User’s Manual

* * *

Reference of the Geometric Bounding Toolbox
Version 7.3

SysBrain Ltd
Southampton, UK, March 2004
Foreword

The routines in GBT 7.3 permit reliable and fast manipulation of high-dimensional polytopes. GBT 7.3 is a unique computational package of its kind because of its numerical error control. Compared with the previous version 6.1, GBT 7.0 was complete rewrite in 2000, based on experience accumulated since 1993 when GBT 1.0 was first launched. Today GBT 7.3 has new features in that non-convex objects can also be handled using non-convex polyedral sets (NPS or complex). Further new types are solid object models, cylinder and in MATLAB object form points, half-spaces, hyperplanes, covariance, convex-object, non-convex object.

When compared with GBT 6.1, the main differences are:

• GBT 7.3 uses a different polytope representation matrix but the same ellipsoid representation. The changes were required to permit a new method for computational error control to be introduced. There are format conversion routines gbt627 and gbt726 to ensure compatibility with GBT 6.1.

• With GBT 7.3, it possible to handle polytopes of arbitrary topological structure. Polytopes with vertices where the number of facets that ‘meet’ exceeds in number the dimension of the representation space can now be handled. Polytopes with lower dimension than that of the representation space, can also be handled. These changes permit arbitrary affine transformations of polytopes.

• For numerical reasons (which may appear at first to be a limitation compared with previous practice), only bounded polyhedra are considered. This restriction is required for effective error control (introduced since GBT 7.1) whether floating point (as in MATLAB) or fixed-point arithmetic (as in some DSPs) is used.

Numerical errors are monitored in all routines to allow nearly bug-free operation [12]. At the heart of the package is a convex hull algorithm, similar to the quickhull [2, 1] algorithm but with special care taken to ensure numerical reliability and efficiency.

Since GBT 7.3 a novelty of GBT is the handling of non-convex polyhedral sets (NPS). NPSs are represented by a list of convex polytopes, the set-union/set-difference/set-intersection of which defines a non-convex set. This choice of representation was made for its simplicity and reliability as opposed to the listing of \((n – 1)\) dimensional facets of a non-convex polyhedral set, which could be more parsimonious. Reliability is placed above saving memory.

GBT 7.3 widens the object classes by solid-object models for polytopes, ellipsoids and cylinders, and the option to handle all classes of objects by MATLAB.
objects. MATLAB objects make computations more elegant and simpler under the MATLAB environment. They cannot however be compiled into C, C++ and executable code. For this reason all the essential computations of GBT 7.3 are also available without MATLAB objects.

Chapter 1 provides an introduction. Chapter 2 defines the standard GBT format of convex and non-convex geometric objects. Chapter 3 explains the handling and monitoring of numerical accuracy within GBT 7.3. Chapter 4 is a reference manual for most routines contained in GBT 7.3 except the GBT MATLAB object methods and some auxiliary routines.

Warning: Although GBT is the most numerically reliable computational package for nD polytope computations, the computations will only be accurate and bug free if the conditions on error control, as described in Chapter 3, are maintained.

Queries on GBT can be addressed to gbt@sysbrain.com.
Contents

Foreword iii

1 Introduction 1
   1.1 Basic concepts ............................................. 1
   1.2 List of Routines ........................................... 4

2 Standard Object Representations 11
   2.1 Polytopes .................................................. 11
   2.2 NPS representation ........................................ 13
   2.3 Ellipsoid representation ................................... 13
   2.4 Cylinder representation ................................... 14
   2.5 Complex objects ........................................... 15
   2.6 Physical Objects .......................................... 15
   2.7 MATLAB Objects in GBT .................................. 15
      2.7.1 Object classes ........................................ 15
      2.7.2 Overview of methods .................................. 18

3 Error control and speed of computation 25
   3.1 Global tolerance \( \delta \) and local precision \( \mu \) ................. 25
   3.2 Speed of computation and memory requirements .................. 27

4 Programming Reference 31
   4.1 Licence Terms and Conditions - GBT 7.2 ....................... 152
Chapter 1

Introduction

1.1 Basic concepts

The objective of this toolbox is to provide tools for numerical computation with polytopes and ellipsoids in the $n$-dimensional Euclidean space for $n \geq 1$. The toolbox covers such computations as convex hull determination in both the sense of vertex enumeration and facet enumeration, polytope addition and difference in the Minkowski sense, intersections, hyper-volumes, surface-areas, orthogonal projections, affine transformations. For ellipsoids various operations are available such as smallest volume ellipsoid covering a polytope, interior and exterior approximations to the sum, difference and intersection of ellipsoids. These operations now find wide applicability in science and engineering [6, 7, 5, 10, 3].

Notation

$\mathbb{R}^n$ will denote the $n$-dimensional Euclidean space, its points will be represented by column vectors $\mathbf{a} = [a_1, a_2, \ldots, a_n]^T$.

$|\mathbf{a}|$ denotes $||a_1||, ||a_2||, \ldots, ||a_n||$ for a vector $\mathbf{a} = [a_1, a_2, \ldots, a_n]^T$.

The distance of two points $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^n$ will be measured in the 2-norm as $||\mathbf{a} - \mathbf{b}||_2$, i.e. $\left[ \sum_i^n (a_i - b_i)^2 \right]^{1/2}$.

The rows of a matrix $\mathbf{A}$ will be denoted by $\mathbf{A}^1, \ldots, \mathbf{A}^n$ and its column by $\mathbf{A}_1, \ldots, \mathbf{A}_n$.

A subset of row and column vectors from indices $i$ to $j$ will be denoted by $\mathbf{A}^i:j$ and $\mathbf{A}_{i:j}$, respectively.

$\text{Proj}_{\text{orth}}(x|S)$ denotes the orthogonal projection of $x \in \mathbb{R}^n$ onto an affine subspace $S$ in $\mathbb{R}^n$.

$\text{dist}(x, S)$ denotes the $\|\cdot\|_2$-distance of a point $x \in \mathbb{R}^n$ from an affine subspace $S$.

Basic concepts, convex hulls, facet enumeration, vertex enumeration, polytope addition and subtraction in the Minkowski sense will be outlined in this subsection in terms of the ideal mathematical notions. First a few definitions will be introduced.

Definition 1.1.1 For any $\mathbf{a} \in \mathbb{R}^n$, $||\mathbf{a}|| \neq 0$ and any $\mathbf{b} \in \mathbb{R}$ the geometric space

$$ H^{\mathbf{a}_i} \equiv \{ \mathbf{x} \in \mathbb{R}^n \mid \mathbf{a}^T \mathbf{x} \leq \mathbf{b} \} $$

is called a half-space.

Intersections of a finite number of half-spaces are polyhedra and polytopes are those polyhedra which are finite.
CHAPTER 1. INTRODUCTION

Definition 1.1.2 The intersection of a finite set of half-spaces is called a polytope if it is a bounded set.

The convex hull of a finite set of points in \( \mathbb{R}^n \) is the smallest set which contains all the points and also the connecting interval of any two of its points.

Lemma 1.1.1 The convex hull of a finite set of points is a polyhedron, i.e. it can be represented as the intersection of a finite set of half-spaces. \([15]\)

The most important faces of a polytope are its vertices and facets \([15]\). The vertices are 0-dimensional faces and the facets are \((n-1)\)-dimensional faces. \((n-2)\)-dimensional faces are called ridges. The facets of a 3-dimensional polytope are its sides and its ridges are its edges.

Main computational problems

Computing the convex hull of a finite set of points in \( \mathbb{R}^n \) means to determine all the facets of the convex hull polytope.

Problem 1 Given a set of points \( v_1, v_2, \ldots, v_N \in \mathbb{R}^n \), compute the facets of their convex hull.

Problem 1 is also called the vertex enumeration problem \([1, 2]\). Its dual problem is the one in which half-spaces are given an the vertices of the intersecting polytope are to be computed.

Problem 2 Given a set of half-spaces defined by \( a_1^T x \leq b_1, a_2^T x \leq b_2, \ldots, a_N^T x \leq b_N, \|a_i\| \neq 0, a_i \in \mathbb{R}^n \), compute the vertices of the intersecting polyhedron.

The sum and difference of two polytopes will be considered in algebraic sense, which is also called the Minkowski sum and difference.

Definition 1.1.3 Let two polytopes \( P_1 \) and \( P_2 \) be given in \( \mathbb{R}^n \). The sum of polytopes \( P_1 \) and \( P_2 \) will be defined by

\[
P_1 \oplus P_2 \overset{\text{def}}{=} \{ x + y \mid x \in P_1, y \in P_2 \}
\]  

The difference of polytopes \( P_1 \) and \( P_2 \) will be defined by

\[
P_1 \ominus P_2 \overset{\text{def}}{=} \{ x \in \mathbb{R}_n \mid \forall y \in P_1 : x + y \in P_2 \}
\]  

Problem 3 Given two polytopes, each by a list of their defining half-space inequalities and list of vertices, compute their sum and difference.

Wider problems, which can also be solved by finding solution to the problems listed above, are the next two problems.

Problem 4 Given two polytopes only by their defining half-space inequalities, compute their sum and difference.

Finally there are two subproblems listed here for their importance, which relate to Problems 1-2.

Subproblem 1.1 Given a polytope and a vertex, compute their joint convex hull.

This operation is also called convex-hull inclusion.
1.1. BASIC CONCEPTS

Subproblem 2.1 Given a polytope and a half-space, compute their intersection.

This operation is also called polytope updating.

Further problems of common interest are: affine transformations of polytopes, intersection of two polytopes, convex hull of the union set of two polytopes, etc. These operations can be easily derived from solutions to Problems 1-5 as listed and will not therefore be discussed here in detail.

GBT 7.3 can also compute with sets of nonconvex sets, so called non-convex polyhedral sets (NPSs), represented as the difference set of the union of two sets of polytopes, which form a single object and can be referred to by a single name.

Definition 1.1.4 A set \( N \subset \mathbb{R}^n \) is called a non-convex polyhedral set (NPS), if there are two sets of polytopes \( S_1 = \{P_i^+ | i = 1, ..., n_1 \} \) and \( S_2 = \{P_i^- | i = 1, ..., n_2 \} \), so that

\[
N = \bigcup_{i=1}^{n_1} P_i^+ \setminus \bigcup_{i=1}^{n_2} P_i^-
\]  

(1.4)

Operations of union, intersection, volume (in 3D) and area (in 2D) computations are available. The routines operating on nonconvex sets (in the form of NPSs) are distinguished by names ending with .

GBT is a unique package which ensures best computational accuracy and logical consistency. The accuracy problem proves to be crucial for logical consistency as otherwise no reliable decisions can be made on whether a vertex lies on a given hyperplane or not. The major stumbling block for earlier efforts [9, 14, 11, 1, 2, 8] was how to decide on suitable error bounds which could have been used to make adjacency decisions on vertices and facets. The numerical problems related to facet and vertex computation persisted and have been experienced by those who needed such computations. A few computational problems were also discovered during the use of the Geometric Bounding Toolbox [13, 4] between 1993-1999. Polytope computation in GBT Versions 1.0-6.1 was based on state of the art algorithms published in the late 80s and early 90s [9, 14, 11] which did not offer consistent numerical solution. GBT 7.1 was the first version of the toolbox which implemented logically consistent numerical algorithms.
1.2 List of Routines

The following lists the MATLAB-object-free routines of the toolbox as on 18.02.2004. More routines are constantly added to GBT and they can be checked at www.sysbrain.com/gbt/gbtlist7.htm

**Routines for polytope creation**
- `convh` - computes the convex hull of a set of points
- `fconvh` - computes the intersection of a set of half-spaces
- `defbox` - generates an axis aligned box
- `defpipe` - defines a parallelepiped
- `def_nps` - defines a nonconvex polyhedral set (NPS)
- `defsimp` - defines a regular centred simplex with edge size 1
- `ellapprx` - computes a polytope approximation to a given ellipsoid

**Characteristics of polytopes**
- `dim` - computes the dimension of a polytope
- `ranges` - computes a tight axis aligned box around a polytope
- `diameter` - computes the diameter of a polytope
- `dirext` - computes extremal points in a given direction
- `maxface` - computes the maximum possible number of facets
- `maxvert` - computes the maximum possible number of vertices
- `polcent` - computes the centre of a polytope
- `polvol` - computes the volume of a polytope
- `polvol_` - computes the volume of an NPS
- `surfarea` - computes the surface area of a polytope

**Polytope operations**
- `intersct` - computes the intersection of two polytopes
- `intersc_` - computes the intersection of two NPSs
- `union_` - computes the union of two NPSs
- `ltran` - performs affine transformation on a polytope
- `ltran_` - performs affine transformation on an NPS
- `poladd` - computes the sum of two polytopes
- `poladdi` - computes internal approximation to the sum of two polytopes
- `poldual` - computes the dual of a polytope
- `mindim` - minimizes the dimension of representation
- `subtract` - computes the Minkowski difference of two polytopes

**Extracting features from polytopes**
- `faces` - extracts the half-space inequalities of polytope
- `facet` - computes a polytope of facet in lower dimension
- `fv` - produces the facet-vertex Boolean adjacency-table
- `vvf` - produces vertex-vertex and vertex-facet adjacency-tables
  (in vvf 2 is a memory sparing version)
- `fvfitacc` - extracts maximum fitting error of vertices and facets
- `planeigh` - lists all neighbouring facet planes to a facet
1.2. **LIST OF ROUTINES**

*Interaction of polytopes with points and half-spaces*
- **p_conv**: computes point inclusion into convex hull
- **intest**: tests whether a set of points is in polytope
- **intpoint**: finds an internal point within a polytope
- **projpnt**: projects a point onto polytope
- **projpol**: projects a polytope onto an affine subspace
- **update**: computes the intersection of a half-space and a polytope
- **slicegeb**: computes intersection of a polytope and a hyperplane

*Polyhedral graphics*
- **view2d**: displays 2D wire-frame projections of a polytope onto pairs of axes
- **view2d**: displays 2D wire-frame projections of an NPS onto pairs of axes
- **view3d**: displays 3D view of 3D polytope

*Ellipsoid creation*
- **defell**: defines an ellipsoid
- **sel**: finds minimal volume ellipsoid around a set of points

*Cylinder creation*
- **defcyl**: defines a cylinder
- **slice cyl**: computes a slice of a cylinder

*Ellipsoid operations*
- **elladd**: computes an outer bounding ellipsoid to the sum of two ellipsoids
- **elladde**: computes outer bounding ellipsoid-variants to the sum of two ellipsoids
- **elladdi**: computes inner bounding ellipsoid to the sum of two ellipsoids
- **elldife**: computes an outer bounding ellipsoid to the difference of two ellipsoids
- **elldiff**: computes an inner bounding ellipsoid to the difference of two ellipsoids
- **ellint**: computes an ellipsoid around the intersection of two ellipsoids
- **ellintc**: computes the minimum-volume convex combination of two ellipsoids
- **projell**: computes the projection of an ellipsoid onto a linear subspace
- ** sphsep**: computes the joining surface area of spheres
- **elltrans**: computes the affine transform of an ellipsoid

*Characteristics of ellipsoids*
- **ellrange**: computes the width (range) of ellipsoid in each dimension
- **ellvolum**: calculates the volume of ellipsoid
- **ellcent**: extracts the centre of ellipsoid
- **elicov**: extracts the matrix of an ellipsoid
- **inell**: tests whether a given point is in an ellipsoid
- **mcvol**: estimates the volume of the intersection of ellipsoids
- **ellrdrxt**: computes extremal points of an ellipsoid in a given direction

*Ellipsoid graphics*
- **view2del**: displays 2D projection on axes pairs
- **view3del**: displays 3D view of an ellipsoid
**CHAPTER 1. INTRODUCTION**

*Point and vector set operations*

- **dif** - computes set-difference of vector sets
- **dirgen** - generates an approximately uniform set of direction vectors
- **dualpon** - computes a set of half-spaces "normal" to set of vectors
- **uni** - computes the union of two point-sets
- **vreduce** - eliminates multiple points from array of points
- **diamet** - computes diameter of a set of points
- **findsimp** - finds the vertices of the largest simplex from a set of points

*Routines for compatibility with GBT 6.1*

- **bounded** - tests whether a polyhedron is bounded
- **gbt627** - converts from GBT 6.1 formats to GBT 7.2 format
- **gbt726** - converts from GBT 7.2 format to type 4 format of GBT 6.1
The following is a list of most MATLAB object free m-files in GBT in alphabetical order:

**BOUNDED.M** - test whether a polyhedron is bounded  
**COMBIN.M** - compute the k-class combination of n objects  
**CONVH.M** - compute convex hull of points  
**DEFBOX.M** - generate an axis aligned box  
**DEFCYL.M** - define a cylinder  
**DEFELL.M** - define an ellipsoid  
**DEFPIPE.M** - define a parallelepiped  
**DEF_NPS.M** - define an NPS  
**DEFSIMP.M** - define a regular centred simplex with edge size 1  
**DEMO1.M** - demo generates a random set of points for dimensions 2-4  
**DEMO10.M** - demo simulates 5 random sets of 5 spheres  
**DEMO11-23.M** - various demos of GBT features  
**DEMO2.M** - demo tests the worst-case fitness ratio for half-space  
**DEMO3.M** - demo tests the worst-case fitness ratio for convex hull  
**DEMO4.M** - demo m-file adds three polytopes: two boxes and a random  
**DEMO5.M** - demo illustrates the sensitivity of facet updating.  
**DEMO6.M** - demo generates 3 random 3D ellipsoids  
**DEMO7.M** - demo computes a maximal polytope ‘inside’ a set of points (not optimal procedure).  
**DEMO8.M** - demo m-file estimates the volume of the intersection of  
**DEMO9.M** - demo tests the accuracy of the PLANFIT routine.  
**DIAMET.M** - compute the diameter of a set of points  
**DIAMETER.M** - compute the diameter of a polytope  
**DIF.M** - compute the set-difference of vector sets  
**DIM.M** - compute the dimension of a polytope  
**DIREXT.M** - compute extremal points in a given direction  
**DIRGEN.M** - generate an approximately uniform set of direction-vectors  
**DUALPON.M** - compute a set of half-spaces “normal” to a set of vectors  
**ELLADDE.M** - compute outer bounding ellipsoid to the sum of two ellipsoids.  
**ELLADDI.M** - compute inner bounding ellipsoid to the sum of two ellipsoids.  
**ELLAPPRX.M** - compute a polytope approximation to a given ellipsoid  
**ELLCENT.M** - extract the centre of an ellipsoid  
**ELLDIFE.M** - compute an outer bounding ellipsoid to the difference of two ellipsoids.  
**ELLDIFI.M** - compute an inner bounding ellipsoid to the difference of two ellipsoids.  
**ELLDRXT.M** - compute extremal points of an ellipsoid in given direction  
**ELLICOV.M** - extracts the matrix of an ellipsoid.  
**ELLINT.M** - compute an ellipsoid around the intersection of two ellipsoids  
**ELLINTEC.M** - compute minimum convex combination of two ellipsoids  
**ELLRANGE.M** - compute the width (range) of ellipsoid in each dimension  
**ELLTRAN.M** - compute the affine transform of an ellipsoid  
**FACES.M** - extract half-space inequalities of polytope
CHAPTER 1. INTRODUCTION

- compute the polytope of a facet in lower dimension
- compute the intersection of a polytope and a set of half-spaces
- find largest \((d-1)\)-dimensional simplex from a set of points
- combine points near the first point in a list
- produce facet-vertex Boolean table of adjacency
- extract the maximum fitting error of vertices and facets
- calculate subgradient of cost for LPCENT
- convert from GBT 6.1 formats to GBT 7.0 format
- convert GBT 7 format to type 4 format of GBT 6.1
- test whether a given point is in an ellipsoid
- find a maximal volume ellipsoid within a polytope
- compute the intersection of two polytopes
- compute the intersection of two NPSs
- test whether a set of points is in polytope
- find an internal point within a polytope
- calculate the lp-centre of a set of points
- perform affine transformation on a polytope
- compute the vector of maximum no. faces for given number of vertices
- compute the vector of maximum no. vertices for given number of facets
- estimate the volume of the intersection of ellipsoids
- minimize the dimension of the representation of a polytope
- list neighbouring facet planes
- fit a hyperplane to a set of points
- compute the sum of two polytopes
- compute diamond inside the sum of two polytopes
- compute the centre of a polytope
- compute the dual of a polytope
- compute the volume of a polytope
- compute the volume of an NPS
- project ellipsoid onto subspace
- project polytope onto linear manifold
- project point onto polytope
- include a point into a convex hull polytope
- compute the lower and upper corners of a tight axis aligned box to polytope
- find the minimal volume ellipsoid around a set of points
- is a script M-file not a function.
- compute the intersection of a polytope and a hyperplane
- compute the intersection of a cylinder and an axis aligned hyperplane in 3D
- compute the joining surface area of spheres
- compute the Minkowski difference of two polytopes
- compute the surface area of a polytope
- compute the union of two point sets
1.2. LIST OF ROUTINES

UNION.M - compute the union of two NPSs
UPDATE.M - compute the intersection of a half-space and a polytope
VERTICES.M - produce the list of vertices of a polytope
VIEW2D.M - display 2D wire-frame projections of a polytope onto pairs of axes
VIEW2D_M - display 2D wire-frame projections of an NPS onto pairs of axes
VIEW2DEL.M - display 2D views of ellipsoids
VIEW3D.M - display 3D view of 3D polytope
VIEW3DEL.M - display 2D views of ellipsoids
VREDUCE.M - eliminate multiple points from array of points
VTXFIT.M - fit a point to d hyperplanes
VVF.M - produce vertex-vertex and vertex-facet adjacency tables (version no. 1)
VVF_2.M - produce vertex-vertex and vertex-facet adj tables (version no. 2)
WARNDEL.M - m-file gives error message about the violation of
WARNKAP.M - m-file gives error message about the violation of
Table 1.1: MATLAB object classes and currently available associated methods

<table>
<thead>
<tr>
<th>MATLAB class</th>
<th>Methods</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>gbtcomplex</td>
<td>gbtcomplex, set, get, display</td>
<td>parent of nps and physical_object</td>
</tr>
<tr>
<td>covariance</td>
<td>covariance, set, get, display</td>
<td>-</td>
</tr>
<tr>
<td>ellipsoid</td>
<td>ellipsoid, set, get, display</td>
<td>child of convex_object</td>
</tr>
<tr>
<td>half_spaces</td>
<td>half_spaces, set, get, display</td>
<td>child of convex_object</td>
</tr>
<tr>
<td>hyperplanes</td>
<td>hyperplanes, set, get, display</td>
<td>child of convex_object</td>
</tr>
<tr>
<td>physical_object</td>
<td>physical_object, set, get, display</td>
<td>child of gbtcomplex</td>
</tr>
<tr>
<td>points</td>
<td>points, set, get, display</td>
<td>child of non_convex_object</td>
</tr>
<tr>
<td>polytope</td>
<td>polytope, set, get, display, +, intersect</td>
<td>child of non_convex_object</td>
</tr>
<tr>
<td>cylinder</td>
<td>cylinder, set, get, display</td>
<td>child of convex_object</td>
</tr>
<tr>
<td>nps</td>
<td>nps, set, get, display</td>
<td>child of gbtcomplex</td>
</tr>
</tbody>
</table>

MATLAB object representations of numerical geometric objects are also available. Although these cannot be compiled into C, C++ and executable programs, they simplify programming under the MATLAB environment considerably. The following table lists the MATLAB object types and associated methods. The methods in their current form are not comprehensive (as on 16.3.2004). Note however that you can easily extend the set of methods for each class by the powerful set of numerical routines available (free of MATLAB objects) in GBT.

