Geometric Bounding Toolbox
Version 7.0

User's Manual & Reference

Foreword

The routines in *GBT 7.0* permit reliable and fast manipulation of high-dimensional polytopes. GBT 7.0 is a unique computational package of its kind because of its *numerical error control.* Compared with the previous version, GBT 7.0 is a complete rewrite - based on experience accumulated since 1993 when GBT 1.0 was launched.

When compared with GBT 6.1, the main differences are:

- GBT 7.0 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 7.0, 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 in GBT 7.0) 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. At the heart of the package is a convex hull algorithm, similar to the *quickhull* [1, 2] algorithm but with special care taken to ensure numerical reliability and efficiency.

Chapter 1 defines the standard format of convex and non-convex polytopes. Chapter 2 explains the handling and monitoring of numerical accuracy within GBT 7.0. Chapter 3 is a reference manual for all routines contained in GBT 7.0.

Associated with this "User's and Reference Manual" there are two further manuals: the "Applications Manual" for the users of the package in various fields and the "Programming Manual" that gives detailed comments on every part of the code and is is intended for the advanced user who wishes to modify the package or to compile it in C.

Queries on GBT 7.0 can be addressed to *gbt@sysbrain.com*.

Sandor M Veres & David Q. Mayne
Southampton, September 2000

Contents

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 [3, 4, 5, 6, 7].

Notation

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

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

The *distance* of two points $a \in \Re^n$ and $b \in \Re^n$ will be measured in the 2-norm as $||a - b||_2$, i.e. $[\sum_1^n (a_i - b_i)^2]^{1/2}$.

The rows of a matrix A will be denoted by $A^1, \ldots A^n$ and its column by $A_1, \ldots, A_n$. A subset of row and column vectors from indices i to j will be denoted by $A^{i:j}$ and $A_{i:j}$, respectively.

$Proj_{orth}(x|S)$ denotes the orthogonal projection of $x \in \Re^n$ onto an affine subspace S in $\Re^n$.

$dist(x, S)$ denotes the $||.||_2$-distance of a point $x \in \Re^n$ 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 $a \in \Re^n$, $||a|| \neq 0$ and any $b \in \Re$ the geometric space*

$$H \overset{def}{=} \{ \ x \in \Re^n \ \mid a^T x \leq b \ \} \tag{1.1}$$

is called a half-space .

Intersections of a finite number of half-spaces are *polyhedra* and *polytopes* are those polyhedra which are finite.

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 $\Re^n$ 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. [8]* $\square$

The most important faces of a polytope are its *vertices* and *facets* [8]. 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 $\Re^n$ means to determine all the facets of the convex hull polytope.

Problem 1 *Given a set of points $v_1, v_2, \ldots, v_N \in \Re^n$, compute the facets of their convex hull.*

Problem 1 is also called the *vertex enumeration problem* [2, 1]. 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 $a_1^T x \leq b_1,\ a_2^T x \leq b_2, \ldots,\ a_N^T x \leq b_N, ||a_i|| \neq 0,\ a_i \in \Re^n$, 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 $\Re^n$. The* sum *of polytopes P_1 and P_2 will be defined by*

$$P_1 \oplus P_2 \overset{def}{=} \{\ x + y \mid x \in\ P_1,\ y \in\ P_2\ \} \tag{1.2}$$

The difference *of polytopes P_1 and P_2 will be defined by*

$$P_1 \ominus P_2 \overset{def}{=} \{\ x \in \Re_n \mid \forall y \in P_1 : x + y \in P_2\ \} \tag{1.3}$$

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*.

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 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, 10, 11, 2, 1, 12] 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, 14] 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, 10, 11] which did not offer consistent numerical solution. GBT 7.0 is the first version of the toolbox which implements logically consistent numerical algorithms.

