure.

`get_goodcurves` - returns the trajectories along which the ellipsoidal approximations are touching the actual reach set.

Parameters:

- `RS` - reach set object.

Returns:

- `X` - array of cells where each cell represents a good trajectory along which one of the computed ellipsoidal approximations touches the reach set.
- `T` - array of time values at which the good trajectories are evaluated. This output parameter is optional.

The number of columns in cells of `X` and values in `T` is specified by the parameter `time_grid` of the `ellOptions` structure.

`get_ia` - returns array of ellipsoids that represent the internal approximation of the reach set.

Parameters:

- `RS` - reach set object.

Returns:

- `E` - array of ellipsoids that internally approximate the reach set. The union of ellipsoids in the given column of this array is the internal approximation of the reach set at the time specified by the corresponding value in the array `T`.
- `T` - array of time values at which the approximations are evaluated. This output parameter is optional.

The number of columns in `E` and values in `T` is specified by the parameter `time_grid` of the `ellOptions` structure.

`get_system` - returns the linear system object for which the reach set was computed.

Parameters:

- `RS` - reach set object.

Returns:

- `LSYS` - the linear system object, which was used for the reach set computation.

`intersect` - checks if the external or internal approximation of the reach set intersects with given ellipsoids, hyperplanes or polytopes.

Parameters:

- `RS` - reach set object.
- `X` - can be array of ellipsoids, hyperplanes or polytopes.
- `s` - if set to `'i'`, indicates that internal approximation should be checked, if set to `'e'` - external. This input parameter is optional, its default value is `'e'`.

Returns 1 if the intersection is nonempty, 0 - otherwise.

`iscut` - checks if the given reach set object resulted from `cut` operation.

Parameters:

- `RS` - reach set object.

Returns 1 if `RS` is a result of `cut` operation, 0 - otherwise.

`isempty` - checks if the given reach set object is empty.

Parameters:

- `RS` - reach set object.

Returns 1 if it is an empty object, 0 - otherwise.

`isprojection` - checks if the given reach set is a result of `projection` operation.

Parameters:

- `RS` - reach set object.

Returns 1 if `RS` is a projection, 0 - otherwise.

`plot_ea` - plots external approximation of the reach set in 2D and 3D.

Parameters:

- `RS` - reach set object.
- `c` - specifies the color:
  - `'b'` - blue;
  - `'c'` - cyan;
  - `'g'` - green;
  - `'k'` - black;
  - `'m'` - magenta;
  - `'r'` - red;
  - `'w'` - white;
  - `'y'` - yellow.

  This parameter is optional.
- `o` - options structure whose fields describe how the reach set must be plotted. The fields of this structure are
  - `width` - specifies line width for 2D plots;
  - `fill` - if set to 1, the set, if plotted in 2D, will be filled with color;
  - `color` - specifies the color of the plot in the RGB format: `[x y z]`;
  - `shade` - the level of transparency for 3D plots, takes values between 0 and 1 (0 - transparent, 1 - opaque).

  This parameter is optional.

Returns: None.

`plot_ia` - plots internal approximation of the reach set in 2D and 3D.

Parameters:

- `RS` - reach set object.
- `c` - specifies the color:
  - `'b'` - blue;
  - `'c'` - cyan;
  - `'g'` - green;
  - `'k'` - black;
  - `'m'` - magenta;
  - `'r'` - red;
  - `'w'` - white;
  - `'y'` - yellow.

  This parameter is optional.
- `o` - options structure whose fields describe how the reach set must be plotted. The fields of this structure are
  - `width` - specifies line width for 2D plots;
  - `fill` - if set to 1, the set, if plotted in 2D, will be filled with color;
  - `color` - specifies the color of the plot in the RGB format: `[x y z]`;
  - `shade` - the level of transparency for 3D plots, takes values between 0 and 1 (0 - transparent, 1 - opaque).

  This parameter is optional.

Returns: None.

`projection` - projects the reach set onto the given orthogonal basis.

Parameters:

- `RS` - reach set object with dimension n.
- `B` - matrix in $\mathbf{R}^{n\times m}$ whose columns are orthogonal, they represent the basis vectors.

Returns:

- `PRS` - reach set object with the projected reach set.

`reach` - constructor for the reach set object and the main function that computes the reach set.

Parameters:

- `LSYS` - the linear system object with state space dimension n.
- `X0` - ellipsoid object with dimension n, it represents the set of initial conditions.
- `L0` - matrix in $\mathbf{R}^{n\times m}$ whose columns represent the directions for which the new approximations need to be computed.
- `T` - time interval in the form `[t0 t1]`. If `t1` is smaller than `t0`, then the backward reach set should be computed. If `T` is a scalar, then it is assumed that `t0 = 0`.
- `o` - options structure with fields
  - `approximation` - if set to 0, then only external approximation is computed, if set to 1, then only internal approximation is computed, if set to 2 (default value), then both approximations are computed.
  - `save_all` - if set to 0 (default value), then the intermediate calculation data should not be saved, 1indicates the opposite.

  This parameter is optional.

Returns:

- `RS` - reach set object.

`refine` - adds new approximations for the specified directions to the given reach set object, thus improving the overall approximation.

Parameters:

- `RS` - reach set object with dimension n. This object must contain the intermediate calculation data, otherwise, the refinement is not possible.
- `L0` - matrix in $\mathbf{R}^{n\times m}$ whose columns represent the directions for which the new approximations need to be computed.
- `o` - options structure with fields
  - `approximation` - if set to 0, then only external approximation is computed, if set to 1, then only internal approximation is computed, if set to 2 (default value), then both approximations are computed.
  - `save_all` - if set to 0, then the intermediate calculation data should not be saved, 1 (default value) indicates the opposite.

  This parameter is optional.

Returns:

- `RRS` - reach set object with refined approximation.

A.5 Miscellaneous Functions

`ell_simdiag` - computes the orthogonal transformation matrix that simultaneously diagonalizes two symmetric matrices.

Parameters:

- `A` - symmetric positive definite matrix in $\mathbf{R}^{n\times n}$.
- `B` - symmetric positive semidefinite matrix in $\mathbf{R}^{n\times n}$.

Returns:

- `T` - orthogonal matrix in $\mathbf{R}^{n\times n}$, such that TAT^T is identity matrix, and TBT^T is diagonal martrix.

`ell_unitball` - creates the ellipsoid object that represents a unit ball of the given dimension.

Parameters:

- `n` - dimension of the space in which the unit ball is defined.

Returns:

- `B` - ellipsoid object with identity shape matrix centered at the origin.

`ell_valign` - computes the orthogonal matrix that aligns two vectors.

Parameters:

- `w` - vector in $\mathbf{R}^n$.
- `v` - vector in $\mathbf{R}^n$ that needs to be rotated to be parallel to `w`.

Returns:

- `S` - orthogonal matrix in $\mathbf{R}^{n \times n}$, such that $Sv = \frac{\|v\|_2}{\|w\|_2} w$.

`hyperplane2polytope` - converts array of hyperplanes of the same dimension into the polytope object of the Multi-Parametric Toolbox.

Parameters:

- `H` - array of hyperplanes, all of which must have the same dimension.

Returns:

- `P` - polytope object as defined in the Multi-Parametric Toolbox.

`polytope2hyperplane` - converts the polytope object of the Multi-Parametric Toolbox into the array of hyperplanes.

Parameters:

- `P` - polytope object as defined in the Multi-Parametric Toolbox.

Returns:

- `H` - array of hyperplanes.