A MATLAB object "gbtcomplex" is essentially a set of polytopes, ellipsoids or cylinders which are connected by the set operations of union and set difference.

A MATLAB object "physical_object" is a set of complexes with associated lists of physical properties for each complex. Normally each complex is a homogenous material for which physical properties such as specific weight, material descriptions can be listed in a structure. More MATLAB objects are described in Chapter 2.
Chapter 2

Standard Object Representations

The toolbox uses a polytope representation that makes the best compromise between detailed description and simplicity in computations involving geometric objects. The choice was greatly influenced by the long-term experience gained with GBT [13] and the feedback received from many users of that toolbox.

Because of its importance to the user, and for sake of clarity, we present here the matrix form chosen to represent a polytope in GBT 7.0, which is also called type 0 representation in GBT. Those familiar with earlier versions of GBT may know about type 2 and type 4 polytope representations. type 0 stores the least amount of data from all representations.

2.1 Polytopes

The standard form of a polytope in the $d$-dimensional Euclidean space is defined by a matrix the first row of which consists of data; the next set of rows consists of equalities defining the linear manifold in which the polytope resides and is followed by a set of inequalities defining the polytope in the manifold; the final set of rows specify the vertices of the polytope. Thus the GBT representation has the form:

$$P = \begin{bmatrix}
    n_e & n_i & \mu & 0 & \cdots & 0 \\
    1st & affine & subspace & & & \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \\
    n_e,th & affine & subspace & & & \\
    1st & facet & inequality & & & \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \\
    n_i,th & facet & inequality & 0 & & \\
    1st & vertex & vector & -1 & & \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \\
    n_v,th & vertex & vector & -1 & 
\end{bmatrix}$$

(2.1)

In the first row, $n_e$ is the number of equalities and $n_i$ the number of inequalities; the number of vertices $n_v$ is therefore given by $n_v = n - 1 - n_e - n_i$, where $n$ is the
number of rows in $P$. The third component of the first row, $\mu > 0$, is the worst-case fitting error of polytope $P$:

**Definition 2.1.1** For a facet $f$ (defined by $a^T x = b$, $\|a\| = 1$) and vertex $v$ the quantity $\Delta(f, v) = |a^T v - b|$ is called the **fitting precision**. A bound $\delta > 0$ is called the **adjacency tolerance** of a polytope if all its fitting precisions are less than $\delta$.

**An example**

Using standard operations a random polytope was generated in GBT by the following commands:

```matlab
>> V=rand(5,3);
>> P=convh(V)
```

The P matrix is displayed in the command window:

```
P =
    0    6.0000    0.0000    0
   -0.6373    0.3434   -0.6899   -0.5369
   -0.0934    0.2086    0.9735    0.8446
   -0.5369   -0.6277   -0.5637   -0.8570
   -0.0497    0.8126   -0.5806    0.2147
    0.2775    0.2537    0.9266    1.0273
    0.9197   -0.3663    0.1415    0.6818
    0.2311    0.4565    0.7919   -1.0000
    0.6068    0.0185    0.9218   -1.0000
    0.8913    0.4447    0.1763   -1.0000
    0.4860    0.8214    0.7382   -1.0000
    0.9501    0.7621    0.6154   -1.0000
```

Closer examination of the $P(1,3)$ element reveals that it is not actually zero:

```matlab
>>P(1,3)
```

```
an =
     1.1102e-015
```

The $P(1,3)$ entry is the calculated accuracy bound of the vertex-facet adjacency (the fitting precision) in this numerical representation of the polytope. All entries displayed in the command window are given to four decimal points; their internal representations are more precise.

What else can we read from this matrix by inspection?

- As the representation matrix has 4 columns, the maximum dimension of the polytope is 3.
- $P(1,1)=0$ indicates that the polytope is full dimensional (no equality constraints) which in this case means that the polytope is non-degenerate in 3D.
2.2. NPS REPRESENTATION

- \( P(1,2)=6 \) indicates the number of 2-dimensional facets on the boundary of the polytope. The linear forms associated with each facet are listed in rows 2-7.
- As the total number of rows is in the matrix representation is 12, the number of vertices is 12-1-6=5. By convention all vertex rows are ended by a -1.

Note that the linear forms for each facet are defined by normalised vectors, this can for instance be verified by the command:

```matlab
>> diag(P(2:7,1:3)*P(2:7,1:3)')
ans =
1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
```

By definition, -1 appears in the last column of each row of the vertex list; this convention enables vertices to be easily identified and saves time in some computations.

2.2 NPS representation

The concept of the nonconvex polyhedral set, i.e. NPS, has been defined in Definition 1.1.4. Each NPS is represented by a two-column cell array: the first column contains polytope representations as described above, the second column contains ‘+’ or ‘-’ signs to indicate whether the polytope is in the \( S_1 \) or \( S_2 \) set in the representation:

\[
N = \bigcup_{i=1}^{n_1} P_i^+ \setminus \bigcup_{i=1}^{n_2} P_i^-
\] (2.2)

where \( S_1 = \{ P_i^+ | i = 1, \ldots, n_1 \} \) and \( S_2 = \{ P_i^- | i = 1, \ldots, n_2 \} \). The polytopes in \( S_1 \) are referred to as the \( +\)polytopes and the polytopes in \( S_2 \) as the \( -\)polytopes of the representation.

For instance let \( p_1, p_2 \) and \( p_3 \) be three polytopes. The routine `def_nps` can be used to define an NPS which represents \((p_1 \cup p_2) \setminus p_3\):

```matlab
>> def_nps({p1,p2},{p3})
ans =
[ 9x3 double]     '+'
[ 9x3 double]     '+'
[11x3 double]    '-'
```

The NPS format is recognized by routines with names ending with \_nps, so operations of union, intersection, volume and other computations can be carried out with NPSs. NPS is a special case of the ”complex representation” as described below.

2.3 Ellipsoid representation

Let \( P \) be a positive definite \( d \times d \) symmetric matrix and \( c \in \mathbb{R}^d \). Then the set

\[
E = \{ \ x \mid (x - c)^T P^{-1} (x - c) \leq 1 \ \}
\] (2.3)
is called a \textit{d–dimensional ellipsoid}. For singular \( P \) a degenerate ellipsoid is defined by
\[
E_{\text{deg}} = \{ x \mid (x - c)^T P^{-1} (x - c) \leq 1 \}
\] (2.4)
where \( P^{-1} \) denotes the Moore-Penrose semi-inverse of \( P \).

The matrix representation of an ellipsoid in MATLAB is
\[
P^{\text{rep}} = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
P_{11} & \cdots & P_{1d} & c_1 \\
P_{21} & \cdots & P_{2d} & c_2 \\
\vdots & \vdots & \vdots & \vdots \\
P_{d1} & \cdots & P_{dd} & c_d
\end{bmatrix}
\] (2.5)

The 1 in the first row indicates that the ellipsoid is full dimensional. If \( P^{\text{rep}}_{11} \) is 0 then by definition the ellipsoid is degenerate with a singular \( P \). If \( n = P^{\text{rep}}_{11} \geq 2 \) then the second row down to the \( n \)th row define equations for affine subspaces in which the lower dimensional ellipsoid lies. From row \( n + 1 \) down to the last row of \( P^{\text{rep}} \) the matrix \( P \) and centre \( c \), as in the case of the full dimensional matrix, are listed.

\section*{2.4 Cylinder representation}

A cylinder is represented by its central symmetry axis \( a \), which is given as a straight infinite line, and for radius \( r \) the infinite cylinder if given by
\[
C = \{ x \mid \| x - \text{Proj}(x, a) \| \leq r^2 \}
\] (2.6)

Here \( a \) is of the form
\[
a = \{ x = v_0 + v_1 \lambda \mid \lambda \in \mathbb{R} \}
\] (2.7)

where \( v_0 \) is some point on \( a \) and \( v_1 \) is a direction vector for the line. \( \text{Proj}(x, a) \) denoted the orthogonal projection of point \( x \) onto line \( ^*a \).

A bounded cylinder can be defined as the intersection of two half-spaces and an infinite cylinder. So the structure of a cylinder representation is
\[
P = \begin{bmatrix}
2 & 0 & \cdots & 0 \\
v_0 & \text{vector} & 0 \\
v_1 & \text{vector} & 0 \\
1\text{st half-space inequality} \\
2\text{nd half-space inequality}
\end{bmatrix}
\] (2.8)

This representation is minimum 2 dimensional, so that the above matrix has at least 3 columns. If the representation has only 3 rows than the cylinder is double infinite, otherwise with 4 rows it can be semi-infinite and it can be bounded with 5 rows. Later cylinder type is represented by the \textit{cylinder} class under MATLAB, the above representation is its \"value\" property. (See demo23.m for an example.)
2.5 Complex objects

A generalization of the above NPS can be defined by allowing the first column of the cell array to represent any of a polytope, ellipsoid, nps or cylinder and second column is used to denote whether the inside or the outside of the complex object should be taken into the gbtcomplex, for instance

\[
\{ \text{P1 }^+, \text{P2 }^+, \text{P3 }^-, \text{E1 }^+, \text{C1 }^-, \text{C2 }^+ \}
\]

where P1-P3 are polytopes and E1 is an ellipsoid and C1, C2 are two cylinders. This type of geometric object representation is not object oriented and this representation does not fit into a class hierarchy. Note however that within GBT MATLAB objects the class \texttt{gbtcomplex} class is a parent of \texttt{nps} and \texttt{physical_object} as described in the next section.

2.6 Physical Objects

Physical objects are normally composed of parts each of which can be approximated as some homogenous material. Each of these homogenous parts can now be modelled as a complex object. A physical object is a cell array that contains a list of complex objects in its first column and in a second columns it contains physical characteristics of the part. The second column is used define the specific mass, a material description, temperature if known, permittivity, conductivity, electric potential, etc. The format of this representation of \texttt{physical_object} is

\[
\{ \text{Cx1 }'\text{structure of physical descriptions}' \text{Cx2 }'\text{structure of physical descriptions}' \text{Cx3 }'\text{structure of physical descriptions}' : : : : \text{Cxn }'\text{structure of physical descriptions}' \}
\]

where \textit{structure of physical descriptions} is a structure with fieldnames as physical characteristics such as \texttt{density}, \texttt{temperature}, \texttt{potential}, \texttt{material}, etc. The content of the fields are physical quantities with numbers and physical dimensions in "char" type, i.e. text format, for instance the "density" field could be '7482 kg/m$^3$', "potential" could be '0.1 V ', the "temperature" field could be the string '50 F ', etc.

2.7 MATLAB Objects in GBT

2.7.1 Object classes

This section assumes that the reader is familiar with MATLAB object oriented programming to some degree, as it is described in the MATLAB manual. MATLAB
objects of GBT geometric objects are useful when one wants to work in the MATLAB environment. It greatly simplifies the programming effort by encapsulation of object properties and methods.

The geometric object representations in the previous sections are free of MATLAB objects and therefore their operations can be compiled into C,C++ and executable code. In the GBT MATLAB objects each object has the "value" property which is identical to the object representation as described above, i.e. without MATLAB objects.

GBT MATLAB objects, as outlined in this section, use the above representations for the "value" of the object but in addition they allow for properties that are dependent on the "value" and can be interdependent. It is therefore important that methods acting on objects take into account the inherent dependence between properties and produce objects consistent with these. "value" is the minimal core representation of a GBT MATLAB object and in any new operation it is advisable to obtain this one first and then to derive the rest of the properties ("derived properties" in the code) from the "value" property. This technique enforces a valuable systematic approach to geometric operations on objects in order to avoid inconsistencies of properties in geometric objects instances.

Some properties of GBT MATLAB objects are described in the table below. These property lists can be extended according to the needs of GBT users. Note however that within one MATLAB session the set and order of properties of any class must remain constant. After changing property lists MATLAB should be restarted.

In GBT all object classes have four properties: value, class, type and dimension. The property "value" is the non-object representation of the geometric body as in the object free GBT. The property "class" is an indicator of the GBT object class, i.e. one of polytope, nps, hyperplane, half_space, point, ellipsoid, cylinder, gbtcomplex, physical_object. The property "type" allows to name a subclass of of a "class", for instance a type of a polytope can be "simplex" or "cube", a type of an ellipsoid can be "sphere". The content of the type property is used by the methods that can handle the object. If the type of an object is not empty then it normally imposes some constraint on the "value" property and the rest of the properties of the GBT MATLAB object.

Why to use the "value" property? The main reason is that "value" is a conventional object representation in GBT and as such it contains a reasonably parsimonious set of independent parameters that define the geometric object. Hence all the rest of the parameters are derivable from the content of the value property. Normally there are several different sets of properties that could determine a geometric object (for instance a polytope is determined by its set of vertices and set of facet inequalities as well). The interdependence of object properties is quite complex. To avoid logical inconsistencies and to simplify procedures, the "value" is used as a core object representation. When methods compute a new GBT MATLAB object then the value property is computed first, and then the rest of the properties are derived.

Fig. 2.1 displays the current object hierarchies that are open for extension. The current version of GBT with objects only contains a limited set of methods: constructors, set, get, display routines. Using the object architecture of GBT, any user can produce new methods by applying the standard set of routines of the non-object GBT.
### Table 2.1: MATLAB object classes and their properties

<table>
<thead>
<tr>
<th>MATLAB class</th>
<th>Property : property type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>hyperplanes</td>
<td>value:double, class:char, type:char, number_of_equalities:double, list:double, dimension:double</td>
<td>parent object is convex_object</td>
</tr>
<tr>
<td>half_spaces</td>
<td>value:double, class:char, type:char, number_of_inequalities:double, list:double, dimension:double</td>
<td>parent object is convex_object</td>
</tr>
<tr>
<td>points</td>
<td>value:double, class:char, type:char, dimension:double, diameter:double, diameter_endpoints:double</td>
<td>parent object is non_convex_object</td>
</tr>
<tr>
<td>ellipsoid</td>
<td>value:double, class:char, type:char, centre:double, matrix:double, covariance:double, diameter:double, smallest_diameter:double, axis_directions:double, dimension:double</td>
<td>parent class is convex_object</td>
</tr>
<tr>
<td>cylinder</td>
<td>value:double, class:char, type:char, dimension:double, axis_point:double, axis_direction:double, bounding_plane1:double, bounding_plane2:double</td>
<td>parent class is convex_object</td>
</tr>
<tr>
<td>covariance</td>
<td>value:double, class:char, type:char, dimension:double, singular_values:double, orthogonal_transform:double, maximum_singular_value:double, minimum_singular_value:double</td>
<td></td>
</tr>
<tr>
<td>physical_object</td>
<td>value:double, class:char, type:char, dimension:double, list_of_physical_properties:cell, list_of_objects:double, list_of_objects:double</td>
<td>parent class is non_convex_object</td>
</tr>
</tbody>
</table>
2.7.2 Overview of methods

This section reviews some of the features of the basic methods for each object class.

Methods for "polytope" objects

The constructor polytope accepts various kind of inputs in any order.

- If any of the strings 'cube', 'simplex' occurs in the input list then the first 'double' type input will be the dimension of the object. The second double scalar input is the edge size of the object. The default dimension is 3, the default edge size is 1.

Examples:

\[
\text{P=polytope}(1.45,3,\text{'cube'}); \quad \text{P=polytope}(\text{'simplex'},3); \quad \text{P=polytope}(\text{'cube'});
\]

\[
\text{P=polytope}(2,\text{'simplex'});
\]

- If 'box' occurs in the input list then two numerical vectors are needed to be defined among the inputs for the lower and upper corner of the box.

Examples:

\[
\text{P=polytope}(\text{'box'},[1 \ -1 \ 1.3],[2 \ 2.1 \ 3]); \quad \text{P=polytope}([1 \ -1 \ 1.3],[2 \ 2.1 \ 3],\text{'box'});
\]
Table 2.2: MATLAB object classes and currently available associated methods

<table>
<thead>
<tr>
<th>MATLAB class</th>
<th>Methods</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>complex</td>
<td>complex, set, get, display</td>
<td>display is graphical</td>
</tr>
<tr>
<td>covariance</td>
<td>covariance, set, get, display</td>
<td>display is numerical</td>
</tr>
<tr>
<td>ellipsoid</td>
<td>ellipsoid, set, get, display</td>
<td>display is graphical</td>
</tr>
<tr>
<td>half_spaces</td>
<td>half_spaces, set, get, display</td>
<td>display is numerical</td>
</tr>
<tr>
<td>hyperplanes</td>
<td>hyperplanes, set, get, display</td>
<td>display is numerical</td>
</tr>
<tr>
<td>physical_object</td>
<td>physical_object, set, get, display</td>
<td>display is graphical</td>
</tr>
<tr>
<td>points</td>
<td>points, set, get, display</td>
<td>display is numerical</td>
</tr>
<tr>
<td>polytope</td>
<td>polytope, set, get, display, +, intersect</td>
<td>display is graphical</td>
</tr>
<tr>
<td>convex_object</td>
<td>convex_object, set, get, display</td>
<td>display is numerical</td>
</tr>
<tr>
<td>non_convex_object</td>
<td>non_convex_object, set, get, display</td>
<td>display is numerical</td>
</tr>
</tbody>
</table>

- If the string 'convex hull' is in the input list with a matrix of row vectors then these are interpreted as a list of points and their convex hull is computed to obtain the polytope.

Examples:

\[
P = \text{polytope}(\text{rand}(10,3),\text{’convex hull’}); P = \text{polytope}(\text{’convex hull’}, \text{rand}(10,2));
\]

- If the string 'half spaces' is in the input list with a matrix of row vectors then these are interpreted as a list of inequalities each defining a half-space and the their intersection is computed to obtain the polytope.

Example:

\[
H = \text{half\_spaces}(\text{rand}(20,3)); P = \text{polytope}(H);
\]

- If the input is a single object that is of type 'points' or 'hyperplanes', 'half spaces', then the corresponding convex hull, and intersections are computed, respectively.

Examples:

\[
H = \text{half\_spaces}(\text{rand}(20,3)); P = \text{polytope}(H); V = \text{points}(\text{rand}(20,3)); P = \text{polytope}(V);
\]

- If the input is a single object that has type any of 'nps', 'complex', 'physical object', then that is converted into a polytope as long as a single polytope description is found in the object representation.

Standard methods are `display`, `get`, `set` operate in the usual way. Note that if no semicolon is used after an assignment statement that has a polytope object on its right hand side, then a figure will display the 2D or 3D view of the object (for other dimensions numerical array is displayed).