1.2 List of Routines

The following list is the content of the toolbox as on 20.12.2000 . 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
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
surfarea	- computes the surface area of a polytope

Polytope operations

intersct	- computes the intersection of two polytopes
ltran	- performs affine transformation on a polytope
poladd	- computes 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
fvfitacc	- extracts maximum fitting error of vertices and facets
planeigh	- lists all neighbouring facet planes to a facet

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
projpont	- projects a point onto polytope
update	- computes the intersection of a half-space and a polytope

Polytope graphics

view2d	- displays 2D wire-frame projections of polytope on 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

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
elldifi	- computes an inner bounding ellipsoid to the difference of two ellipsoids
ellint	- computes an ellipsoid around the intersection of two ellipoids
projell	- computes the projection of an ellipsoid onto a linear subspace
sphsep	- computes the joining surface area of spheres

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
ellicov	- 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

Ellipsoid graphics

view2del	- displays 2D projection on axes pairs
view3del	- displays 3D view of an ellipsoid

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

Routines for compatibility with GBT 6.1

bounded	- tests whether a polyhedron is bounded
gbt627	- converts from GBT 6.1 formats to GBT 7.0 format
gbt726	- converts from GBT 7.0 format to type 4 format of GBT 6.1

1.3 Standard Object Representations

The toolbox uses a polytope representation that makes the best compromise between detailed description and simplicity in computations involving geometric objects. Chapter 2 gives a detailed analysis of the problem and explains why this type of representation was chosen for GBT. 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 respresentations. *type* 0 stores the least amount of data from all representations.

1.3.1 Convex polyhedra

The *standard form* of a convex 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:

$$P = \begin{bmatrix} n_e & n_i & \mu & 0 & \cdots & 0 \\ \text{1st} & \text{affine} & & & \text{subspace} & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_e\text{th} & \text{affine} & & & \text{subspace} & \\ \text{1st} & \text{facet} & & & \text{inequality} & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_i\text{th} & \text{facet} & & & \text{inequality} & 0 \\ \text{1st} & \text{vertex} & & & \text{vector} & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_v\text{th} & \text{vertex} & & & \text{vector} & -1 \end{bmatrix} \tag{1.4}$$

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 therfore 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 1.3.1 *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 δ.*

An example

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

```
>> 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:

```
>>P(1,3)

ans =

  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.
- 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:

```
>> 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.

1.3.2 Ellipsoid representation

Let P be a positive definite $d \times d$ symmetric matrix and $c \in \Re^d$. Then the set

$$E = \{ \ x \mid (x-c)^T P^{-1} (x-c) \leq 1 \ \} \tag{1.5}$$

is called a $d-$*dimensional ellipsoid*. For singular P a degenerate ellipsoid is defined by

$$E_{deg} = \{ \ x \mid (x-c)^T P^{-} (x-c) \leq 1 \ \} \tag{1.6}$$

where P^- denotes the Moore-Penrose semi-inverse of P.

The matrix representation of an ellipsoid in MATLAB is

$$P^{rep} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ P_{11} & \cdots & P_{1d} & c_1 \\ P_{21} & \cdots & P_{2d} & c_2 \\ \vdots & & \vdots & \vdots \\ P_{d1} & \cdots & P_{dd} & c_d \end{bmatrix} \tag{1.7}$$

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

Chapter 2

Numerical Error Monitoring

2.1 Computational problems

During the computation of polyhedral regions of parameters, states or inputs, the vertices and faces will inevitably be described by finite numerical precision. To handle the inaccuracies is important as in some cases small numerical errors can propagate to cause large errors later and the geometric regions will be computed inaccurately. The problems are non-negligeable and can cause serious malfunction, especially in on-line applications where there is no time or engineer available to investigate the cause of problems.

The numerically and logically consistent algorithms will here be based on the assumption of floating point arithmetic and in case of fixed-point arithmetic the algorithms would have to be modified to make them bug free in applications.

A binary floating point number is represented in the form

$$f = m \times 2^n, \quad 1 \leq m < 2, \quad -L \leq n \leq L \tag{2.1}$$

where m is the *mantissa* of fixed length l and L is the maximal allowed exponent.

In standard MATLAB computations the floating point relative accuracy ϵ (called **eps** in MATLAB) is the distance from 1.0 to the nearest floating point number less than 1. ϵ (written for **eps**)is used as a default tolerance by PINV (pseudo-inverse) and RANK (rank of a matrix), as well as several other MATLAB functions. **realmax** and **realmin** are the largest and smallest positive floating point numbers representable on a given computer, respectively. Using single precision in MATLAB on an IBM compatible PC, these values are