The "+" operation is defined in the "plus.m" method. The Minkowski sum of two polytope objects \( P_1, P_2 \) can be obtained by \( P = P_1 + P_2 \).

The intersection of two polytope objects can be obtained by the "intersect.m" method, example \( P = \text{intersect}(P_1, P_2) \).

A range of other object operations (for instance Pontryagin difference, rotations, affine transforms, etc) can be defined using the non-object-based GBT routines.
Methods for "points" objects

Constructor of a points object can do the following:

- If the single input variable is a matrix (class 'double') then the set of points will consist of the points defined by the row vectors of the matrix.
  Example: \( P = \text{rand}(10,3); p = \text{points}(P); \)

- If the single input variable is a polytope then the set of vertices is formed for the set of points.
  Example:
  \( P = \text{convh(rand}(10,3)); p = \text{points}(P); \)

- If there are several input variables then the union of points is take for the outputs set of points.
  Example:
  \( P_1 = \text{convh(rand}(10,3)); P_2 = \text{rand}(12,3); p = \text{points}(P_1, P_2); \)

Methods for "ellipsoid" objects

Constructor of a ellipsoid object can do the following:

- If the single input variable is a covariance object then an ellipsoid object with centre in the origin is defined.
  Example:
  \( A = \text{rand}(4,4); c = A' \cdot A; C = \text{covariance}(c); E = \text{ellipsoid}(C); \)

- If the single input variable is a positive definite squared matrix then object that is used for a covariance matrix with centre at the origin to define the output ellipsoid.
  Example:
  \( c = \text{rand}(1,4); A = \text{rand}(4,4); C = A' \cdot A; E = \text{ellipsoid}(C, c); \)

- If from two input variables one is a covariance object and the other variable is a double vector of the same dimension, then the latter is used for the centre of the ellipsoid.
  Example:
  \( A = \text{rand}(4,4); c = A' \cdot A; C = \text{covariance}(c); E = \text{ellipsoid}(C, \text{ones}(1,4)); \)

The standard methods display, get, set operate in the usual way. If semicolon is omitted after a 2D or 3D ellipsoid object, it will be plotted or listed in the MATLAB command window.
2.7. MATLAB OBJECTS IN GBT

Methods for "non-convex polyhedral set" (NPS) objects

Constructor of an \textit{NPS} object can do the following:

- If the two input variables are each cell arrays (sets), then the first set will be the + polytopes and the second set will be the - polytopes in the NPS defined. Each set must be a set (cell array of maximum dimension 2) of polytope objects or raw GBT polytope formats.

  Example:
  
  \begin{verbatim}
  P1=convh(rand(10,3)); P2=polytope(rand(20,3),'convex hull'); P3=polytope('cube',0.2);
  Nps=nps(P1,P2,P3);
  \end{verbatim}

- If the single input variable is a cell array (set) with each entry defining a polytope object, then only + polytopes will occur in the NPS.

  Example:
  
  \begin{verbatim}
  P=; for k=1:10, p=convh(rand(5,2));P=[P,p];end;Nps=nps(P);
  \end{verbatim}

\texttt{get, set} operate in the usual way. If semicolon is omitted after a 2D or 3D nps object, it will be plotted or listed in the MATLAB command window. The "display" does not yet plot the negative objects in this version, later versions will have this, but the user can easily write a routine for that.

Methods for "cylinder" objects

Constructor of a \textit{cylinder} object can do the following:

- If two double vectors and a positive scalar is defined in the input variables then the first vector is the direction of the cylinder axis and the second one is a point on the axis. The scalar will be the radius.

\begin{verbatim}
  P1=convh(rand(10,3)); P2=polytope(rand(20,3),'convex hull'); P3=polytope('cube',0.2);
  Nps=nps(P1,P2,P3);
  P=; for k=1:10, p=convh(rand(5,2));P=[P,p];end;Nps=nps(P);
\end{verbatim}
CHAPTER 2. STANDARD OBJECT REPRESENTATIONS

- If one double vector and a positive scalar is defined in the input variables then the vector is the direction of the cylinder axis going through the origin and the second one is a point on the axis.

Display method \texttt{display} only works for finite cylinders. The \texttt{slice} routine only works for axis aligned hyperplane slices for finite cylinders. The following figure 2.3 displays the 3D view of a cylinder object with a slice contour which is the result of demo23.m on the disk.

![Figure 2.3: A cylinder object with a slice contour indicated.](image)

Methods for "half spaces" objects

Constructor of a \texttt{half-spaces} object can do the following:

- For a single input variable matrix of MATLAB class double, the dimension of the half spaces is the number of columns minus 1. Each row of the matrix represents a linear inequality to define a half-space. A matrix row \([a_1, a_2, \ldots, a_n, b]\) means the half-space

\[
a_1x_1 + a_2x_2 + \ldots + a_n x_n \leq b
\]  

(2.9)

Example: \texttt{H=half\_spaces(rand(10,3));}.

The standard methods of \texttt{set}, \texttt{get} are functional but \texttt{display} only provides a numerical listing of the "value" (see above).
2.7. MATLAB OBJECTS IN GBT

Methods for ”covariance“ objects

Constructor of a covariance object can do the following:

- If the single input variable is a matrix of type double then it has to be a positive semi definite one to be able to define a covariance object.

Example: 

\[
A = \text{rand}(4,4); \quad c = A' \cdot A; \quad C = \text{covariance}(c);
\]

The standard methods of set, get are functional but display only provides a numerical listing of the ”value“ (see above).

Methods for ”hyperplanes“ objects

Constructor of a hyperplanes object can do the following:

- For a single input variable matrix of MATLAB class double, the dimension of the hyperplanes is the number of columns minus 1. Each row of the matrix represents a linear equality to define a hyperplane. A matrix row \([a_1, a_2, \ldots, a_n, b]\) means the hyperplane

\[
a_1x_1 + a_2x_2 + \ldots + a_nx_n = b
\]

(2.10)

The standard methods of set, get are functional but display only provides a numerical listing of the ”value“ (see above).

Methods for ”complex“ objects

Constructor of a gbtcomplex object can do the following:

- The inputs of the constructor must be of object class ”polytope”, ”ellipsoid”, ”cylinder”, ”nps” or a cell array with all entries being one of these four types. If any of these four types occur as an input variable then that input is taken as a ”+“ objects. (See the definition of a complex in the previous section.)

- The first cell array entries are taken as ”+“ objects and entries of the second cell array input to the constructor are taken as ”-“ objects in the complex. Any individual (non cell element) object input is taken as a ”+“ object in the complex.

The standard methods of set, get work in the usual way and display plots and object without the negative polytopes taken into account (future versions will amend this).

Methods for ”physical objects“

Constructor of a ”physical object“ object can do the following:

- Each input variable to the constructor must be a cell vector of geometric objects in object free format or member of the classes gbtcomplex, nps, polytope, ellipsoid or cylinder. The associated physical quantities can be filled in after
construction. In the second cell column of the physical properties are contained in a structure with fieldnames for the physical properties (using underlines for spaces). For instance, if \texttt{Py} is physical object then a possible model is:

\begin{verbatim}
>> class(Py)
ans =
physical_object

>> v=get(Py,'value')

v =

[18x4 double] [1x1 struct]
[18x4 double] [1x1 struct]
[18x4 double] [1x1 struct]

>> v{1,2}

ans =

density: '1782 kg/m^3'
material: 'Al201 alloy'
potential: '0.1 V'
temperature: '50 F'
\end{verbatim}

Physical object properties can be manipulated by new methods developed by the user, GBT provides a standard format of object representations.

The standard methods of \texttt{set}, \texttt{get} work in the usual way and \texttt{display} plots and object without the negative polytopes taken into account (future versions will amend this). The "physical quantity" objects are left open for the user of GBT to use to their purpose. This can be done by agreeing on a list of properties and developing methods which handle the object by making use of these.
Chapter 3

Error control and speed of computation

3.1 Global tolerance $\delta$ and local precision $\mu$

Computational precision in GBT is a measure of the distance of vertices from the facet in which they should lie. Fitting precision is an absolute error as opposed to the floating point relative error.

The equation of the hyperplane $H$ containing the facet $F$ is:

$$H: \quad xa^T = b$$

A vertex on facet $F$ is given by a row vector:

$$v = [v_1, v_2, \ldots, v_n]$$

The fitting precision of $v$ to $F$ (and $H$) is defined to be:

$$\mu = |va^T - b|$$

For a given polytope $P$, the largest fitting error of vertices to facets is denoted by $\mu(P)$.

In the type 0 polytope representation $P$, the fitting precision $\mu(P)$ is stored in the $P(1, 3)$ entry for dimension $d > 2$ and in the $P(1, 2)$ entry for $d = 2$.

The global tolerance $\delta$ is a quantity that must not be exceeded by $\mu(P)$ for any polytope $P$ generated when using GBT 7.0. This requirement can be ensured by making $\delta$ depend on two factors:

1. $d$: the dimension of the Euclidean space in which all the computations are performed.

2. $\kappa$: half the edge size of the smallest hypercube centred to the origin that contains all vertices and facets of all polytopes that will be computed.

The default formula implemented for $\delta$ in GBT 7.0 is

$$\delta = 1000 \times \kappa \times \epsilon$$ (3.1)
where \( \epsilon \) is the relative floating point precision. For most low dimensions \((d \leq 7)\) this is greater than the theoretical and conservative lower bound \((3d^{2.5} + 6d^2 + 4d^{1.5} + 9d) \times \kappa \times \epsilon\). For double precision numbers in MATLAB the \( \epsilon = 2.2204e-016 \). The global fitting precision \( \delta \) is maintained by all routines in GBT.

Fitting precision is lower for large polytopes than for small polytopes. The main reason for this is that fitting accuracy is measured by absolute error. On the other hand the numbers used in floating point computation allow for bounding relative errors and not absolute errors. This means that for large numbers the same relative error results larger absolute errors.

When computation is performed with large polytopes, it is worth normalizing the enclosing hypercube to have an edge-size 2 and centering it on the origin (i.e. setting \( \kappa = 1 \)). Even this method cannot however allow for computation simultaneously with vertices of co-ordinates less than \( \epsilon \) and with vertices with co-ordinates around 1. This is an unavoidable theoretical limit of computation by digital floating-point DSPs. Note that this situation would not improve by using fixed-point arithmetic: precision essentially depends on the number of bits used in number representations by the DSP.

If higher precision is needed to allow for both small and large scale locations of vertices, then the relative error \( \epsilon \) of floating point computation has to be reduced to suit the particular application. This means higher number of bits for floating point numbers. Apart from the possible use of symbolic computation in MATLAB, in practice this can also be achieved by compiling the GBT routines into C and compiling and running that code on a DSP arithmetic with higher floating point precision. For users who want to compile GBT routines into C there is a Programmer’s Reference provided in the last part of this manual. For more details on error control see [12].

Setting the right size of \( \delta \) and \( \kappa \) for a session of polytope computations can be done by a command such as

```matlab
global DELTA; DELTA=10e-10; global KAPPA; KAPPA=1000;
```

where the specified \( \delta \) (i.e. \texttt{delta}) must be greater than \((3d^{2.5} + 6d^2 + 4d^{1.5} + 9d) \times \epsilon \times \kappa\) where \( d \) is the dimension of the space. By convention in MATLAB, the global variables are named by all capital letters. Another possibility is to use the routine \texttt{setdelta}:

```matlab
>> setdelta
```

\texttt{setdelta} prompts to \texttt{KAPPA} to be typed in:

Half-edge-size of a hypercube centred to the origin which will include all polytopes to be used [default of KAPPA is 100]: \texttt{KAPPA=}

After \texttt{KAPPA} is typed in, a value of \texttt{DELTA} is computed and confirmed by \texttt{setdelta}:

The facet-vertex fitting tolerance \texttt{DELTA} is set to the following value for all computations in GBT routines:

\texttt{DELTA = 1.0000e-011}
3.2. SPEED OF COMPUTATION AND MEMORY REQUIREMENTS

If one forgets to use `setdelta` before a GBT, `setdelta` is automatically activated once. The global variable `ERRFACTOR` is used for error control of internal and outer bounding polytopes to reduce complexity. `CONVH, P_CONV, FCONVH, UPDATE, POLADD, SUBTRACT` do take the input options 'inter', 'outer' and 'numer' for strictly internal, outer bounding and numerical accuracy modes, respectively. The default value is 'numer' when the best numerical accuracy is considered, without guaranteeing inner or outer bounding of the 'ideal' polytope by the computed one.

If, during computations, the enclosing hypercube (with edges $2\kappa$) is violated, a warning message is given:

```
Warning: Enclosing hypercube was violated and polytopes are clipped.
Use "setdelta".
```

If the global error bound $\delta$ is violated by a $\mu(P)$ in any of the routines in GBT, then an error message is produced indicating the routine where violation occurred, for instance:

```
??? Global variable "delta" was violated. Use "setdelta".
```

Violation of the vertex-facet fitting error-bound $\delta$ cannot be allowed; if the error bound is violated, logical decisions concerning vertex-facet adjacency may be incorrect, possibly resulting in an explosion of numerical errors. To avoid major problems with numerical precision, it is good practice to check in advance the maximum size of polytopes that will be encountered during the computation. The choice of the hypercube has to be reasonable as large choice of an enclosing hypercube would reduce precision of facet-vertex fitting. For more details on error control see [12].

3.2 Speed of computation and memory requirements

Computation with higher dimensional polytopes has serious limits due to the possible high number of vertices and facets. GBT is reasonably efficient with memory and the routines can be speeded up by the MATLAB C-compiler and run in executable form. This chapter illustrates the objective difficulties one faces due to the amount of computation to be performed (and not due to the limitation of the software). Computation with approximate polytopes is therefore necessary in practice and future versions of GBT will provide more new routines to reduce computational complexity. In GBT 7.0 there are two routines which can be used to reduce polytope complexity: `POLADDI` computes an internal approximation to the sum of two polytopes, `VREDUCE` reduces the number of points in a set by combining the ones which are DELTA-near. This section presents exercises which the Reader can try in GBT 7.0 to get a feel for the complexity of polytopes with higher dimensions. For this section the definition of a contemporary average PC is one with 800MHz with Intel Pentium III, 256MByte memory and the computation are done in MATLAB 6.

1. Type
CHAPTER 3. ERROR CONTROL AND SPEED OF COMPUTATION

```matlab
>> n=(1:5); for i=4:8, V=rand(20,i); P=convh(V); [F,n(i-3)]=faces(P); end; n
to get a feel for the number of hyperplanes a random polytope can have which
is obtained as the convex hull of 20 random vertices. A typical output is

```
4
```
```
61 158 293 676 1058
```

The last entry is the number of hyperplane facets of a random, 8-dimensional
polytope with less than 20 vertices (as it is the convex hull of 20 randomly
generated points). When on cuts this last polytope with a hyperplane, the cut
has to be computed with several hundreds of hyperplanes. 1058 hyperplanes
facets is not however near the maximum possible number of facets for 20
vertices. This can be checked by

```matlab
>> maxface(20,8)
ans =
    3265
    3415
    4845
   11500
   14450
   9100
   2275
```
gives the maximum number of 1D, 2D, and finally the 7D (i.e. 7-dimensional)
faces of a polytope with 20 vertices. The maximum number of facets of an
8D polytope with 20 vertices is 2275.

2. For higher than 8D the maximum number of facets of a polytope drastically
increases. This can be checked by

```matlab
>> n=(1:6); for i=8:13, nf=maxface(20+5*i,i); n(i-7)=nf(i-1); end; n
```
```
 n =
    393525     975270    8895264   22477026   200271120   513703190
```
The last number is the maximum number of facets in 13D for 50 vertices.
Maybe 51 million faces are possible to list but certainly cannot easily be
manipulated with on a contemporary PC under the MATLAB environment.
Compilation in C and running on more powerful computers is becoming a
necessity when one computes with higher dimensional polytopes.

```matlab
V=rand(24,20); [F,nf]=faces(P); P=convh(V); [F,nf]=faces(P); nf
```
```
 nf =
    1566
```
3.2. SPEED OF COMPUTATION AND MEMORY REQUIREMENTS

that even a small difference in the dimension and the random number of points can give rise to 1566 facets.

3. The computational speed is illustrated by the following run:

\[ i=2; V = \text{rand}(100, i); \text{com} = \text{clock}; P = \text{convh}(V); \text{fin} = \text{clock}; \text{fin}(6) - \text{com}(6) \]

A typical result on a contemporary PC, as defined above, is

\[ \text{ans} = 0.6600 \]

which means a runtime of 0.66s. For larger dimensions this changes to

\[ i=4; V = \text{rand}(20, i); \text{com} = \text{clock}; P = \text{convh}(V); \text{fin} = \text{clock}; \text{fin}(6) - \text{com}(6) \]

\[ \text{ans} = 1.8100 \]

In high dimensional spaces this increases considerably:

\[ i=6; V = \text{rand}(20, i); \text{com} = \text{clock}; P = \text{convh}(V); \text{fin} = \text{clock}; 60 - \text{com}(6) + \text{fin}(6) \]

\[ \text{ans} = 32.8500 \]

In 8D the computation time is substantially greater:

\[ i=8; V = \text{rand}(20, i); \text{com} = \text{clock}; P = \text{convh}(V); \text{fin} = \text{clock}; 60 - \text{com}(6) + \text{fin}(6) \]

\[ \text{ans} = 324.4500 \]

which is more than 5 minutes.

4. The intersection of a random set of half-spaces

\[ i=8; V = \text{rand}(20, i); \text{com} = \text{clock}; P = \text{convh}(V); \text{fin} = \text{clock}; (\text{fin}(5) - \text{com}(5)) * 60 + \text{fin}(6) - \text{com}(6) \]

; also takes a long time, a typical running time is

\[ \text{ans} = 19.8300 \]

; i.e. around 20 seconds.

Similar runs as shown in this section can be easily generated to develop a feel for the computational time and size of polytope descriptions used.
The total central memory needed in the above computations in number of Bytes is a 8-30 times the product of the number of vertices and the number of facets of the largest polytope used. It can therefore easily happen that MATLAB runs out of memory.

In conclusion, it has to be measured up, before any task is to be run, whether the computer is powerful enough. Limited complexity polytopes, such as the generalized diamond as output by the ELLADDI routine, are becoming increasingly important and future versions of GBT will contain more routines for approximate, reduced complexity computations.

For more details on error control see [12].
Chapter 4

Programming Reference

The square brackets after listed input-output variables give default values defined by the routine. Although every effort has been made to make this reference manual accurate, some typos may occur, the GBT team is grateful for any comments made by the user of this manual.
Purpose

The purpose of Bounded is to test whether a polyhedron is bounded.

Usage

function B=bounded(P);

where the variables used are:
P - polytope in GBT 6.1 type 4 format
B - Boolean value 0 or 1

Brief Description of the Algorithm

The adjacency tables extracted from the GBT 6 format are used to see whether some vertices have less than ‘d’ adjacent vertices. If this occurs then the polyhedron is not bounded.

Example

The following code defines a type 4 polytope representation (in 1D) and it is tested whether it is bounded:

```matlab
P=[2 2; 1 1; -1 1;-1 -1; 1 -1;2 0;1 0;2 0;1 0];P
P =
 2  2
 1  1
-1  1
-1 -1
 1 -1
 2  0
 1  0
 2  0
 1  0
B=bounded(P);B
B =
1
```
**Purpose**

The purpose of **COMBIN** is to compute the \(k\)-class combination of \(n\) objects.

**Usage**

```matlab
function [C,nc]=combin(n,k);
```

where the variables used are:

- \(n\) - number of objects
- \(k\) - number of classes
- \(C\) - array of combinations
- \(nc\) - number of combinations

**Brief Description of the Algorithm**

**COMBIN** computes the \(k\)-class combination of \(n\) objects by the formula

\[
C(n, k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}
\]

so that \(C(n, k) = n!/(k!/(n-k)!\) . **COMBIN** also provides a list of all combination in its first output.

**Example**

The following code demonstrates the use of **COMBIN**:

```matlab
C = combin(3,2)
C =
  1 2
  1 3
  2 3
```
CONVH

Purpose

The purpose of CONVH is to compute convex hull of points.

Usage

function [H,acc]=convh(V);

where the variables used are:

V[0] - set of points
H[ [1 eps; 1 0] ] - convex hull polytope
acc[eps] - maximum fitting error of adjacent vertices and facets

Brief Description of the Algorithm

Convex hull computation for a set of points means determining all the equations of the hyperplane facets of the polytope which form the convex hull. The algorithm of ‘convh’ is based on consecutive inclusion of points into the convex hull one-by-one. The subroutine ‘p_conv’ is performing the inclusion of one further point. ‘convh’ is first looking for the vertices of a large simplex among the points given and then always includes the farthest lying point into the polytope using ‘p_conv’. If the initial simplex is lower dimensional than that of the space where the points are defined, then ‘convh’ switches to a lower dimensional space and calls itself to compute the convex hull. The lower dimensional representation obtained is finally converted back to the original dimension. See Chapter 1 on the format of polytope representations.

Example

The following demo generates a random set of points in 3D and displays the projected 2D wire-frame views of their convex hull:

V=rand(6,3);P=convh(V);view2d(P,[1,2,3]); % no.figs=1
DEF_NPS

Purpose
The purpose of DEF_NPS is to define an NPS (nonconvex polyhedral set).

Usage

function P=def_nps(List1,List2);

where the variables used are:
List1 - cell vector of +polytopes
List2 - cell vector of -polytopes
P - nonconvex polyhedral set

Brief Description of the Algorithm
The algorithm is a trivial composition of provided entries into a two column format (see Chapter 2 for the definition of an NPS representation).

Example

>> p1=convh(rand(10,2));
>> p2=convh(rand(10,2));
>> p3=convh(rand(10,2));
>> def_nps({p1,p2},{p3})

Figure 4.1: Demo figure to routine CONVH.
Purpose

The purpose of DEFBOX is to generate an axis aligned box.

Usage

\[
\text{function } P = \text{defbox}(\text{vertex1}, \text{vertex2})\]

where the variables used are:

- vertex1\text{[-1]} - lowest position corner of the box
- vertex2\text{[1]} - highest position corner of the box
- \text{P}[[0 \text{ eps}; 1 1; -1 -1; -1 -1; 1 -1]] - box polytope

Brief Description of the Algorithm

The algorithm is based on building up lists of vertices in a way as binary numbers are build up from 0s and 1s with the difference being that the 0s and 1s are respectively replaced by lower-corner and upper corner components. The number of vertices thus becomes \(2^d\) where \(d\) is the dimension of the space. The \(2d\) facet inequalities are formed by unit vectors and components of the lower and upper corners.

Example

The following code defines a box with diagonal corners [0.4,0.5,1] and [1.2,1.4,2] and displays its projected wire-frame views:

\[
P = \text{defbox}([0.4,0.5,1],[1.2,1.4,2]); \text{view2d}(P, [1 2 3]); \% \text{no.figrs=1}
\]

\[
\text{ans =}
\begin{array}{c}
[9x3 \text{ double}] \\
[9x3 \text{ double}] \\
[11x3 \text{ double}]
\end{array}
\]

\[
\begin{array}{c}
+ \\
+ \\
- \\
\end{array}
\]
Figure 4.2: Demo figure to routine DEFBOX.

---

**DEFCYL**

**Purpose**

The purpose of **DEFCYL** is to generate a cylinder that is bounded by two half-spaces.

**Usage**

```matlab
function P=defcyl(r,p,a,h1,h2);
```

where the variables used are:

- **r** - radius of the cylinder
- **p** - a point on the cylinder axis
- **a** - direction vector of cylinder axis
- **h1** - half-space inequality to contain the finite cylinder
- **h2** - half-space inequality to contain the finite cylinder

**Brief Description of the Algorithm**

The algorithm is based on simple construction of the cylinder matrix representation.

**Example**

The following code defines a cylinder with radius 2.1, an axis point [0 1 0], axis direction [1 1 1] and bounding half-spaces [1 1 1], [-1 -1 -1], and displays it
C=defcyl(1,[0 1 1],[1 1 0],[1 1 -.1 1],[-1 -1 -1 1]);view3dcyl(C);% no.figs=1

Figure 4.3: Demo figure to routine DEFCYL.
Purpose

The purpose of DEFELL is to define an ellipsoid.

Usage

\[
\text{function } E = \text{defell}(P,c)
\]

where the variables used are:

- \( P[1] \) - defining matrix of ellipsoid
- \( c \) - centre of ellipsoid
- \( E[[1 0;1 0]] \) - ellipsoid

Brief Description of the Algorithm

This is a direct definition of the ellipsoid representation of \( E = \{ x | (x-c)^T P^{-1} (x-c) \leq 1 \} \) in a matrix format:

\[
\begin{bmatrix}
1 & 0 \\
P & c
\end{bmatrix}
\]

where \( P \) is the ellipsoid covariance and \( c \) is the ellipsoid centre.

Example

The following code defines a 3D ellipsoid with random covariance and random centre.

\[
A = \text{rand}(3,3); P = A*A; c = \text{rand}(3,1);
E = \text{defell}(P,c);
\text{view2del}(E,[1 2 3]); \text{ no.figs=1 }
\]
DEFPIPE

Purpose
The purpose of DEFPIPE is to define a parallelepiped.

Usage

\[
\text{function } P = \text{defpipe}(b1, A, b2);
\]

where the variables used are:

- \( b1[-1] \) - left hand side of the inequalities
- \( A[1] \) - matrix of the inequalities
- \( b2[1] \) - right hand side of the inequalities
- \( P[[0 \text{ eps};1 1;-1 -1; 1 -1]] \) - parallelepiped polytope

Brief Description of the Algorithm
The polytope defined by the inequalities \( b_1 \leq Ax \) and \( Ax \leq b_2 \) is computed by forming a single set of inequalities as

\[
\begin{bmatrix}
A \\
-A
\end{bmatrix}x \leq \begin{bmatrix}
b_2 \\
-b_1
\end{bmatrix}
\]

and then the routine ‘fconvh’ is applied to get the polytope of a parallelepiped.

Example
The following code defines a random parallelepiped satisfying the inequalities $b_1 \leq Ax$ and $Ax \leq b_2$:

$$
A = \text{rand}(5, 3) - 0.5; b_1 = -\text{ones}(5, 1); b_2 = \text{ones}(5, 1);
$$

$$
P = \text{defpipe}(b_1, A, b_2); \text{view2d}(P, [1 2 3]); \% \text{no.figs}=1
$$

Figure 4.5: Demo figure to routine DEFPIPE.
DEFSIMP

Purpose

The purpose of \textbf{DEFSIMP} is to define a regular centred simplex with edge size 1.

Usage

\begin{verbatim}
function S=defsimp(d);
\end{verbatim}

where the variables used are:

\begin{itemize}
  \item \texttt{d[1] - dimension}
  \item \texttt{S} - \texttt{[[0 eps;1 0.5;-1 0.5;-0.5 -1; 0.5 -1]] = simplex}
\end{itemize}

Brief Description of the Algorithm

Regular simplexes are recursively built up from lower dimension to higher dimension by computing a new vertex equidistant from all former vertices. The facet inequalities are directly computed without the use of ‘convh’ or ‘fconvh’. The initial 1D simplex is on the real axis with “vertices” at -0.5 and 0.5.

Example

The following code defines a regular simplex and displays it:

\begin{verbatim}
P=defsimp(3);view2d(P,[1 2 3]); % no.figs=1
\end{verbatim}

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{fig4.6}
\caption{Demo figure to routine DEFSIMP.}
\end{figure}
This demo generates a random set of points for dimensions 2-4 and computes a convex hull polytope, then recomputes this polytope from its facets. If this m-file does not run correctly please contact by email: gbt@sysbrain.com

The demo repeats random point generations, convex hull computations, facet extraction and polytope computations from facet inequalities for dimensions 2, 3 and 4. The polytopes 'P' and 'P1' should be nearly the same in each figure.

```matlab
for i=1:3,
    figure(i);
    V=rand(10,i+1); % random points
    P=convh(V); % convex hull
    F=faces(P); % facet inequalities
    P1=fconvh(F); % polytope from facets
    figure(i);
    a=view2d(P,(1:max(2,i+1))); % display P
    view2d(P1,(1:max(2,i+1)),'c',a); % display P1 % no.figs=3
end;
echo
```

Figure 4.7: Figure to illustrate DEMO1.
This demo tests the worst-case fitness ratio for half-space updatings with polytopes in dimensions 6-7 over 10 repetitions. If this m-file does not run correctly please contact us at Email address: gbt@sysbrain.com

Define number of samples for each dimension:
nosamp=[100, 50, 20, 15, 11, 12, 13, 14, 15];

Initialize of the worst fitting accuracy over all computations:

mumax=eps;

Repeat the random generation of half-space 10 times:

for j=1:10;
  for i=6:7,
    [j,i]
    F=[rand(nosamp(i-1),i)-0.5 ones(nosamp(i-1),1)];
    P=fconvh(F);
    if fvfitacc(P)>mumax, mumax=fvfitacc(P);end; % update
    end;
  end;
end;

Display the worst-case fitting accuracy as multiple of EPS:

disp('Worst-case fitness ratio (=\mu/\epsilon):');
mumax/eps

echo
This demo tests the worst-case fitness ratio for convex hull computations in dimensions 2-7 over 10 repetitions. If this m-file does not run correctly please contact us. Email: gbt@sysbrain.com

Number of samples for each dimension:

\[ \text{nosamp} = [100, 50, 20, 15, 11, 12, 13, 14, 15] \];

Initialize the worst fitting accuracy over all computations:

\[ \text{mumax} = \text{eps}; \]

Repeat the random generation of half-spaces 10 times:

\begin{verbatim}
for j=1:10;
for i=2:4,
 [i,j] V=rand(nosamp(i-1),i);
P=convh(V);
if fvfimacc(P)>mumax, mumax=fvfimacc(P);end;
end;
end;
\end{verbatim}

Display the worst-case fitting accuracy in multiples of EPS:

\begin{verbatim}
 disp('');
disp('Worst-case fitness ratio (=mu/eps):');
mumax/eps
end;
\end{verbatim}
This demo m-file adds three polytopes: two boxes and a random polytope, based on convex hull computations. If this m-file does not run correctly please contact us
Email: gbt@sysbrain.com

Define of two boxes:

\[
P_1 = \text{defbox}([-1,-2],[2,2]);
\]
\[
P_2 = \text{defbox}(0.1 \cdot [-1,-1], 0.1 \cdot [1,1]);
\]

Computing the Minkowski sum of the two boxes:

\[
P = \text{poladd}(P_1, P_2);
\]

Extract the vertices and facets of the sum:

\[
[v, nv] = \text{vertices}(P); [f, nf] = \text{faces}(P);
\]
\[
\text{disp('Number of vertices and faces of the box sum:');}
\]
\[
[nv, nf]
\]

Generate a set of 50 random points:

\[
P_3 = \text{convh}([\text{rand}(50,2)]); \% \text{computing convex hull}
P_4 = \text{poladd}(P_2, P_3); \% \text{adding P2 and P3}
\]
\[
[v, nv] = \text{vertices}(P_4); [f, nf] = \text{faces}(P_4); \% \text{extracting vertices and facets}
\]

Display the number of vertices and facets:

\[
\text{disp('Number of vertices and faces of the polytope sum:');}
\]
\[
[nv, nf]
\]
echo off
This demo illustrates the sensitivity of facet updating. In the first part a triangle is cut by a half-space nearly coincident with a side of the triangle. The second part recomputes a random polytope from its facet inequalities. If this m-file does not run correctly please contact us Email: gbt@sysbrain.com

Convex hull of three points:

\[
P = \text{convh}([0 \ 0; 1 \ 0; 0 \ 1])
\]

Define a half-space with borderline near to an edge:

\[
h = [1 \ 1 \ 0.994];
\]

Update 'P' with 'h':

\[
P1 = \text{update}(h, P);
\]

View the updated and original triangle in the same plot:

\[
ax = \text{view2d}(P, [1 \ 2]) ; \text{view2d}(P1, [1 \ 2], 'g', ax);
\]

This part generates a random 3D polytopes and recomputes them from their facet inequalities. First extract facets from random polytope:

\[
P = \text{convh(rand(10,3))) ; F = \text{faces}(P);
\]

Update with first facet inequality:

\[
h = F(:,1) ; P1 = \text{update}(h, P);
\]

View original and updated polytopes in same plot:

\[
ax = \text{view2d}(P, [1 \ 2 \ 3]); \text{view2d}(P1, [1 \ 2 \ 3], 'g', ax);
\]
echo off; \% no.figs=2

Figure 4.10: Figure to illustrate DEMO5.
This demo generates 3 random 3D ellipsoids and approximates them by inner and outer bounding polytopes. The volume of the intersection is estimated and lower and upper bounds are given. If this m-file does not run correctly please contact us Email: gbt@sysbrain.com

Generate 3 random 3D ellipsoids:

```matlab
d=3;  % dimension
n=3;  % no of ellipsoids
E=zeros(d+1,d+1,n);
for i=1:n,
    C=rand(10,d)-0.5;C=C'*C/2;
    E(:,:,i)=defell(C,0.1*rand(d,1));
end;
```

Generate hyperplanes of inner and outer approximations and collect them in arrays ‘Hoall’ and ‘Hiall’

```matlab
nord=4; Hiall=[];Hoall=[];
for i=1:n,
    [Ho,Hi]=ellapprx(E(:,:,i),nord);
    Hoall=[Hoall; faces(Ho)];
    Hiall=[Hiall; faces(Hi)];
end;
```

Intersect all half-spaces in ‘Hoall’:

```matlab
h1=size(Hoall,1);
B=defbox(-10*ones(1,d),10*ones(1,d));  % outer bound
Pouter=B;
for i=1:h1,
Pouter=update(Hoall(i,:),Pouter);
end;
```

Intersect all half-spaces in ‘Hiall’:

```matlab
h2=size(Hiall,1);
B=defbox(-10*ones(1,d),10*ones(1,d));
Pinner=B;
for i=1:h2,
```
Pinner=update(Hiall(i,:),Pinner);
end;

View the outer and inner approximations of intersection:
figure;ax=view2d(Pouter,[1 2 ]);view2d(Pinner,[1 2 ],'g',ax); % no.figs=1

Volume computations for both inner and outer polytope of intersection:
disp(' upper bound of volume:')
upvol=polvol(Pouter)
disp('lower bound of volume:')
lovol=polvol(Pinner)

Estimate the intersection by the average of the internal and outer polytope volumes:
disp('estimate of intersection volume:')
(upvol+lovol)/2
echo off

Figure 4.11: Figure to illustrate DEMO6.
This demo computes a maximal polytope ‘inside’ a set of points (not optimal procedure). First generates a random set of points around the origin and leaves out points near the origin. Then it constructs a polytope inside the set of points. If this m-file does not run correctly please contact us by Email: gbt@sysbrain.com

Generate random set of points with empty space in the middle:

```matlab
N=30;d=2; % number of points and dimension
nV=rand(N,d)-0.5;V=[];
```

Eliminate points near to origin:

```matlab
for i=1:N,if norm(nV(i,:))>0.25, V=[V;nV(i,:)]end;end;
```

Compute the "associated set of half-spaces" to the set of points V:

```matlab
H=dualpon(V);
```

Intersect the "associated dual set of half-spaces" H:

```matlab
P=fconvh(H);
```

Display original points and largest polytope P inside the set of points:

```matlab
figure;ax=newplot;plot(V(:,1),V(:,2),'x');
view2d(P,[1 2 ],'g',ax);

echo off; % no.figs=1
This demo m-file estimates the volume of the intersection of 30 ellipsoids in 4D. If this m-file does not run correctly please contact us by Email: gbt@sysbrain.com

Generate 3 random 4D ellipsoids:

d=4; % dimension
n=3; % no of ellipsoids
E=zeros(d+1,d+1,n);
for i=1:n,
    C=rand(20,d)-0.5;C=C’*C/2;
    E(:,:,i)=defell(C,0.1*rand(d,1));
end;

View of the ellipsoids by projections onto 1-2-3-4 axes:

E1=[]; bool=1>0;
ax=view2del(E(:,:,1),[1 2 3 4]);
for i=2:n,
    E1=[E1;E(:,:,i)];
    view2del(E(:,:,i),[1 2 3 4],'blue',ax,'-',100,bool);
end;
Use the Monte-Carlo volume routine MCVOL:

\[ \text{vol} = \text{mcvol(E1,10)} \]

\[ \text{echo off; } \% \text{ no.fig} = 1 \]

Figure 4.13: Figure to illustrate DEMOS.
This demo tests the accuracy of the PLANFIT routine. If this m-file does not run correctly please contact us by email: gbt@sysbrain.com

Test for three almost aligned points in 3D space:

\[
N=10; ac=(1:N)'; 
for i=1:N, 
V=[0 0 0; 0 1+eps*i 0; 0 2 0]; 
[a,b,ac(i)]=planfit(V); 
end; 
\]

Display the worst-case error of fitting:

\[
\text{max(abs(ac/eps))} 
\]

Generate points for dimensions d=4:10 so that 3rd and 4th points are linear combinations of the first two points and compute the planes fitted:

\[
\text{for d=4:10,} \text{disp('dimension computed: ');} 
\text{for i=1:N,} 
V=rand(d,d); 
V(3,:)=V(1,:)+V(2,:); 
V(4,:)=V(2,:)+0.2*V(1,:); 
[a,b,ac(i)]=planfit(V); 
end; 
\]

Display the worst case fitting error:

\[
[d, \text{max(abs(ac/eps))}] 
\]

\[
\text{echo off; \% no.fig=9} 
\]
This demo simulates 5 random sets of 5 spheres and computes the area of the joining flat surfaces. If this m-file does not run correctly please contact us by email: gbt@sysbrain.com Five ellipsoids are defined for each of 10 simulations:

```matlab
N=5;est=0; % initialize
for j=1:N,
    E1=defell(eye(3));
    E2=defell(eye(3),rand(1,3)-0.5);
    E3=defell(eye(3),rand(1,3)-0.5);
    E4=defell(eye(3),rand(1,3)-0.5);
    E5=defell(eye(3),rand(1,3)-0.5);
    E=[E1;E2;E3;E4;E5];

    [e,o,i]=sphsep(E);
    disp(' Exp. No., the estimate (e), upper bound(o), lower bound (i):');
    [e,o,i]
    est=est+e;
end;
```

The average surface area over 10 simulations is estimated and displayed:

```matlab
disp('Average of joining surface area:');
est/N
echo
```
This demo generates a set of random inequalities for dimensions 2-4 and computes the intersection polytope, then recomputes this polytope from its vertices. If this m-file does not run correctly please contact by email: gbt@sysbrain.com

Repeating random point generations, convex hull computations, facet extraction and polytope computations from facet inequalities for dimensions 2, 3 and 4. The polytopes ‘P’ and ‘P1’ should be nearly the same in each figure.

```matlab
close all;
for i=1:3,
    figure(i);
    F=[rand(10,i+1)-0.5 ones(10,1)]; % random points
    P=fconvh(F); % convex hull
    V=vertices(P); % facet inequalities
    P1=convh(V); % polytope from facets
    figure(i);
    a=view2d(P,(1:max(2,i+1))); % display P
    view2d(P1,(1:max(2,i+1)),'c',a); % display P1
end;

echo % no.figs=3
```

![Figure 4.14: Figure to illustrate Demo11.](image)
This demo generates two random 2-dimensional NPSs and computes their intersection and union. Then the area of the intersection and union is computed.

% Definition of first random nonconvex set P1:
P1=cell(0,2); N=8; m=2; for i=1:N,
offset=rand(1,n)*1;
V=rand(N,n)+ones(N,1)*offset;
P1=convh(V);
z=rand(1); if z>0.0, P1=[P1;{P1,'+'}];
else P1=[P1;{P1,'-'}]end;
end;
disp('The randomly generated area of P1 is displayed:');
figure(1);
view2d_(P1,[1,2],'y');axis equal

title(['Area of P1 is ',num2str(polvol_(P1)),' km^2']);

% Definition of second random nonconvex set P2:

P2=cell(0,2); for i=1:N,
offset=rand(1,n)*1;
V=rand(N,n)+ones(N,1)*offset;
P1=convh(V);
z=rand(1); if z>0.0, P2=[P2;{P1,'+'}];
else P2=[P2;{P1,'-'}]end;
end;
disp('The randomly generated area of P2 is displayed:'); figure(2);
view2d_(P2,[1,2],'b');axis equal;
title(['Area of P2 is ',num2str(polvol_(P2)),' km^2']);

% Computing the intersection :
Pi=intersec_(P1,P2);figure(3); ax1=view2d_(P1,[1,2],'y');
view2d_(P2,[1,2],'b',ax1); view2d_(Pi,[1,2],'g');axis equal;
title(['Area of P1 and P2 intersection is ',...
num2str(polvol_(Pi,20)),' km^2']);

% Computing the union:
Pu=union_(P1,P2);figure(4); ax2=view2d_(P1,[1,2],'y');
view2d_(P2,[1,2],'b',ax2);
view2d_(Pu,[1,2],[0.7 0.7 0.7],ax2);axis equal;
title(['Area of P1 and P2 union is ',...
num2str(polvol_(Pu,20)),' km^2']);
Area of P1 is 1.4852 km$^2$

Figure 4.17: Figure to illustrate Demo12.

Area of P2 is 1.699 km$^2$

Figure 4.18: Figure to illustrate Demo12.
Area of P1 and P2 intersection is 1.4543 km$^2$

Figure 4.19: Figure to illustrate Demo12.

Area of P1 and P2 union is 2.0633 km$^2$

Figure 4.20: Figure to illustrate Demo12.