```
eps=2.2204e-016, realmax= 1.7977e+308, realmin=2.2251e-308
```

The implications of this are that any irrational real number x can only be represented with some relative accuracy:

$$x = x_0 \times (1 + \epsilon) \tag{2.2}$$

where x_0 is the precise desired (theoretical) value, ϵ is the relative error such that $|\epsilon| < \epsilon$. In addition to this $|x| \leq \; realmax$ must be satisfied, anything bigger is handled as inf (to be more precise $(1 + 10^{12} * eps) * 1.7976e + 308 \; = \; inf$ but

$(1 + 10^{11} * eps) * 1.7976e + 308$ is still represented as a number). Thus inf is a symbol to denote excessively large numbers in MATLAB.

Relative accuracy ϵ represents however only the largest possible accuracy of sinle number and not the actual accuracy after some computations. For instance assume that an inequality

$$ax \leq b \tag{2.3}$$

is given to bound a semi-infinite interval in dimension 1 (1D), i.e. on the real axis. Assume that $a > 0$ and b are given with relative accuracy ϵ so that they are represented by $\tilde{a}$, $\tilde{b}$ numerically. The end point of the interval is in theory

$$v = b/a \tag{2.4}$$

if a is non-zero, otherwise inequality (2.3) does not define a semi-infinite interval. The actually computed v will rather be

$$\tilde{v} = \tilde{b}/\tilde{a} = \frac{b + \delta_b}{a + \delta_a}, \tag{2.5}$$

where $\delta_a \leq |a|\epsilon$ and $\delta_b \leq |b|\epsilon$. The relative accuracy of v will then be

$$\frac{\tilde{v} - v}{v} = \frac{(b + \delta_b)/(a + \delta_a) - v}{v} = \frac{a\delta_b - b\delta_a}{(a + \delta_a)b} = \frac{(a + \delta_a)\delta_b - (b + \delta_b)\delta_a}{(a + \delta_a)b} =$$

$$= \frac{\delta_b}{b} - \left(1 + \frac{\delta_b}{b}\right) \frac{\delta_a}{a + \delta_a} \leq \epsilon + (1 + \epsilon)\epsilon = 2\epsilon + \epsilon^2,$$

where the last inequality can be attained as equality by suitably chosen errors. This means that the relative worst-case accuracy of $\tilde{v}$ representing v is roughly proportional to $2\epsilon + \epsilon^2$. If the endpoint $\tilde{v}$ of this interval is used in further numerically sensitive computations, then the initial numerical error can be further amplified to become a major source of error.

This example at least demonstrates that if linear inequalities are given with maximum numerical accuracy to define a polyhedron, then its vertices may not be possible to compute with the same maximum accuracy. The example provided is the simplest and it is in 1D. In higher dimensions the problem becomes more acute because of the matrix inversions needed to compute vertex and hyperplane locations. If no sufficient care is taken then after long computations the adjacency tables (Boolean tables indicating which vertex fits to which facet), can become incorrect as a result of numerical errors and can lead to logical inconsistencies of the definition of polytope adjacency tables. The aim of this section is to introduce logically consistent algorithms for computation with polytopes.

2.2 Analysis of possible polytope representations

In this section some alternatives of polytope representation are looked at which are different from the one used in GBT 7.0 . The aim is to explain why the representation defined in Chapter 1.3 was selected for use in GBT 7.0 . The analysis will conclude that the representation used by GBT 7.0 makes the best compromise between informative storage of polytope features and speed of operations on polytopes.

*

One definition of a polyhedron is that it is the intersection of a finite number of half-spaces. For polytopes, i.e. bounded polyhedra, another possible definition is that a polytope is the convex hull of a finite set of points. Both of these definitions offer a way of representing a polytope numerically. We will now assess what advantages/disadvantages these representations would bring when large number of operations are to be carried out on these geometric objects.

1. **Polytope representation by a list of inequalities only.** This representation is called *efficient* if the inequalities listed are all *active*, i.e. none of the inequalities is implied by some others in the list.

   Affine transformations. These are straightforward computations if the transformation is of full rank. For singular tranformation the procedure of finding inactive redundant inequalities may lead to a large number of LP problems.

   Intersection of two polytopes. Straightforward computation, large number of LP problems may have to be used to check for inactive inequalities.

   Computing the vertices of the polytope requires the computation of the *convex hull* represented by vertices.

   Union of two polytopes. Computationally heavy as convex hull computation is involved.