---

### DIAMET

**Purpose**

The purpose of **DIAMET** is to compute the diameter of a set of points.

**Usage**
function [diam,ds,de,ij,j1]=diamet(V);

where the variables used are:

V[1] - set of points
diam[0] - length of diameter
ds[1] - starting point of a diameter
de[1] - ending point of a diameter

Brief Description of the Algorithm

The diameter for a set of points is computed by comparing distances of all pairs of points in the set (note that this routine handles a set of points and not a polytope). The maximum diameter is done by double application of the numerically efficient routine ‘max’ provided by MATLAB.

Example

The following code finds the diameter of a random set of points and plots them with the diameter:

```matlab
V=rand(10,2);[diam,ds,de]=diamet(V);
plot(V(:,1),V(:,2),'rx');X=[ds';de'];
line(X(:,1),X(:,2)); % no.figs=1
```

![Figure 4.21: Demo figure to routine DIAMET.](image-url)
**Purpose**

The purpose of **DIAMETER** is to compute the diameter of a polytope.

**Usage**

```matlab
function [diam,ds,de]=diameter(P);
```

where the variables used are:

- `P[[1 eps;1 1]]` - polytope
- `diam[0]` - length of diameter
- `ds[1]` - starting point of a diameter
- `de[1]` - ending point of a diameter

**Brief Description of the Algorithm**

The diameter of a polytope is computed by comparing distances of all pairs of vertices. The choice of the maximum vertex distance is done by double application of the efficient routine `max` provided by MATLAB. A similar routine, but acting on a set of points, is `diamet`.

**Example**

The following code finds the diameter of a random polygon and plots its 2D wireframe views with the diameter:

```matlab
V=rand(10,2);P=convh(V);[diam,ds,de]=diameter(P);
view2d(P,[1 2]); X=[ds';de'];line(X(:,1),X(:,2)); % no.figs=1
```
DIF

Purpose

The purpose of DIF is to compute the set-difference of vector sets.

Usage

function U=dif(U1,U2);

where the variables used are:

U1[1] - first set
U2[1] - second set
U[ ] - set difference

Brief Description of the Algorithm

All points in ‘U2’ are tested whether they are DELTA-near to a point in ‘U1’. If the test is positive then the point is not listed in the result set ‘U’. All other points of ‘U1’, which are not DELTA-near to points in ‘U2’, are listed in ‘U’.

Example

The following code takes the set-difference of two sets of points:

Set1=[1 2; 2 1; 1 1]; Set2=[1 3; 2 1];
Set=dif(Set1, Set2)
Purpose
The purpose of DIM is to compute the dimension of a polytope.

Usage

function [d,dactual,mu]=dim(P);

where the variables used are:

P[ [1 eps;1 1] ] - polytope
d[1] - dimension of polytope representation
dactual[0] - actual dimension of polytope
mu[eps] - largest fitting error of facets and vertices

Brief Description of the Algorithm
The dimension ‘d’ of a polytope matrix-representation is the number of columns in the representation -1. This routine does not recompute the actual dimension, it uses the ‘P(1,1)’ entry for the degree of degeneracy, ‘d-P(1,1)’ is defined as the actual dimension of the polytope.

Example
The following code defines a random 2D triangle in 3D space and displays it and computes its real dimension:

V=rand(3,3);P=convh(V);view2d(P,[1 2 3]); % no.figs=1
[d,d_actual]=dim(P)
d =
3

d_actual =
2
CHAPTER 4. PROGRAMMING REFERENCE

Projection to 1−2 axes

Projection to 1−3 axes

Projection to 2−3 axes

Figure 4.23: Demo figure to routine DIM.

DIREXT

Purpose

The purpose of DIREXT is to compute extremal points in a given direction.

Usage

```matlab
function [vmax,vmin,width]=dirext(G,di);
```

where the variables used are:

- `G[[1 eps;1 1]]` - polytope
- `di[1]` - direction vector
- `vmax[1]` - extremum in + direction
- `vmin[1]` - extremum in - direction
- `width[0]` - the width in the direction

Brief Description of the Algorithm

The extreme point in a given direction is obtained by projecting all vertices onto a line aligned with the given direction. The maximum and minimum points and their indices are found by the efficient routines ‘max’ and ‘min’ provided by MATLAB. The corresponding vertices are found by the indices from the vertex list.

Example
The following code generates a random polygon, then finds its extreme point in the [1, 1] direction and places an ‘o’ to that vertex.

```matlab
P=convh(rand(7,2));v=dirext(P,[1 1]);
view2d(P,[1 2]); hold on; plot(v(1),v(2),'bo');
```

---

**DIRGEN**

**Purpose**

The purpose of **DIRGEN** is to generate an approximately uniform set of direction-vectors.

**Usage**

```matlab
function [D,ndir]=dirgen(d,n)
```

where the variables used are:

- `d[1]` - dimension
- `n[2]` - number of directions for each dimension
- `D` - set of row direction vectors

**Brief Description of the Algorithm**

The routine builds up a set of directions recursively by adding uniformly distributed directions for each new dimension included. First in 2D 2n uniform circular direction are produced in [0, 2π] with angles \(2k\pi/2n\), \(k = 1, \ldots, 2n\). Then for each new dimension the number of uniformly distributed new directions is in \([-\pi/2, \pi/2]\] with angles \(-\pi/2 + k\pi/n\), \(k = 1, \ldots, n - 1\) and two more points \([0, \ldots, 0, 1]\) and \([0, \ldots, 0, -1]\) are also added as ‘poles’.

**Example**

The following demo generates approximately uniformly distributed directions in 3D. 6 divisions are produced in the first two dimensions and 3 in the third dimension:

```matlab
D=dirgen(3,3)
D =
    0    0    1
    0    0   -1
0.43301 0.75 -0.5
-0.43301 0.75 -0.5
-0.866031 0.605e-016 -0.5
-0.43301 -0.75 -0.5
0.43301 -0.75 -0.5
```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0.86603-2.1211e-016</strong></td>
<td><strong>-0.5</strong></td>
<td></td>
</tr>
<tr>
<td><strong>0.43301</strong></td>
<td><strong>0.75</strong></td>
<td><strong>0.5</strong></td>
</tr>
<tr>
<td><strong>-0.43301</strong></td>
<td><strong>0.75</strong></td>
<td><strong>0.5</strong></td>
</tr>
<tr>
<td><strong>-0.866031.0605e-016</strong></td>
<td><strong>0.5</strong></td>
<td></td>
</tr>
<tr>
<td><strong>-0.43301</strong></td>
<td><strong>-0.75</strong></td>
<td><strong>0.5</strong></td>
</tr>
<tr>
<td><strong>0.43301</strong></td>
<td><strong>-0.75</strong></td>
<td><strong>0.5</strong></td>
</tr>
<tr>
<td><strong>0.86603-2.1211e-016</strong></td>
<td><strong>0.5</strong></td>
<td></td>
</tr>
</tbody>
</table>
Purpose
The purpose of DUALPON is to compute a set of half-spaces "normal" to a set of vectors.

Usage

function H=dualpon(V);

where the variables used are:
V[1] - set of points
H[[1 1]] - set of half-space inequalities

Brief Description of the Algorithm
The algorithm starts from a given set of points ‘V’. For each vector ‘v’ in ‘V’ the equation of the hyperplane orthogonal to ‘v’, and fitting to its endpoint, is computed. The list of these hyperplane equations gives the output list ‘H’.

Example
The next code defines a random set of points and calculates the intersection of the corresponding half-spaces to form an ‘internal’ polygon.

V=rand(50,2)-0.5; V=V(diag(V*V')>0.09,:); H=dualpon(V); P=fconvh(H);
view2d(P,[1 2]); hold on; plot(V(:,1),V(:,2),'bo');
**ELLADDE**

**Purpose**

The purpose of **ELLADDE** is to compute outer bounding ellipsoid to the sum of two ellipsoids.

**Usage**

```matlab
function [e,p,v]=ELLADDE(e1,e2,rin);
```

where the variables used are:

- `e1` - first ellipsoid
- `e2` - second ellipsoid
- `rin` - criterion (options: `rin>0`, `'vol'`, `'tr1'`, `'tr2'`)
- `e` - outer bounding ellipsoid of the sum
- `p` - optimization parameter received
- `v` - criterion value optimized

**Brief Description of the Algorithm**

The family of inclusion-minimal external ellipsoids for the sum of two ellipsoids with covariance matrices $Q_1, Q_2$ and centres $a_1, a_2$ is of the form $\text{defell}(Q(p), a_1 + a_2)$ where $p \in [\sqrt{\lambda_{\text{min}}}, \sqrt{\lambda_{\text{max}}}]$, and

$$Q(p) = (1 + 1/p)Q_1 + (1 + p)Q_2$$

and $\lambda_{\text{min}}, \lambda_{\text{max}}$ are are the minimal and maximal eigenvalues of $Q_1Q_2^{-1}$. The problem is therefore to optimize $p$ with respect to some criterion. Depending on the value of `rin`, for `e=defell(Q,a)`, the

**Example**

The following code generates two random ellipses and computes their sum. The ellipses and their sum (with dashed line) are displayed in the figure.

```matlab
A=rand(2,3);e1=defell(A*A',rand(2,1));
A1=rand(2,2);e2=defell(A1'*A1);
ax=view2del(e1);
title('The two ellipses to be added and their sum (dashed line)');
ax=view2del(e2,[1 2], 'red',ax);
e=elladde(e1,e2);
ax=view2del(e,[1 2], 'r', ax, '-.'); % no.figs=1
```
ELLADDI

Purpose

The purpose of ELLADDI is to compute inner bounding ellipsoid to the sum of two ellipsoids.

Usage

function [e,S,v]=ELLADDI(e1,e2,S0);

where the variables used are:

- \( e1[[1 0; 1 0]] \) - first ellipsoid
- \( e2[[1 0; 1 0]] \) - second ellipsoid
- \( S0['vol'] \) - criterion of optimization (options: 'vol', given matrix)
- \( e[1 0; 4 0] \) - outer bounding ellipsoid of the sum
- \( S \) - parameter matrix obtained
- \( v \) - criterion value received

Brief Description of the Algorithm

The family of inclusion-maximal internal ellipsoids for the sum of two ellipsoids with covariance matrices \( Q_1, Q_2 \) and \( a_1, a_2 \), respectively, is of the form defell\((Q(p), a_1 + a_2)\), and

\[
(SQ(S)S)^{1/2} = (SQ_1S)^{1/2} + (SQ_2S)^{1/2}
\]
with $S$ any positive definite symmetrical matrix. The problem is therefore to optimize $S$ with respect to some criterion. Depending on the values of '$S0$', for `$S0`='vol' the maximal volume internal ellipsoid or for a given `$S0`=$S$ the corresponding internal ellipsoid is computed. $S0$ = 'vol', for `e=defell(Q,a)`, the ellipsoid with maximal `det(Q)` is calculated; if `$S0` is an invertible matrix, then the inner ellipsoid corresponding to `$S$=$S0` is computed.

**Example**

The following code generates two random ellipses and computes an internal approximation to their sum, based on the volume criterion. The ellipses and their “internal” sum with volume criterion (denoted by dashed line) is displayed in the figure.

```matlab
A=rand(2,3);e1=defell(A*A',rand(2,1));
A1=rand(2,2);e2=defell(A1'*A1);
ax=view2del(e1);
title('The two ellipses to be added and their sum (dashed line)');
ax=view2del(e2,[1 2],'red',ax,);
e=elladdi(e1,e2);
ax=view2del(e,[1 2],'r',ax,'--'); % no.figs=1
```

![Figure 4.25: Demo figure to routine ELLADDI.](image)
Purpose

The purpose of **ELLAPPRX** is to compute a polytope approximation to a given ellipsoid.

Usage

```
function [Pout, Pin]=ellapprx(e1, nord);
```

where the variables used are:

- `e1[[1 0;1 0]]` - ellipsoid
- `nord[2]` - number of approximations in each dimension
- `Pout[ [0 eps;1 1;-1 1;-1 -1;1 -1] ]` - tight outer bounding polytope
- `Pin[ [0 eps;1 1;-1 1;-1 -1;1 -1] ]` - tight inner bounding polytope

Brief Description of the Algorithm

The algorithm of this routine is based on first producing a set points ‘T’ on the surface of the ellipsoid by ‘dirgen’. To each of these surface points a tangential hyperplane is fitted. The polytope enclosed by the set of these hyperplanes is the external polytope computed by ‘ellapprx’. The internal polytope is computed as the convex hull of the points in ‘T’.

Example

The following code generates a random ellipsoid, computes internal and external approximations and displays the results:

```
A=rand(3,4);A=eye(3)+A*A';E=defell(A);
[Pout, Pin]= ellapprx(E,5);
ax=view2d(Pout,[1 2 3]);view2d(Pin,[1 2 3],'b',ax);
```
Purpose

The purpose of ELLCENT is to extract the centre of an ellipsoid.

Usage

    function c=ellcent(E);

where the variables used are:

E[[1 0;1 0]] - ellipsoid

    c[0] - centre of ellipsoid

Brief Description of the Algorithm

This routine simply extracts the center of the ellipsoid from its GBT representation-matrix.

Example

The following code defines a random ellipse and displays its centre with an ‘o’:

```matlab
A=rand(5,3);E=defell(A'*A,rand(3,1));ax=view2del(E);
c=ellcent(E);axes(ax(1)),hold on,plot(c(1),c(2),'rx');
axes(ax(2)),hold on, plot(c(1),c(3),'rx');
axes(ax(3)),hold on, plot(c(2),c(3),'rx');
```
ELLDIFE

Purpose

The purpose of ELLDIFE is to compute an outer bounding ellipsoid to the difference of two ellipsoids.

Usage

function [e,S,v]=ELLDIFE(e1,e2,S0);

where the variables used are:
e1[[1 0;1 0]] - first ellipsoid
e2[[1 0;1 0]] - second ellipsoid
S0['vol'] - criterion of optimization (options:'vol', given matrix)
e[1 0;4 0] - outer bounding ellipsoid of the Minkowski difference
S - parameter matrix obtained
v - criterion value received

Brief Description of the Algorithm

The family of inclusion-minimal external ellipsoids for the difference of two ellipsoids with covariance matrices $Q_1$, $Q_2$ and $a_1$, $a_2$, respectively, is of the form
defell($Q(p),a_1-a_2$), and

$$(SQ(S)S)^{1/2} = (SQ_1S)^{1/2} - (SQ_2S)^{1/2}$$

with $S$ any positive definite symmetrical matrix. The problem is therefore to optimize $S$ with respect to some criterion. Depending on the values of ‘S0’, for ‘S0’=‘vol’ the minimal volume external ellipsoid or for a given ‘S0’=S the corresponding internal ellipsoid is computed. S0 = ‘vol’, for ‘e=defell(Q,a)’, the ellipsoid with minimal ‘det(Q)’ is calculated; if ‘S0’ is an invertible matrix, then the corresponding tight external ellipsoid is computed.

Example

The following code generates two random ellipses and computes an external approximation to their difference, based on the volume criterion. The ellipses and their “external” difference (denoted by dashed line) is displayed in the figure.

```matlab
A=rand(2,3);e1=defell(4*eye(2)+A*A',rand(2,1));
A1=rand(2,2);e2=defell(A1'*A1);
ax=view2del(e1);tx1='The two ellipses to be added and ';
title([tx1 'their external difference (dashed line)']);
ax=view2del(e2,[1 2],'red',ax);
e=elldife(e1,e2);
```
ax = view2del(e,[1 2],'r',ax,'--') \% no.fig=1
ax =
1.0071

Figure 4.26: Demo figure to routine ELLDIFE.
ELLDIFFI

Purpose

The purpose of ELLDIFFI is to compute an inner bounding ellipsoid to the difference of two ellipsoids.

Usage

function [e,p,v]=ELLDIFFI(e1,e2,rin);

where the variables used are:

- e1[[1 0;1 0]] - first ellipsoid
- e2[[1 0;1 0]] - second ellipsoid
- rin['vol'] - criterion (options: rin>0,'vol','tr1','tr2')
- e[[1 0;4 0]] - outer bounding ellipsoid of the sum
- p - optimization parameters received
- v - criterion value optimized

Brief Description of the Algorithm

The family of inclusion-maximal internal ellipsoids for the difference of two ellipsoids with covariance matrices $Q_1$, $Q_2$ and $a_1$, $a_2$, respectively, is of the form
defell($Q(p), a_1 - a_2$) where $p \in [\sqrt{\lambda_{\text{min}}}, \sqrt{\lambda_{\text{max}}}] \cap (1, \lambda_{\text{min}})$, and

$$Q(p) = (1 - 1/p)Q_1 + (1 - p)Q_2$$

and $\lambda_{\text{min}}, \lambda_{\text{max}}$ are the minimal and maximal eigenvalues of $Q_1Q_2^{-1}$. The problem is therefore to optimize $p$ with respect to some criterion. Depending on the value of 'rin', for 'e=defell(Q,a)', the

Example

The following code generates two random ellipses and computes their difference. The ellipses and their difference (with dashed line) are displayed in the figure.

A=rand(2,3);e1=defell(25*A*A',rand(2,1));
A1=rand(2,2);e2=defell(A1'*A1);
ax=view2del(e1);
txl='The two ellipses to be added and their ';
title([txl 'internal difference (dashed line)']);
ax=view2del(e2,[1 2],'red','ax');
e=elldifi(e1,e2,'tr1');
ax=view2del(e,[1 2],'r','ax','--'); % no.figs=1
Purpose

The purpose of ELLDRXT is to compute extremal points of an ellipsoid in given direction.

Usage

\[
\text{function } [v_{\text{max}}, v_{\text{min}}, \text{width}] = \text{elldrxt}(G, \text{di});
\]

where the variables used are:

- \( G[[1 \ 0; 1 \ 0]] \) - ellipsoid
- \( \text{di}[1] \) - direction vector
- \( v_{\text{max}}[1] \) - extremum in + direction
- \( v_{\text{min}}[-1] \) - extremum in - direction
- \( \text{width}[2] \) - the width in the direction

Brief Description of the Algorithm

This routine computes the extreme points by the formulae

\[
v_{\text{min}} = c - C\text{di}/(1 + d_i^T C d_i)
\]

and \( v_{\text{max}} = c + C\text{di}/(1 + d_i^T C d_i) \) for direction vector \( d_i \).

Example
The following code generates a random 3D ellipsoid and computes its most extreme point in the $[1,-0.5,0.1]$ direction. This point is indicated by 'x' on the plot.

```matlab
A=rand(6,3);E=defell(A'*A,rand(3,1)); c=elldrxt(E, [1,-0.5,0.1]); ax=view2del(E);
axes(ax(1)),hold on,plot(c(1),c(2),'rx');
axes(ax(2)),hold on,plot(c(1),c(3),'rx');
axes(ax(3)),hold on,plot(c(2),c(3),'rx'); % nofigs=1
```

Figure 4.28: Demo figure to routine ELLDRXT.
**Purpose**

The purpose of **ELLICOV** is to extracts the matrix of an ellipsoid.

**Usage**

```matlab
function C=ellicov(E);
```

where the variables used are:

- `E([1 0;1 0])` - ellipsoid
- `C[1]` - matrix of ellipsoid

**Brief Description of the Algorithm**

This routine simply extracts the ‘covariance’ matrix of the ellipsoid from its GBT representation matrix.

**Example**

This demo defines a random ellipsoid and extracts its covariance matrix:

```matlab
A=rand(3,3);E=defell(A*A',rand(3,1));C=ellicov(E)
```

```
C =
  1.3208  1.2929  0.90015
  1.2929  1.5404  0.77393
  0.90015  0.77393  0.66755
```
ELLINT

Purpose

The purpose of ELLINT is to compute an ellipsoid around the intersection of two ellipsoids.

Usage

function [E,P]=ELLINT(e1,e2,nord);

where the variables used are:

- e1[[1 0; 0 1]] - first ellipsoid
- e2[[1 0; 1 0]] - second ellipsoid
- nord[2] - order of approximation in each dimension
- E[[1 0; 1 1]] - ellipsoid around the intersection
- P - approximate polytope around the intersection

Brief Description of the Algorithm

The algorithm first approximates each ellipsoid by external polytopes using ‘ellapprx’, then takes the intersection of the two polytopes and computes an ellipsoid around the vertices of the intersection using the ‘sel’ routine. The routine can be easily modified to obtain internal ellipsoid to the intersection by using the internal polytopes produced by ‘ellapprx’ and by using ‘intell’ to compute an internal ellipsoid in the intersection.

Example

The following code illustrates the computation of the intersection of two random ellipses:

A=rand(10,2);E1=defell(A'*A);
A=rand(10,2);E2=defell(A'*A,[1 0]');
ax=view2del(E1);view2del(E2,[1 2],'b',ax);
E=ellint(E1,E2,10);view2del(E,[1 2],'r',ax);
**ELLINTC**

**Purpose**

The purpose of **ELLINTC** is to compute minimum convex combination of two ellipsoids.