   Minkowski sum and difference of two polytopes in the form of a set of inequalities are heavy computational procedures.

   Updating of polytope by a new half-space. Relatively straightforward. Finding the inactivated facets can amount to a lengthy LP problem.

2. **Polytope representation by a list of vertices only.** The representation is called *efficient* if the vertices listed are all *active*, i.e. none of the vertices is contained in the convex hull of some others in the list.

   Affine transformations. These are straightforward computations if the transformation is of full rank. For singular tranformations finding redundancy of some vertices may lead to a large number of LP problems.

   Intersection of two polytopes. Computationally heavy.

   Computing the facets of the polytope requires the computation of the *convex hull* represented by faces.

   Union of two polytopes. Computationally straightforward, large number of LP problems may have to be used to check redundancy of inequalities.

   Minkowski sum and difference of two polytopes in the form of a set of inequalities are heavy computational procedures requiring convex hull computations.

   Updating of a polytope by a new half-space. Finding the new vertices can lead to a complex logical analysis of facet/vertex adjacency.

3. **Polytope representation by a list of vertices and hyperplanes of facets and accuracy parameter δ.** A representation is called *consistent* if the vertex-facet adjacency table defined by δ corresponds to the adjacency table of a polytope.

Affine transformations. These are straightforward computations if the transformation is of full rank. For singular transformations the algorithm of *convex hull operation* can be used.

Intersection of two polytopes. Facet inequalities can be combined and the convex hull operation can be applied in the dual space.

Computing the vertices to a set of inequalities would require to compute the *convex hull* using the convex hull operation.

Union of two polytopes. Computationally straightforward, the union of the set of vertices can be formed and convex hull operation applied.

Minkowski sum and difference The sum is computationally straightforward, the Minkowski sum of the two sets of vertices can be formed and convex hull operation applied. Similarly for the difference of polytopes A and B: using the vertices of B, a set of inequalities can be can be formed for the difference $A \ominus B$. First converting to the dual-space, convex hull operation can be used to computer the resulting polytope.

Updating of polytope by a new half-space. By first converting to the dual-space, convex hull operation can be applied to get the new polytope.

4. **Polytope representation by a list of vertices and hyperplanes of facets and adjacency lists of vertices and facets.**

   There are two versions of this approach:

   (i) One version handles only *simple polyhedra*, i.e. ones which have only d facets adjacent at each vertex, where d is dimension of the space. This approach can be made to work in practice (as in earlier versions of GBT [13]) by "micro-randomization" of hyperplane directions, which would produce a small cloud of very near vertices in the rare case a polytope is required which has more than d facets at a vertex. The numerical accuracy can be controlled by suitably small micro-randomization.

   (ii) Another version handles any kind of polyhedron in which any large number of facets can meet at a vertex. In this version full adjacency tables are updated. The complete testing of this algorithm is complicated and bug free operation is not guaranteed until further definitions are introduced for what is meant by numerical adjacency of a vertex and a facet [15].

In principle each of the first two and the fourth approaches outlined above could be made working. Unfortunately none of these have received sufficient logical analysis (in numerical terms) in the published literature so that they could work reliably in all situations in on-line applications. The purpose of the following sections will be to give a completely algorithmic description on the basis of representation (3).

2.3 Logically consistent computations

To follow the approach outlined under (3) in the previous subsection, the following representation of a polyhedron will be agreed upon.

Definition 2.3.1 *In the d-dimensional space a convex* numerical polyhedron P *will be represented as a quadruple* $[E, F, V, \delta]$ *where* E *is a set of* $(d-1)$*-dimensional affin* *subspaces which all contain P, F is a set of facet inequalities, V is a set of vertices and δ is a tolerance constant which helps to define the accuracy of facet-vertex adjacency. Furthermore, in this representation it is assumed that the elements of E and F are all defined by using unit normal vectors.*