**Usage**

\[
\text{function } [E,P]=\text{ELLINTC}(e1,e2);
\]

where the variables used are:

- `e1[[1 0;1 0]]` - first ellipsoid
- `e2[[1 0;1 0]]` - second ellipsoid
- `E[[1 0;1 1]]` - ellipsoid around the intersection
- `P` - approximate polytope around the intersection

**Brief Description of the Algorithm**

The minimum volume convex combination of the two ellipsoids is computed by minimizing the relevant cost function.

**Example**

The following code illustrates the computation of the intersection of two random ellipses:

```plaintext
A=rand(10,2);E1=defell(A'*A);
A=rand(10,2);E2=defell(A'*A,[1 0]');
ax=view2del(E1);view2del(E2,[1 2],ax);
E=ellintc(E1,E2);view2del(E,[1 2],'r',ax);
```
ELLRANGE

Purpose

The purpose of ELLRANGE is to compute the width (range) of ellipsoid in each dimension.

Usage

```
function [vmin,vmax]=ellrange(P);
```

where the variables used are:

- $P[[1 \ 0; 1 \ 0]]$ - ellipsoid
- $vmin[-1]$ - low-vector of range
- $vmax[1]$ - high vector of range

Brief Description of the Algorithm

The range vector of the ellipsoid in each axes direction is computed directly by the formula $c \pm \sqrt{\text{diag}(P)}$.

Example

The following code generates a random 3D ellipsoid and encases it into an axes aligned box:

```
A=rand(4,3)-0.5;E=defell(A'*A,rand(3,1));
[a,b]=ellrange(E);B=defbox(a,b);
ax=view2d(B,[1 2 3]);view2del(E,[1 2 3],'b',ax);
```
Purpose

The purpose of ELLTRAN is to compute the affine transform of an ellipsoid.

Usage

\[
\text{function } P = \text{elltran}(P0, A, a);
\]

where the variables used are:

- \(P0\) - ellipsoid
- \(A\) - affine transform matrix
- \(a\) - offset of affine transform
- \(P\) - ellipsoid

Brief Description of the Algorithm

The covariance matrix is transformed by \(P_1 = APA^T\) and its centre by \(c_1 = A \text{centre}(E) + c\).

Example

The following code generates a random 3D ellipsoid and transforms it into another ellipsoid:

\[
A = \text{rand}(4,3)-0.5; E = \text{defell}(A'*A, \text{rand}(3,1));
E1 = \text{elltran}(E, \text{rand}(3,3)-0.5);
ax = \text{view2del}(E, [1 2 3]); \text{view2del}(E1, [1 2 3], 'r', ax);
\]
Purpose
The purpose of **FACES** is to extract half-space inequalities of polytope.

Usage

```matlab
function [F,nh,ne]=faces(P);
```

where the variables used are:

- **P[1 0;1 1]** - polytope
- **F** - list of half-space inequalities with equalities twice listed
- **nh** - number of proper half-space inequalities
- **ne** - number of equalities

Brief Description of the Algorithm
This routine lists all the half-space inequalities which define a given polytope. In case of degenerate polytopes each equality constraint defines two inequalities in the list. The algorithm is a straightforward extraction of inequalities from the matrix representation of the polytope.

Example
The following code defines a random polytope and extracts its facet inequalities.
Here faces \( F = [A \ b] \) means the system of inequalities \( Ax \leq b \):

```matlab
P=convh(rand(5,3));view2d(P,[1 2 3]);F=faces(P)
```

\[
\begin{array}{cccc}
-0.50521 & -0.1023 & -0.85691 & -0.62927 \\
0.039752 & 0.99814 & -0.046322 & 0.82735 \\
0.99591 & 0.034624 & 0.083485 & 1.0009 \\
0.9558 & -0.17157 & 0.23875 & 0.94221 \\
-0.64534 & -0.1293 & 0.75288 & 0.04127 \\
-0.72567 & -0.63473 & -0.26556 & -0.87175 \\
\end{array}
\]
Purpose

The purpose of **FACET** is to compute the polytope of a facet in lower dimension.

Usage

```matlab
function [fac,P1]=facet(P,plno);
```

where the variables used are:

- `P(1:0;1:0)` - polytope
- `plno[1]` - index of facet
- `fac[0]` - facet polytope in lower dimension
- `P1[0]` - same dimensional facet polytope

Brief Description of the Algorithm

The algorithm is based on the routine ‘planeigh’ which lists the indices of neighbouring facets and adjacent vertices for a given facet. The matrix representation of the degenerate polytope forming the facet is built up from the information obtained by ‘planeigh’.

Example

The following code defines a random polytope and extracts its facet inequalities. Here faces $F = [A \ b]$ means the system of inequalities $Ax \leq b$:

```matlab
P=convh(rand(5,3));ax=view2d(P,[1 2 3]);F=facet(P,2)
F =
  0  32.2204e-016
 0.86562  -0.5007  0.78979
-0.9481  -0.31798  -0.79853
 0.51248  0.8587  1.0554
 0.868  -0.076771  -1
 0.53765  0.9082  -1
1.2068  0.50887  -1
view2d(F1,[1 2 3],'b',ax);
```
Purpose

The purpose of FCONVH is to compute the intersection of a polytope and a set of half-spaces.

Usage

```matlab
function [P1,acc]=fconvh(H,P);
```

where the variables used are:

- `H([1 1;-1 1])` - list of half-spaces
- `P([0 eps;1 1;-1 1 -1;1 -1])` - polytope
- `P1([0 eps;1 1;-1 1 -1;1 -1])` - polytope
- `acc[eps]` - maximum fitting error of adjacent vertices and facets

Brief Description of the Algorithm

The updating with a set of half-spaces is based on a sequence of updatings with each half-space one-by-one using the ‘update’ routine.

Example

The following code defines a random set of inequalities to define a set of half-spaces and computes the intersection of these:

```matlab
H=[rand(10,3)-0.5 ones(10,1)];P=fconvh(H);
view2d(P,[1 2 3]);
```
Purpose

The purpose of **findsimp** is to find largest (d-1)-dimensional simplex from a set of points.

Usage

```matlab
function [V1,B1]=findsimp(V);
```

where the variables used are:

- **V** - set of d-dimensional points
- **V1** - set of points spanning (d-1)-dimensional simplex
- **B1** - Boolean vector to say which points are selected from 'V' for the simplex

Brief Description of the Algorithm

The algorithm first finds the diameter, i.e. two most distance points. Then the most distant new point is added recursively. If the distance of a new point is less than DELTA then the points lie in a d1\|d dimensional affine subspace. The first d1 points are taken to form the vertices of the output simplex defined by 'V1'.

Example

The following code defines a random set of points, project them into a higher dimensional space and selects a simplex of dimension less by 1 :

```matlab
V0=rand(6,3);A=rand(4,3);V=V0*A'
```

```
V =
0.34102 0.67848 0.29603 0.32974
0.24254 0.67749 0.39646 0.37359
0.49523 1.3051 0.61233 0.59822
0.17037 0.66893 0.31039 0.26505
0.22587 0.70727 0.21211 0.19832
0.38989 1.0542 0.42422 0.41606
```

```
V1=findsimp(V)
```

```
V1 =
0.49523 1.3051 0.61233 0.59822
0.22587 0.70727 0.21211 0.19832
0.24254 0.67749 0.39646 0.37359
0.34102 0.67848 0.29603 0.32974
```
Purpose

The purpose of FIRSTEQU is to combine points near the first point in a list.

Usage

```matlab
function [v,nV]=firstequ(V);
```

where the variables used are:

- `V` - list of points
- `v` - combined point
- `nV` - list of the rest of the points

Brief Description of the Algorithm

This routine finds which points in the set ‘V’ are DELTA near to the first point and combines these with the first point by taking their mean value. The rest of the are listed unchanged in ‘nV’. ‘firstequ’ is mainly used by ‘vreduce’ and it has little use of its own.

Example

The following code illustrates the use of FIRSTEQU on a four point set:

```matlab
V=[1 1;1 2;1 1+1e-15;3 1];V
V =
1 1
1 2
1 1
3 1
[v,nV]=firstequ(V)
```

```matlab
v =
1 1
nV =
1 2
3 1
```
Purpose

The purpose of \textbf{FV} is to produce facet-vertex Boolean table of adjacency.

Usage

\texttt{function [FV,nh,ne,acc]=fv(H);}

where the variables used are:

\begin{align*}
\text{H} & : \text{0 eps;1 1;-1 1;-1 -1;1 -1]} & \text{- polytope} \\
\text{FV} & : \text{[0 1;1 0]} & \text{- vertex-facet adjacency table} \\
\text{nh} & : \text{number of face inequalities} \\
\text{ne} & : \text{number equations} \\
\text{acc} & : \text{largest fitting error of facets-vertices}
\end{align*}

Brief Description of the Algorithm

First the list of facet inequalities and list of vertices is extracted form the polytope. Then each facet is tested for fitting against each vertex with DELTA accuracy. Finally, the list of vertex indices, for which fitting was found, is listed for each facet. The unfilled matrix entries of ‘FV’, in which each row lists the vertex indices fitting to a facet, are filled with zeros.

Example

The following code defines a random polytope and extracts its facet-vertex adjacency table:

\begin{verbatim}
P=convh(rand(5,3));fv(P)
fv(P) =
2 3 4
1 3 4
1 2 4
1 2 5
1 3 5
2 3 5
\end{verbatim}
Purpose

The purpose of **FVFITACC** is to extract the maximum fitting error of vertices and facets.

Usage

```matlab
function mu=fvfitacc(P);
```

where the variables used are:

- `P([[0 eps; 1 1; -1 1; -1 -1; 1 -1]])` - polytope
- `mu[eps]` - maximum fitting error

Brief Description of the Algorithm

The worst-case fitting accuracy is extracted either from the ‘P(1,2)’ for $d = 1$ or the ‘P(1,3)’ entries for $d > 1$.

Example

The following code defines a random polytope and FVFITACC extracts its facet-vertex worst case fitting accuracy.

```matlab
P=convh(rand(5,3));fvfitacc(P)
fvfitacc(P) =
2.2204e-016
```
Purpose
The purpose of \texttt{F\_LP} is to calculate subgradient of cost for LP\_CENT.

Usage

\begin{verbatim}
function [f,g]=f_lp(x,A,p),
\end{verbatim}

where the variables used are:

- \texttt{x} - vector \texttt{x} (row vector)
- \texttt{A} - set of points, each row represents a point
- \texttt{p} - parameter of Chebyshev norm
- \texttt{f} - value of \texttt{F(x)}
- \texttt{g} - subgradient of \texttt{F(x)} (row vector)

Brief Description of the Algorithm
Calculates subgradient of \( F(x) = \max(\sum((\text{abs}(\text{ones}(\text{size}(A,1),1)*x-A).^p)^T)^T )\)

Example
This routine is not stand-alone and is only demonstrated in connection with the ‘lpcent’ routine.

\begin{verbatim}
disp('See demo of lp\_cent');
\end{verbatim}
**Purpose**

The purpose of **GBT627** is to convert from GBT 6.1 formats to GBT 7.0 format.

**Usage**

```
function P2=gbt627(P);
```

where the variables used are:

- \( P = [2\ 2; \ 1\ 1; \ -1\ 1; -1\ -1; \ 1\ -1; 2\ 0; 1\ 0; 2\ 0; 1\ 0] \) - polytope
- \( P2 = [0\ \text{eps}; \ 1\ 1; -1\ 1; -1\ -1; 1\ -1] \) - polytope

**Brief Description of the Algorithm**

The adjacency tables extracted from the GBT 6 format are used to re-compute the worst-case fitting accuracy. The final GBT 7 representation is built up from the list of facet inequalities and vertices.

**Example**

The following code defines a type 4 polytope representation (in 1D) and converts it to the type 0 representation of GBT 7.0:

```
P=[2\ 2; \ 1\ 1; \ -1\ 1; -1\ -1; \ 1\ -1; 2\ 0; 1\ 0; 2\ 0; 1\ 0];
P1=gbt627(P);P1
P1 =
0 0
1 1
-1 1
-1 -1
1 -1
```
GBT726

**Purpose**

The purpose of **GBT726** is to convert GBT 7 format to type 4 format of GBT 6.1.

**Usage**

```matlab
function P2=gbt726(P);
```

where the variables used are:

- `P` - polytope
- `P2` - polytope in type 4 format

**Brief Description of the Algorithm**

The vertex-vertex and vertex-facet adjacency tables are produced by the ‘vvf’ routine. If a vertex is found to have more than \( d \) adjacent vertices than the conversion is impractical and the output is empty matrix. Otherwise the GBT 6 type 4 format is build up.

**Example**

The following code defines a type 0 polytope representation (in 1D) and converts it to the type 0 representation of GBT 7.0:

```matlab
P=[0 eps;1 1;-1 1;-1 -1; 1 -1];P
P =
02.2204e-016
 1   1
-1   1
-1  -1
 1  -1
P1=gbt726(P);P1
P1 =
  4  2
  1  1
-1  1
-1 -1
  1 -1
  2  0
  1  0
  0  0
  0  0
```
Purpose

The purpose of INELL is to test whether a given point is in an ellipsoid.

Usage

```
function bool=inell(p,E);
```

where the variables used are:
- \( p[0] \) - point
- \( E[1 0;1 0] \) - ellipsoid
- \( bool[1] \) - Boolean value

Brief Description of the Algorithm

The satisfaction of the inequality

\[
(p - \text{centre}(E))^T \text{ellcov}(E)^{-1}(p - \text{centre}(E)) < 1
\]

is tested.

Example

The following code generates a random ellipsoid and tests whether the origin is contained in it:

```matlab
A=rand(5,3);E=defell(A'*A(rand(3,1))); % no.figs=1
ax=view2del(E,[1 2 3]);inell([0 0 0],E)
inell([0 0 0],E) = 1
axes(ax(1)),hold on,plot(0,0,'rx');
axes(ax(2)),hold on, plot(0,0,'rx');
axes(ax(3)),hold on, plot(0,0,'rx');
```
**Purpose**

The purpose of `intell` is to find a maximal volume ellipsoid within a polytope.

**Usage**

```matlab
function E=intell(C,options),
```

where the variables used are:

- `C[0 eps;1 1;-1 1;1 1;-1 1]` - polytope in GBT type 0 format
- `options[1.e-4,150]` - termination tolerance, number of iterations
- `E` - ellipsoid

**Brief Description of the Algorithm**

The maximum volume internal ellipsoid to a polytope is optimized by successive space transformations. First the vertices are found and the polytope shifted so that it contains the origin inside. The facet inequalities are brought to a form with 1s on the right-hand-side. The left hand-side is manipulated in the optimization procedure which successively obtains larger and larger volume ellipsoids inside the polytope.

**Example**

The following code defines a random polygon and plots a maximal volume tight ellipsoid inside it:

![Demo figure to routine INELL](image-url)
Purpose

The purpose of **INTERSCT** is to compute the intersection of two polytopes.

Usage

```matlab
function G=intersct(G1,G2);
```

where the variables used are:

- `G1[[1 eps;1 1]]` - first polytope
- `G2[[1 eps;1 1]]` - second polytope
- `G[[1 eps;1 1]]` - polytope

**Brief Description of the Algorithm**

The algorithm computes the intersection of two polytopes by updating the second polytope with all the half-space inequalities of the first one using the ‘fconvh’ routine.

**Example**

The following code defines two random polytopes in 3D, computes their intersection and plots the intersection in a separate figure.

```matlab
P1=convh(rand(10,3));P2=convh(rand(16,3)-0.1);
P=intersct(P1,P2);ax=view2d(P1,[1 2 3]); % no.fig=2
view2d(P2,[1 2 3],’b’,ax);figure;view2d(P,[1 2 3]);
```
CHAPTER 4. PROGRAMMING REFERENCE

Figure 4.30: Demo figure to routine INTERSC.

INTERSC

Purpose

The purpose of INTERSC is to compute the intersection of two polytopes.

Usage

function G=intersc(G1,G2);
where the variables used are:

- \( G_1 \) - first NPS
- \( G_2 \) - second NPS
- \( G \) - resulting NPS

**Brief Description of the Algorithm**

The algorithm computes the intersection of two NPSs by updating the second NPS with all the polytopes of the first one using the ‘intersct’ routine. See `demo12` for an illustration of `intersct`.

**Purpose**

The purpose of `INTEST` is to test whether a set of points is in polytope.

**Usage**

```matlab
function Boole=intest(V,P);
```

where the variables used are:

- \( V[0] \) - array of points
- \( P[[0 eps;1 1;-1 1;-1 -1;1 -1]] \) - polytope
- \( Boole[1] \) - Boolean value

**Brief Description of the Algorithm**

First the half-space inequalities of the polytope are extracted and then each point in ‘V’ is tested whether satisfies all the inequalities.

**Example**

The following code defines a random polytope and a random point and it tests whether the point is inside the polytope:

```matlab
P=convh(rand(10,3));ax=view2d(P,[1 2 3]);
c=rand(3,1);intest(c,P);
axes(ax(1)),hold on,plot(c(1),c(2),'bx');
axes(ax(2)),hold on, plot(c(1),c(3),'bx');
axes(ax(3)),hold on, plot(c(2),c(3),'bx');
```
INTPOINT

Purpose

The purpose of INTPOINT is to find an internal point within a polytope.

Usage

function p=intpoint(P);

where the variables used are:

P([[0 eps;1 1;-1 1;-1 -1; 1 -1]]) - polytope
p - internal point

Brief Description of the Algorithm

First the mean of the vertices is tried for suitability as internal point. If this is not suitable then random points are generated around the mean to find one which is suitable. If the random tests are unsuccessful then an empty matrix is returned. If the actual dimension of the polytope is less than $d$ than an empty matrix is returned.

Example

The following code finds an internal point within a randomly generated polygon:

P=convh(rand(8,2));p=intpoint(P);
ax=view2d(P);hold on;plot(p(1),p(2),’x’);% no.figs=1
LPCENT

Purpose

The purpose of **LPCENT** is to calculate the lp-centre of a set of points.

Usage

\[
\text{function } [x,f]=\text{lpcent}(A,p,\text{options}),
\]

where the variables used are:

- \( A \) - a set of points
- \( p[2] \) - norm-parameter \( p >= 1 \)
- \( \text{options}[1.e-4,150] \) - tolerance and max. no. iterations
- \( x \) - center of ‘\text{convh}(A)’
- \( f \) - radius of ‘\text{convh}(A)’

Brief Description of the Algorithm

The maximum of the \( l_p \) norms of the distances to members of the set is minimized by a subgradient method with space dilatation.

Example

The following code generates a random polygon and computes its Chebyshev center in 2-norm:
LTRAN

Purpose

The purpose of **LTRAN** is to perform affine transformation on a polytope.

Usage

```
function [Pt,acc]=ltran(P,A,a);
```

where the variables used are:

- **P** - polytope
- **A** - transform matrix
- **a** - offset vector after transformation
- **Pt** - transformed polytope
- **acc** - error bound of vertex-facet adjacency

Brief Description of the Algorithm

If the transform matrix ‘A’ is a full-rank square matrix then the new set of vertices and inequalities is computed by direct matrix transformations. In any other case the set of vertices is transformed and the convex hull of the transformed set is computed by ‘convh’ to obtain the transformed polytope.

Example

The following demo generates a random rectangle and transforms it by a random matrix:

```
x=rand(1,2);P=defbox(x,x+[0.5,0.3]);
A=rand(2,2)
A =
0.15531 0.22431
0.99579 0.088014
a=rand(2,1)
a =
0.40688
0.33022
P1=ltran(P,A,a);
ax=view2d(P);view2d(P1,[1 2],'b',ax);
```
Purpose

The purpose of MAXFACE is to compute the vector of maximum no. faces for given number of vertices.

Usage

function f=maxface(n,d);

where the variables used are:

n[3] - number of vertices
d[2] - dimension
f[3] - vector of max. no. faces

Brief Description of the Algorithm

The formulas are different for even and odd dimensions ‘d’: For even ‘d’ the maximum number of ‘j’-dimensional faces for ‘n’ vertices is:

\[
\sum_{i=1}^{d/2} \frac{n}{n-i} \binom{n-i}{i} \binom{i}{2i-j-1}
\]

For odd ‘d’ the maximum number of ‘j’-dimensional faces for ‘n’ vertices is:

\[
\sum_{i=1}^{(d-1)/2} \frac{j+2}{n-i} \binom{n-i}{i+1} \binom{i+1}{2i-j}
\]

Example

The following code computes the maximum potential number of faces for 10 vertices of a 6D polytope. The kth (k=1,2,\ldots,5) entry is the maximum number of k-dimensional faces:

```
maxface(10,6)
maxface(10,6) =
95
120
185
150
50
```
MAXVERT

Purpose

The purpose of MAXVERT is to compute the vector of maximum no. vertices for given number of facets.

Usage

function m=maxvert(n,d);

where the variables used are:

n[3] - number of facets
d[2] - dimension
f[3] - vector of max. no. vertices

Brief Description of the Algorithm

Computing the maximum possible number of vertices for ‘n’ hyperplanes is a dual problem of that solved by MAXFACE. This number is equal to the maximum number of ‘d-1’ dimensional faces for ‘n’ vertices.

Example

The following code computes the maximum potential number of vertices for 10 facets of a 6D polytope:

maxvert(10,6)
maxvert(10,6) =
50
Purpose

The purpose of MCVOL is to estimate the volume of the intersection of ellipsoids.

Usage

```matlab
function vol=mcvol(E,n);
```

where the variables used are:

- `E[[1 0;1 0]]` - array of ellipsoids
- `n[1]` - number of grid points in each dimension
- `vol` - volume of ellipsoid

Brief Description of the Algorithm

The algorithm is based on counting the portion of grid points falling inside the ellipsoid intersection. The grid points are uniformly distributed in the smallest axes-aligned box covering all the ellipsoids.

Example

The following code estimates the volume of the intersection of a set of three random 3D ellipsoids:

```matlab
E=[];for i=1:3, A=rand(6,3)-.5;E=[E; defell(A'*A)];end;
ax=view2del(E(1:4,:),[1 2 3]);
view2del(E(5:8,:),[1 2 3],ax);
view2del(E(9:12,:),[1 2 3],ax);
volume=mcvol(E,20)
volume = 0.469
```
Purpose

The purpose of MINDIM is to minimize the dimension of the representation of a polytope.

Usage

function [P1,affsbsp]=mindim(P);

where the variables used are:

P([1 eps; 1 0]) - polytope
P1[0] - polytope
affsbsp([1 0]) - equation of affine subspace

Brief Description of the Algorithm

This routine computes a minimal dimensional matrix representation of a polytope. The algorithms accepts the information already available in the polytope representation on the actual dimension $d_a$ of the polytope. Using $d_a$, the polytope is orthogonally projected onto a $d_a$-dimensional subspace parallel to the polytope. The data on vertices and faces is extracted from the projection to obtain a $d_a$ dimensional polytope representation.

Example

The following code defines a random 2D triangle in 3D space and displays it and computes its minimal dimensional representation:

```matlab
V=rand(3,3);P=convh(V);view2d(P,[1 2 3]); % no.figs=2
P1=mindim(P)
P1 =
0 32.2204e-016
-0.1333 -0.99108 0.11087
0.99858 0.053287 0.64997
-0.83208 0.55465 -0.3258
0.63167 0.36023 -1
0.2909 -0.151 -1
0.66162 -0.20086 -1
figure;view2d(P1);
```
Figure 4.33: Demo figure to routine MINDIM.

Figure 4.34: Demo figure to routine MINDIM.

**PERIMETER**

**Purpose**

The purpose of **PERIMETER** is to compute the perimeter of a 2D NPS.

**Usage**

```matlab
function Pl=perimeter_(P);
```
where the variables used are:

- $P$ - a 2 dimensional NPS
- $P_l$ - length of perimeter

**Brief Description of the Algorithm**

The length of the perimeter is build up from linear sections found on the boundary of the NPS.

See [DEMO12](#) for an illustration of `PERIMETER`.

---

### PLANEIGH

<table>
<thead>
<tr>
<th>Purpose</th>
<th>PLANEIGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>The purpose of <strong>PLANEIGH</strong> is to list neighbouring facet planes.</td>
<td></td>
</tr>
</tbody>
</table>

#### Usage

```matlab
function [plist,vlist]=planeigh(P,plno);
```

where the variables used are:

- $P[[1 0;1 0]]$ - polytope
- $plno[1]$ - index of facet
- $plist[[-1 0]]$ - facet-plane equations of neighbouring facets
- $vlist[1]$ - index list of vertices on facet

#### Brief Description of the Algorithm

The algorithm first computes the facet-vertex adjacency list. The facets which share $d - n_v - 1$ vertices with the facet are listed as neighbours where $n_v$ is the degree of degeneracy of the polytope (i.e. ‘$P(1,1)$’ or the number of equalities).

#### Example

The following code defines a random 3D polytope and first lists its facet equations and vertices, then it lists the indices of the facets and vertices adjacent with the 2nd facet:

```matlab
P=convh(rand(7,3));F=faces(P)
F =
    -0.89582  -0.37756  -0.23441  -0.445
    0.22752   0.83445   0.50192  -0.031682
    0.58505   0.10411  -0.80429  -0.012039
    0.56935   0.54579  -0.61478   0.41823
```
0.036286  -0.18431  0.9822   0.60463
0.48573   0.84349  -0.22934 0.83131
-0.80247  0.44185  0.40101  0.99086
0.037047  0.99086  -0.12972 0.86909

V=vertices(P)
V =
0.83888  0.697  0.7154
0.070377  0.89836  0.18245
0.30476  0.28565  0.27363
0.10605  0.49012  0.70365
0.31349  0.92143  0.42808
0.40603  0.96424  0.78154

[plist,vlist]=planeigh(P,2)
plist =
1  3  5
vlist =
1  3  4

view2d(P,[1 2 3]); % no.fig=1

Figure 4.35: Demo figure to routine PLANEIGH.
Purpose

The purpose of **PLANFIT** is to fit a hyperplane to a set of points.

Usage

```matlab
function [a,b,fitacc]=planfit(V);
```

where the variables used are:

- **V** - set of points
- **a** - left hand side of linear equation
- **b** - right hand side of linear equation
- **fitacc** - maximum fitting error equation to points

Brief Description of the Algorithm

Fits a hyperplane described by \( a^T x = b \) to \( d \) vertices given in the rows of the \( d \times d \) matrix \( V \). The algorithm uses SVD to find a normal to the plane containing the points.

Example

The following code generates 3 random points in 3D space and computes a normalized equation of a hyperplane fitting to all three points (a normalized equation has 2-norm 1 of the linear form vector of coefficients):

```matlab
V=rand(3,3)
V =
0.21857 0.28868 0.89092
0.79885 0.97614 0.87092
0.27477 0.90928 0.95531
[a,b,fitacc]=planfit(V)
a =
0.17242 -0.11709 0.97804
b =
0.87524
fitacc =
0
```
**POLADD**

**Purpose**

The purpose of **POLADD** is to compute the sum of two polytopes.

**Usage**

```matlab
function [O,acc]=poladd(O1,O2);
```

where the variables used are:

- `O1[[1 eps;1 0]]` - first polytope
- `O2[[1 eps;1 0]]` - second polytope
- `O[[1 eps;1 0]]` - sum of polytopes
- `acc` - max. error of vertex-facet adjacency

**Brief Description of the Algorithm**

The algorithm first lists the vertices of each polytope. All combinations of sums of vertices from the first and second polytope are listed in a set of points $V_{1+2}$. The convex hull of $V_{1+2}$ is computed to obtain the polytope representing the Minkowski sum.

**Example**

The code below generates a random polygon and adds a small rectangle to it:

```matlab
P=convh(rand(6,2));ax=view2d(P);B=defbox(-0.1*[1 1],0.1*[1 1]);
P1=poladd(P,B);view2d(P1,[1 2],’b’,ax); % no.figs=1
```
Figure 4.36: Demo figure to routine POLADD .

POLADDI

**Purpose**

The purpose of **POLADDI** is to compute diamond inside the sum of two polytopes.

**Usage**

```matlab
function [O,acc]=poladdi(O1,O2,dplus,accur);
```

where the variables used are:

- `O1[[1 eps;1 0]]` - first polytope
- `O2[[1 eps;1 0]]` - second polytope
- `O[[1 eps;1 0]]` - generalized diamond inside the sum
- `dplus` - number of vertices to be added to the diamond
- `accur` - max. error of vertex-facet adjacency
  ( `dplus` is not used if `accur` is defined)

**Brief Description of the Algorithm**

The algorithm first lists the vertices of each polytope. All combinations of sums of vertices from the first and second polytope are listed in a set of points $V_{1+2}$. The routine ‘diamet’ is used to find the two most distant vertex sums which are called ‘diameter 1’. The remaining points are projected to the orthogonal complement of ‘diameter 1’ and the two most distant points are determined from the projection. The originals these two points are used to defined a co-diameter called ‘diameter
2’. The rest of the points is projected onto the orthogonal complement of the space spanned by the vectors of ‘diameter 1’ and ‘diameter 2’. This procedure is continued for the selection of ‘d’ diameters. The convex hull of the set of endpoints of all diameters selected is computed to obtain a polytope inside the sum. If ‘dplus’ is given then a further ‘dplus’ number of most distant points are added to the convex hull. If both ‘dplus’ and ‘accur’ are given then further points from the sum of vertices are added until all remaining points remaining outside are to a distance less than ‘accur’ from the faces of the polytope. Hence ‘accur’ is a specifiable accuracy requirement for the internal polytope ‘O’ inside the exact Minkowski sum of of ‘O1’ and ‘O2’. If ‘accur’ is defined as an input to ‘eladdi’ then ‘dplus’ is ignored. This algorithm is also called the generalized diamond method.

Example

The code below generates a random polygon P and computes a polygon with only four vertices inside a the sum of P and a small rectangle B. The exact sum is indicated with dashed lines:

```matlab
P=convh(rand(6,2));ax=view2d(P);B=defbox(-0.1*[1 1],0.1*[1 1]);
P1=poladdi(P,B);view2d(P1,[1 2],'b',ax);
P2=poladd(P,B);view2d(P2,[1 2],'g:',ax); % no.fig=2
```

![Figure 4.37: Demo figure to routine POLADDI.](image-url)
The purpose of POLCENT is to compute the centre of a polytope.

Usage

\[
\text{function } [c,r]=\text{polcent}(P,\text{typ},p);
\]

where the variables used are:

- \(P[[1 \text{ eps};1 0]]\) - polytope
- \text{typ}['cheb'] - 'mass' or 'cheb'
- \(p\) - order of norm \(p \geq 1\)
- \(c[0]\) - centre of polytope
- \(r[0]\) - radius of the polytope

Brief Description of the Algorithm

The algorithm computes the mass centre of a polytope as the mean of its vertices. The \(l_p\) Chebyshev centre for \(p \geq 1\) is computed by the routine 'lpocent'. For \(p = \infty\) the centre of the smallest axis aligned box containing the polytope is computed.

Example

The next code generates a random 3D polytope and computes its 2-norm Chebyshev centre:

\[
P=\text{convh(rand}(8,3)); [c,r]=\text{polcent}(P,'cheb',2)
\]

\[
c =
\begin{bmatrix}
0.42091 \\
0.50441 \\
0.38427
\end{bmatrix}
\]

\[
r = 0.579
\]

\[
ax=\text{view2d}(P,[1 2 3]);
\text{axes(ax(1)),hold on,plot(c(1),c(2),'bx');}
\text{axes(ax(2)),hold on, plot(c(1),c(3),'bx');}
\text{axes(ax(3)),hold on, plot(c(2),c(3),'bx');}
\]

% no.figs=1
**POLDUAL**

**Purpose**

The purpose of **POLDUAL** is to compute the dual of a polytope.

**Usage**

```matlab
function [D,acc]=poldual(P);
```

where the variables used are:

- `P` - polytope
- `D` - dual of a polytope
- `acc` - max. error of vertex-facet adjacency

**Brief Description of the Algorithm**

The dual of a polytope $P \subset \mathbb{R}^d$ is defined by

$$P_{\text{dual}} = \{ x \in \mathbb{R}^d \mid x^T y \leq 1 \ \forall y \in P \}$$

The dual of a polytope $P$ is only defined if the origin is contained in $P$, otherwise the output ‘D’ is an empty matrix. If the vertices of polytope $P$ are $y_k, k \in K$, then the dual can be computed as the intersection of half-spaces

$$\cap_{k \in K} \{ x \in \mathbb{R}^d \mid x^T y_k \leq 1, \ k \in K \}$$
This fact is used to define the facet list of the dual polytope. If \( h^T x \leq h_1 \) is a facet-inequality then \( h/(h_1)^{-1} \) is a vertex of the dual polytope. This is used to list the set of vertices of the dual polytope. The worst-case facet-vertex fitting accuracy of the dual polytope is re-computed by the ‘fv’ routine.

**Example**

The following code generates a random polygon and computes and displays its dual in dashed lines:

```matlab
P=convh(3*rand(10,2)-1.5);P1=poldual(P);
ax=view2d(P);view2d(P1,[1 2],'b:',ax); % no.figs=1
```

![Projection to 1-2 axes](image)

Figure 4.39: Demo figure to routine POLDUAL.
Purpose

The purpose of POLVOL is to compute the volume of a polytope.

Usage

```
function volume=polvol(P)
```

where the variables used are:

- `P` - polytope
- `volume` - volume of polytope

Brief Description of the Algorithm

The algorithm of volume computation is recursive in the dimension of the polytope. The volume of $P$ is computed as the sum of surface areas of the facets with a suitable weighting. If the hyperplane inequality of the $i$th facet is $n_i^T x \leq c_i$, $i \in F$ then the volume is computed as $\sum_{i \in F} h_i \cdot \text{area}(P_i)$ where $P_i$ denotes the $i$th facet of $P$.

Example

The following code generates a random 3D polytope and computes its volume:

```
P=convh(rand(10,3));volume=polvol(P)
volume =
  0.15483
view2d(P,[1 2 3]); % no.fig=1
```

Purpose

The purpose of POLVOL is to compute the volume of a polytope.

Usage

```
function volume=polvol_(P)
```

where the variables used are:

- `P` - polytope
- `volume` - volume of polytope

Example

The following code generates a random 3D polytope and computes its volume:

```
P=convh(rand(10,3));volume=polvol_(P)
volume =
  0.15483
view2d(P,[1 2 3]); % no.fig=1
```
where the variables used are:

- P - an NPS
- volume - volume of the NPS

**Brief Description of the Algorithm**

The algorithm of `polvol`, relies on the use of `polvol` and a sieve formula of set unions which eliminates double counting of regions in the volumetric sum.

See `demo12` for an illustration of the use of `polvol`.
Purpose

The purpose of **PROJELL** is to project ellipsoid onto subspace.

Usage

```
function EP=projell(E,SM);
```

where the variables used are:

- \( E \) - ellipsoid
- \( SM \) - spanning vectors of subspace
- \( EP \) - projected ellipsoid

Brief Description of the Algorithm

\( S_p \) is defined as a matrix of column vectors spanning the subspace onto which projection is to be done. The covariance matrix of the projection is \( M_p = T'MT \) and its centre is \( T'c \) where the covariance of the original ellipsoid is \( M \) and its centre is \( c \). \( T \) is the projection matrix \( T = (S_pS_p^T)^{-1}S_p \).

Example

The following code generates a random 3D ellipsoid, displays and projects it onto the \([1 1 1][x1 x2 x3]'=1\) plane. The projection is presented in 2D:

```
A=rand(6,3);E=defell(A'*A,rand(3,1));
view2del(E,[1 2 3]);
[U,S,V]=svd([1 1 1]);SM=V(:,2:3);EP=projell(E,SM);
figure;view2del(EP); % no.figs=2
```
Figure 4.41: Demo figure to routine PROJELL.

Figure 4.42: Demo figure to routine PROJELL.

<table>
<thead>
<tr>
<th>PROJPONT</th>
<th>PROJPONT</th>
</tr>
</thead>
</table>

**Purpose**

The purpose of **PROJPONT** is to project point onto polytope.

**Usage**

```matlab
function pt=projpont(p,G);
```
where the variables used are:

\( p[1] \) - points
\( G[[1 \ \text{eps};1 \ 0]] \) - polytope
\( pt[0] \) - projected point

**Brief Description of the Algorithm**

This algorithm looks at all combinations of visible facets and projects the point onto their intersection, then compares the distances to obtain the nearest point to the point ‘p’ on the polytope ‘G’, which is defined as the projected point ‘pt’.

**Example**

The following demo generates a random 3D polytope and projects the \([1 \ 1 \ 1]\) points onto its surface:

```matlab
P=convh(rand(12,3));c=projpont([1 1 1],P);
ax=view2d(P,[1 2 3]);
axes(ax(1)),hold on,plot([c(1);1],[c(2);1],'ro');
axes(ax(2)),hold on,plot([c(1);1],[c(3);1],'ro');
axes(ax(3)),hold on,plot([c(2);1],[c(3);1],'ro');
```
Purpose

The purpose of \texttt{P\_CONV} is to include a point into a convex hull polytope.

Usage

\begin{verbatim}
function [O,acc]=p_conv(xn,P);
\end{verbatim}

where the variables used are:

\begin{itemize}
  \item \texttt{xn[0]} - point
  \item \texttt{P[[1 eps;1 0]]} - polytope
  \item \texttt{O[[1 eps;1 0]]} - convex hull updated
  \item \texttt{acc[eps]} - worst-case accuracy of fits
\end{itemize}

Brief Description of the Algorithm

This algorithm adds a new point to a polytope, i.e. computes the convex hull of the polytope and an outside point. If the new point is DELTA-near to the polytope, then the convex hull is not updated and the new point is discarded. Otherwise First the proper visible faces (from the point) are listed so that the point does not lie in the hyperplane of these facets. Next it is examined whether the point lies on any of the hyperplanes of the facets and these are also listed Then the adjacency table of facets and vertices is computed by the \texttt{fv} routine. The indices of the facets neighbouring the visible facets is determined. The new point an \( d - 1 \) points belonging to both a visible facet and a neighbouring facet form a new facet-hyperplane of the new convex hull. All of these are added to the new facet list. Vertices falling inside the new convex hull are discarded from the list. If they do not contain the new point then old visible facets are erased from the list. The new point is added to the list.

Example

The following code defines a 3D random polygon and computes the convex hull of the polygon and the \([1 1]\) point:

\begin{verbatim}
V=rand(6,2);P=convh(V);P1=p_conv([1 1],P); figs=1
figure;a=view2d(P);view2d(P1,[1 2],’r’,a);
\end{verbatim}
Purpose

The purpose of Projpol is to project polytope onto linear manifold.

Usage

function Pp=projpol(P,spanvecs,offset);

where the variables used are:

P - polytope
spanvecs - spanning vectors of subspace
offset - point in manifold
Pp - projected polytope in same dimension

Brief Description of the Algorithm

spanvecs is a matrix of column vectors spanning the subspace giving the direction of the linear manifold onto which projection is to be done. The vertices of ‘P’ are projected onto the submanifold first and the convex hull of the projections is taken to compute the projection ‘Pp’ which is degenerate, i.e. lower dimensional than the representation space.

Example
The following code generates a random 3D polytope, displays and projects it onto the $[1 \ 1 \ 1][x_1 \ x_2 \ x_3]'=1$ plane. The projection is presented in 2D:

```matlab
P = convh(rand(6, 3));
[u, d, V] = svd([1 1 1]);
Pp = projpol(P, V(:, 2:3), [1 1 1]/3);
ax = view2d(P, [1 2 3]);
view2d(Pp, [1 2 3], 'b', ax);
```
Purpose

The purpose of RANGES is to compute the lower and upper corners of a tight axis aligned box to polytope.

Usage

```
function [vmin,vmax]=ranges(P);
```

where the variables used are:

- \( P[[0 \text{ eps};1 1;-1 1;-1 -1; 1 -1]] \) - polytope
- \( \text{vmin}[-1] \) - lower corner of axis aligned box
- \( \text{vmax}[1] \) - upper corner of axis aligned box

Brief Description of the Algorithm

The algorithm gives the “lowest” and “highest” corners of the smallest axis-aligned box containing the polytope.

Example

The demo defines a random polytope and encases it into an axis aligned box (dashed line):

```
P=convh(rand(10,3));[a,b]=ranges(P);B=defbox(a',b');
view2d(P);view2d(B,[1 2],'b:',a);
```
Purpose

The purpose of SEL is to find the minimal volume ellipsoid around a set of points.

Usage

\[
\text{function } E = \text{sel}(C, \text{options});
\]

where the variables used are:

- \(C[-1;1]\) - array of points
- \(\text{options}[[0.0001,150]]\) - \(['\text{termination tolerance}', 'number of iterations']\)
- \(E[[1 \ 0;1 \ 0]]\) - ellipsoid

Brief Description of the Algorithm

For a set of points \(C\) the problem is first solved for an extended set of points \(C' = \{[x,1]| x \in C\}\) with the constraint that the centre of the ellipsoid is at the origin. If the solution to this has covariance matrix

\[
P' = \begin{bmatrix}
P & r \\
r' & p
\end{bmatrix}
\]

then a solution of the original problem can be obtained as an ellipsoid with centre \(a = -P^{-1}r\) and covariance matrix \(P(1 - p + r'P^{-1}r)\). The method used to solve the minimization problem for the extended set of points \(C'\) is the method of successive space dilatation. The options of optimization are: ‘options(1)’ = the termination tolerance for the coefficients of matrix E ; ‘options(2)’ : maximal number of iterations. The default ‘options’ is [1.e-4,150]

Example

The demo defines a random set of points in 2D and covers them with a tight, i.e. minimal area, ellipse:

\[
V = \text{rand}(20,2); E = \text{sel}(V); a = \text{view2del}(E); \text{hold on}; \text{plot}(V(:,1),V(:,2),'x');
\]
This is a script M-file not a function. Declare DELTA and KAPPA as global variables:

```matlab
global DELTA;global KAPPA;global ERRFACTOR;
```

Presenting user interface to type in KAPPA required:

```matlab
disp('');
disp('Half-edge-size of a hypercube centred to the ');
disp('origin which will include all polytopes');
KAPPA=input('to be used [default of KAPPA is 100]: KAPPA=');
if isempty(KAPPA), KAPPA=100;end;
```

Calculating and print the value of DELTA:

```matlab
disp('The facet-vertex fitting tolerance DELTA is set to the ');
disp('following value for all computations in GBT routines:');
DELTA=KAPPA*1e-13
```
SLICECYL

Purpose
The purpose of \texttt{slicecyl} is to compute the intersection of a cylinder and an axis aligned hyperplane in 3D.

Usage

\begin{verbatim}
function [S,S1]=slicecyl(C,h);
\end{verbatim}

where the variables used are:

- \texttt{C} - \texttt{cylinder matrix representation}
- \texttt{h} - \texttt{hyperplane}
- \texttt{S} - \texttt{slice in 3D representation}
- \texttt{S1} - \texttt{list of polygon vertices of the slice}

Brief Description of the Algorithm
The algorithm is based on computing the intersection points of "h" with the ending ellipsoids by vector algebra.

Example
The following code defines a cylinder, displays it, then computes an intersection slice with a hyperplane and displays the slice polygon on the same plot.

\begin{verbatim}
%defining a cylinder
c = [ 2.0000 0 1.0000 0
     0 1.0000 1.0000 0
     1.0000 1.0000 1.0000 0
     1.0000 -1.0000 0.6000 1.0000
    -1.0000 -0.9000 -1.0000 1.0000 ];

view3dcyl(c); % displaying the cylinder
sp = [ 1 1 -2 -2.5 ]; % slicing hyperplane
[S,SL]=slicecyl(c,sp); % slice computation
SL=[SL;SL(1,:)];

% drawing the contour of the slice in 3D
\end{verbatim}
line(sl(:,1),sl(:,2),sl(:,3));

Figure 4.44: Demo figure to routine sliceyl.m.

Figure 4.45: Demo figure to routine sliceght.
SLICEGBT

**Purpose**

The purpose of **slicegbt** is to compute the intersection of a polytope and a hyperplane.

**Usage**

\[
\text{function } [S,S1]=\text{slicegbt}(h,P);
\]

where the variables used are:
- \(h\) - linear equation \(ax'=b\) in form \(h=[a\ b]\)
- \(P\) - polytope
- \(S\) - slice in d-dimensional representation
- \(S1\) - slice in lowest dimensional representation

**Brief Description of the Algorithm**

The algorithm is based on ‘update’ to compute the facet appearing when a hyperplane cuts a polytope. ‘mindim’ is used to reduce the matrix dimension of the representation.

**Example**

The following code generates a random 3D polytope and computes the slice obtained by intersecting it with the plane \(x_1 + x_2 + x_3 = 0\).

```matlab
P=fconvh([rand(10,3)-0.5 ones(10,1)]);
h=[1 1 1 0]; % hyperplane equation
[S,S1]=slicegbt(h,P);
ax=view2d(P,[1 2 3]);view2d(S,[1 2 3],’b’,ax);
figure;view2d(S);title(’The 2D slice’); % no.figs=2
```
Figure 4.46: Demo figure to routine slicegbt.

Figure 4.47: Demo figure to routine slicegbt.

---

**Purpose**

The purpose of **SPHSEP** is to compute the joining surface area of spheres.

**Usage**
function [Totsurf,Totout,Totint]=sphsep(Sph,appord);

where the variables used are:

Sph([1 0;1 0]) - array of spheres
appord[8] - order of polygon approximation of circles
Totsurf[0] - total joining surface area
Totout[0] - upper bound of joining surface area
Totint[0] - lower bound of joining surface area

Brief Description of the Algorithm

This algorithm first computes the interface-planes between intersecting spheres. For each sphere that part of each interface plane is computed which lies ‘outside’ of all interface-planes of other spheres. The total sum of these surface areas is computed. Finally, the 2D surface area of the convex hull of ‘correct’ intersection-points of interface-planes is subtracted from this sum to obtain ‘Totsurf’, the joining surface area required. The areas are computed by internal and outer polygon approximations; ‘Totout’ and ‘Totint’ are upper and lower bounds of ‘Totsurf’.

Example

The following demo generates three random spheres and computes the total area of their flat meeting surfaces:

\[ C=\text{eye}(3); E=[\text{defell}(C,\text{rand}(3,1)-0.5)]; \]
\[ E=[E;\text{defell}(C,\text{rand}(3,1)+0.5)]; \]
\[ E=[E;\text{defell}(C,\text{rand}(3,1)+[0.5;0.5;-0.5])]; \]
\[ \text{Totsurf}=\text{sphsep}(E,10) \]
\[ \text{Totsurf} = 1.8602 \]
\[ a=\text{view2del}(E(1:4,:),[1 2 3]); \]
\[ \text{view2del}(E(5:8,:),[1 2 3],a); \]
\[ \text{view2del}(E(9:12,:),[1 2 3],a); \% no.figs=1 \]
SUBTRACT

Purpose

The purpose of **SUBTRACT** is to compute the Minkowski difference of two polytopes.

Usage

```matlab
function D=subtract(A,B);
```

where the variables used are:

- `A[[1 eps;1 0]]` - first polytope
- `B[[1 eps;1 0]]` - second polytope
- `D[[1 eps;1 0]]` - difference polytope

Brief Description of the Algorithm

If $v_k, k \in K$ are the vertices of polytope $B$ then the Minkowski difference can be obtained as the intersection of half-spaces

$$h_i^T x \leq c_i - \max_{k \in K} (h_i^T v_k), \quad i \in I$$

where $h_i^T x \leq c_i, \quad i \in I$, are all the half-space inequalities defining polytope $A$. The routine ‘iconvh’ is used to compute the intersection of the new set of half-space inequalities.
Example

The following code generates a random polygon and takes its Minkowski difference with a small rectangle:

```matlab
P=convh(rand(10,2));e=0.01*[1 1];B=defbox(-e,e);% no.fig=1
P1=subtract(P,B);a=view2d(P);view2d(P1,[1 2],'b',a);
```

![Projection to 1-2 axes](image)

Figure 4.49: Demo figure to routine SUBTRACT.
Purpose

The purpose of **SURFAREA** is to compute the surface area of a polytope.

Usage

```matlab
function [A,Area2]=surfarea(P);
```

where the variables used are:

- **P** - polytope
- **A[0]** - surface area
- **Area2** - relative surface area

**Brief Description of the Algorithm**

The surface area is computed by adding up the $d - 1$ dimensional volumes of all facets of the polytope.

**Example**

The following code computes the surface area of a random polytope in 3D:

```matlab
P=convh(rand(10,3));view2d(P,[1 2 3]); % no.figs=1
area=surfarea(P)
area =
2.4868
```
Purpose

The purpose of **UNI** is to compute the union of two point sets.

Usage

```matlab
function U=uni(U1,U2);
```

where the variables used are:

- **U1** - matrix of row vectors
- **U2** - matrix of row vectors
- **U** - union of vector sets

Brief Description of the Algorithm

‘U’ is first initialized as the set ‘U2’ All points in ‘U1’ are tested whether they are DELTA-near to a point in ‘U2’. If the test is negative then the points is added to the list for ‘U’.

Example

The following code takes the union of two sets of points:

```matlab
Set1=[1 2; 2 1; 1 1]; Set2=[1 3; 2 1];
Set=uni(Set1, Set2)
```
Set =
1 3
2 1
1 2
1 1

Purpose

The purpose of **UNION** is to compute the union of two NPSs.

Usage

\[
\text{function } U = \text{union}_\text{-}(U1, U2);
\]

where the variables used are:

- **U1** - first NPS
- **U2** - second NPS
- **U** - union NPS

**Brief Description of the Algorithm**

The union of the +polytopes of the two NPSs is updated with the -polytopes. See **demo12** for an example of the application of **union**.

Purpose

The purpose of **UPDATE** is to compute the intersection of a half-space and a polytope.

Usage

\[
\text{function } [Pup, acc] = \text{update}(h, P);
\]

where the variables used are:

- **h[1 0]** - half-space inequality
- **P[1 eps;1 0]** - polytope
**Brief Description of the Algorithm**

This is a direct algorithm to obtain the intersection of a half-space and a polytope. If no vertex of ‘P’ falls DELTA-outside of ‘h’ then the intersection of ‘P’ and ‘h’ remains ‘P’. Then the vertices of ‘P’ inside, outside and on the boundary of the half-space defined by ‘h’, are found. New vertices are only computed by ‘vtxfit’ if two vertices are connected by an edge and one is inside, the other is outside ‘h’. The facets, which are fully on or outside the intersection of ‘P’ and ‘h’, are erased. The facet-inequality ‘h’ is added to the list, vertices outside ‘P’ are erased from the vertex list.

**Example**

The following code generates a random polytope and intersects it with a random half-space going through the origin:

```matlab
P=convh(rand(10,3)-0.5);view2d(P,[1 2 3]);figure;
P1=update([rand(1,3) 0],P);view2d(P1,[1 2 3]);
```

**Pup**[[1 eps;1 0]] – updated polytope

**acc[eps]** – maximum error of adjacency of vertices and facets
Purpose

The purpose of VERTICES is to produce the list of vertices of a polytope.

Usage

function [V,nv]=vertices(P);

where the variables used are:
P[[1 eps;1 0]] - polytope
V[0] - array of vertices
nv[1] - number of vertices

Brief Description of the Algorithm

This is a direct extraction of the vertex list from the polytope type 0 matrix representation.

Example

The following code generates a random polygon, extracts its vertices and places a circle at each vertex:

P=convh(rand(15,2));V=vertices(P);
view2d(P);hold on;plot(V(:,1),V(:,2),'o');
**Purpose**

The purpose of **VIEW2D** is to display 2D wire-frame projections of a polytope on pairs of axes.

**Usage**

```matlab
function axout=view2d(P,a,c,axin);
```

where the variables used are:

- `P([1 eps; 1 0])` - polytope
- `a([1 2])` - indices of axis of interest
- `c('b')` - colour of wire-frame plot
- `axin(gca)` - vector of axes handles to be used
- `axout` - vector of axes handles produced

**Brief Description of the Algorithm**

The ‘vvf’ routine is used to establish vertex-vertex adjacency for the wire-frame plots. All 1D edges are plotted. The number of projection axes indices in ‘a’ can be 2 (default, 1 axis) 3 (3 axes) and 4 (6 axes). The handle of all the axes are in output ‘axout’.

**Example**

The following code generates a random 4D polytope and displays its wire-frame projected views onto 6 axes pairs:

```matlab
P=convh(rand(7,4));view2d(P,[1 2 3 4]);%no.figs=1
```
VIEW2DEL

Purpose

The purpose of VIEW2DEL is to display 2D views of ellipsoids.

Usage

function [axout,ellip]=view2del(E,a,col,axin,styl,resol,aequal);

where the variables used are:

E[[1 0 0;1 0 0;0 1 0]] - ellipsoid
a[[1 2]] - axis of 2D projections
col['blue'] - color of ellipsoid
axin - object handle of axes
styl['-'] - linestyle
resol[100] - resolution of plot
aequal[0] - axis equal setting
axout - axes handle
ellip - object handle of ellipsoid

Brief Description of the Algorithm

The ‘projell’ routine is used to compute ellipsoid projections onto 2D planes spanned by pairs of axes. The number of projection axes indices in ‘a’ can be 2 (default, 1 axis) 3 (3 axes) and 4 (6 axes). The handle of all the axes are in output ‘axout’.
Example

The following code generates a random 4D ellipsoid and displays its projected views onto 6 axes pairs:

\[
A = \text{rand}(6,4) - 0.5; E = \text{defell}(A' \cdot A);
\]
\[
\text{view2del}(E, [1 2 3 4]); \% \text{no.fig}=1
\]

Figure 4.52: Demo figure to routine VIEW2DEL.
CHAPTER 4. PROGRAMMING REFERENCE

VIEW3D

Purpose

The purpose of VIEW3D is to display 3D view of 3D polytope.

Usage

function [axout,p,ligh]=view3D(P,axin,style,pcolor);

where the variables used are:

P - polytope
axin - axes in which polytope is to be displayed P
style['interp'] - style of rendering
pcolor - basic colour of the polytope
p - handle of patch object of polytope
axout - axes handle
ligh - handle of light used

Brief Description of the Algorithm

The plot of the 3D view is based on creating a patch object of faces of the polytope. If 'axin, style and pcolor' are defined then reasonable defaults are taken. The object handles of the axes(axout), patch(p) and light(ligh) appear as outputs of the routine so that they can be used altered or manipulated by the user. See the MATLAB manual on graphics and patch objects.

Example

The following code generates a random 3D polytope and displays its perspective view:

P=convh(rand(7,3));ax=view3d(P);
P=convh(rand(7,3));view3d(P,ax);%no.figs=1
Figure 4.53: Demo figure to routine VIEW3D.

**Purpose**

The purpose of **VIEW3DEL** is to display 2D views of ellipsoids.

**Usage**

```matlab
function [axout,ellip,ligh]=view3del(E,axin,style,resol);
```

where the variables used are:

- `E[[1 0 0 0; 1 0 0 0;0 1 0 0;0 0 1 0]]` - ellipsoid
- `axin` - object handle of axes
- `style[‘smooth’]` - texture of surface (‘smooth’ or ‘mesh’)
- `resol[100]` - resolution of surface plot
- `axout` - axes handle
- `ellip` - object handle of ellipsoid surface
- `ligh` - light handle

**Brief Description of the Algorithm**

The plot of the 3D view is based on creating a patch object of the surface of the 3D ellipsoid. If ‘axin, style and resol’ are not defined then reasonable defaults are taken. The object handles of the axes (axaout), patch(ellip) and light(ligh) appear as outputs of the routine so that they can be used altered or manipulated by the user. See the MATLAB manual on graphics and patch objects.
Example

The following code generates a random 3D ellipsoid and displays its perspective view:

```matlab
A=rand(6,3)-0.5;E=defell(A'*A);ax=view3del(E);% no.fig=1
A=rand(6,3)-0.5;E=defell(A'*A);view3del(E,ax);
```

Figure 4.54: Demo figure to routine VIEW3DEL.
**VREDUCE**

**Purpose**

The purpose of **VREDUCE** is to eliminate multiple points from array of points.

**Usage**

```matlab
function nV=vreduce(V);
```

where the variables used are:

- `V` - array of points
- `nV` - reduced array of points

**Brief Description of the Algorithm**

This algorithm is based on repeated application of the routine ‘firstequ’ to obtain a reduced set of points by combining the DELTA near ones. Note that the result may depend on the order of the points listed.

**Example**

The following code generates 1000 points in a 4D hypercube of edge size 10*DELTA and reduces the number of points by making the DELTA-near ones equal:

```matlab
DELTA=1e-10;V=10*DELTA*(rand(200,4)-0.5);
nV=vreduce(V);no_of_new_points=size(nV,1)
no_of_new_points =
192```
## VTXFIT

### Purpose

The purpose of **VTXFIT** is to fit a point to d hyperplanes.

### Usage

```
function [v,fitacc]=vtxfit(H);
```

where the variables used are:

- **H** - list of linear equalities
- **v** - the point fitted
- **fitacc** - fitting accuracy of the point

### Brief Description of the Algorithm

The algorithm to compute the intersection of \( d \) hyperplanes is based on SVD and Moore-Penrose semi-inverse computation.

### Example

The following code fits a point to 4 hyperplanes in 4D and displays the maximum fitting error:

```matlab
F=[rand(4,4)-0.5 ones(4,1)];F
F =
    0.06553   0.41738  -0.42366   -0.38905   1
    -0.20238   0.070695  0.25812  -0.41507   1
    0.11965  -0.4602  -0.39331    0.12071   1
    -0.3316    0.10412  -0.26176  -0.19146   1
[v,acc]=vtxfit(F);[v,acc]
v =
    -0.63125   -2.0117  -1.3975   -3.3131
acc =
    1.7764e-015
```
Purpose

The purpose of VVF is to produce a vertex-vertex and vertex-facet adjacency tables (version no. 1).

Usage

\[
\text{function } [\text{VV, VF, nv, nh}] = \text{vvf}(H);
\]

where the variables used are:

- \( H \) - polytope
- \( \text{VV} \) - vertex-vertex adjacency table
- \( \text{VF} \) - vertex-facet adjacency table
- \( \text{nv} \) - number of vertices
- \( \text{nh} \) - number of facets

Brief Description of the Algorithm

All facet-inequalities are tested against all vertices to see whether they fit within a DELTA error bound. To obtain ‘VV’, for each vertex the indices of all those vertices are listed which share \( d - 1 \) adjacent facets. To obtain ‘VF’ first the indices of all adjacent facets are listed for each vertex. The maximum length of lists is monitored and this determines the number of columns in ‘VV’ or in ‘VF’. If a list for a vertex is shorter than the number of columns then the rest of the entries is filled with zeros.

Example

The following code generates a random 3D polytope and displays its vertex-vertex and vertex-facet adjacency tables:

```matlab
P = convh(rand(5,3)); view2d(P,[1 2 3]);
[VV, VF] = vvf(P)
VV =
  2 3 4
  1 3 4
  1 2 4
  1 2 3
VF =
  2 3 4
  1 3 4
  1 2 4
  1 2 3
```
This m-file gives error message about the violation of the DELTA error bound:

```matlab
disp(' '); 
disp('DELTA” has been violated! Use SETDELTA .'); 
DELTA
disp(' '); 
```

This m-file gives error message about the violation of the KAPPA vertex bound:

```matlab
disp('');
disp('Warning: polytope has been clipped by hypercube H_d(KAPPA)');
disp(' Use SETDELTA .');
KAPPA
disp('');
```

WARNKAP WARNKAP
4.1 Licence Terms and Conditions - GBT 7.2

SUPPORT:
Staff at SysBrain Ltd provide email support up to 4 hours of work to help you apply GBT to your application (gbt@sysbrain.com).

UPGRADES:
You are entitled to free upgrades of the software within one year of its purchase. New versions will be emailed/posted to you automatically.

GRANTING OF LICENCE:
Subject to payment of the applicable licence fees, SysBrain Ltd grants to you, i.e. the Licensee, a non-exclusive licence to use the Geometric Bounding Toolbox (GBT) Version 7.2 under the following terms. GBT is not sold but licensed to you.

LICENCE TERMS:
You may:
A. use GBT on any single computer;
B. use GBT on a second computer so long as only one copy is used at a time;
You may not:
A. permit other individuals to use GBT except under the terms listed above;
B. rent, lease, grant, or otherwise transfer rights to GBT;
C. remove any proprietary notices in the code of GBT or in its manual.

TITLE:
GBT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The algorithms of GBT are licensed and not sold. This Licence may not be assigned, sublicensed or otherwise transferred by the Licensee, by operation of law or otherwise, without prior written consent from SysBrain Ltd.

ADAPTATION IN COMMERCIAL SOFTWARE:
If any of the algorithms of GBT is compiled into another programming language, such as C, and is used as part of a commercially distributed executable programme or source code, then reasonable amount of royalty is to be payed to SysBrain Ltd. In case of a planned software for commercial distribution, which uses some algorithms of GBT, the Licensee will contact SysBrain Ltd to negotiate a fair amount of royalty, prior to distribution.

TERMINATION:
Without prejudice to any other rights, SysBrain Ltd may terminate this agreement if you fail to comply with the terms and conditions of this agreement.

SysBrain Ltd, Registered in England, Number 4583971, Registered Office: 10 Rothschild Close, Southampton, SO19 9TE, UK
DISCLAIMER OF WARRANTY:

GBT and the accompanying files are sold "as is" and without warranties as to performance of merchantability or any other warranties whether expressed or implied. Because of the various hardware and software environments into which the algorithms of GBT may be put, no warranty of fitness for a particular purpose is offered. Good data processing procedure dictates that any program be thoroughly tested with non-critical data before relying on it. The user must assume the entire risk of using the program. Any liability of the seller will be limited exclusively to product replacement or refund of purchase price.

LIMITATION OF LIABILITY:

Under no circumstances and under no legal theory, tort, contract, or otherwise, shall SysBrain Ltd or its suppliers or resellers be liable to you or any other person for any indirect, special, incidental, or consequential damages of any character including, without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses. In no event will SysBrain Ltd be liable for any damages in excess of the amount SysBrain Ltd received from you for a licence for GBT, even if SysBrain Ltd shall have been informed of the possibility of such damages, or for any claim by any other party. This limitation of liability shall not apply to liability for death or personal injury to the extent applicable law prohibits such limitation. Furthermore, some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, so this limitation and exclusion may not apply to you.

CUSTOMER REGISTRATION:

The registration details will not be passed to any third party.

_____________________________________
Signed and dated by the Licensee
Bibliography