The *standard form* of a convex polyhedron in the d-dimensional Euclidean space will be defined by a matrix

$$P = \begin{bmatrix} n_e & n_i & \mu & 0 & \cdots & 0 \\ \text{1st} & \text{affine} & & & \text{subspace} & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_e\text{th} & \text{affine} & & & \text{subspace} & \\ \text{1st} & \text{facet} & & & \text{inequality} & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_i\text{th} & \text{facet} & & & \text{inequality} & 0 \\ \text{1st} & \text{vertex} & & & \text{vector} & -1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_v\text{th} & \text{vertex} & & & \text{vector} & -1 \end{bmatrix} \tag{2.6}$$

where the number of vertices n_v is the remaining number of rows in the matrix so that $n_v = n - 1 - n_e - n_i$ if n is the number of rows in P.

Definition 2.3.2 *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 of its facet-vertex fitting precisions are less than δ.*

Let V_P and F_P denote the lists of vertices and facets of a polytope P. Precision of vertex-facet adjacency is then described by the function $\Delta(f, v)$, $f \in F_P$, $v \in V_P$. An adjacency tolerance δ for a polytope P is called **tight** if the Boolean function $B(f, v) \overset{def}{=} \Delta(f, v) < \delta$ is identical to the adjacency relationships of a polytope.

Example. Consider the nearly rectangular polygon with vertices

$$V = \{[0.01\ 0.001],\ [0\ 0],\ [0\ 1],\ [0.01\ 1]\}$$

and set of facets

$$F = \{[1\ 0\ 0.01], [-1\ 0\ 0], [0\ 1\ 1], [0\ -1\ 0]\}$$

Then the adjacency matrix will be

$$\Delta = \begin{pmatrix} 0.001 & 0.01 & 0.999 & 0.001 \\ 0.01 & 0 & 1 & 0 \\ 0.01 & 0 & 0 & 1 \\ 0 & 0.01 & 0 & 1 \end{pmatrix}$$

For $\delta = 0.002$ this gives the correct adjacency Boolean table

$$A_P = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

For a too small $\delta = 0.0001$ the Boolean table becomes

$$A_P = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

which does not correspond to the adjacency table of any polygon.

For a large $\delta = 0.02$ the Boolean table becomes

$$A_P = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

which again cannot represent the Boolean table of any polygon. This example shows therefore that in computations the adjacency of vertices and faces is not an absolute concept, it depends on the accuracy tolerance specified.□

Definition 2.3.3 *A numerical polytope $P = [E, F, V, \delta]$ is called* consistent *if its facet-vertex adjacency table defined by δ is topologically equivalent to the adjacency table of a polytope.*

A basic requirement on any of the numerical binary or unitary operations performed on polytopes is now that if their inputs are consistent numerical polytopes than their outputs must also be consistent numerical polytopes. The adjacency tolerance of the output polytope does not have to be equal to the adjacency tolerance of the input polyhedra.

Definition 2.3.4 *The algorithm of a polyhedron operation is called* numerically consistent *if its output is a consistent numerical polytope whenever its inputs are consistent numerical polytopes.*

It was indicated in the previous subsection under polytope representation (3) that the algorithm of convex hull operation (CONVH) is a basic algorithm which can serve as the crucial part of most other operations. To be more precise, this algorithms computes the set of facets to a given set of points so that the resulting numerical polytope is consistent.

2.4 The convex hull operation (CONVH)

The problem is to compute the facets of the convex-hull for a given set of points. The idea of the solution is to start from a simplex of $d + 1$ points and to include all the rest of the elements of the set into the convex hull one-by-one. In this convex-hull algorithm the inaccuracy of the computation of a facet can be caused by ill-conditioning of the linear system of equations which determine the normal of the facet. For instance in 3D, if three points are given which lie nearly along a straight line, then computing the normal of the plane which fits to the three points can be done only with much reduced accuracy. The inaccuracy of the calculation of a facet can lead to an incorrect decision whether a point is inside or outside the convex hull. This may then result that including the points in different order will result in a different solution, i.e. different list of facets and vertices. Also, the accuracy of the resulting convex hull description can vary. This indicates that a mechanism for numerical error control is necessary.

The input of this algorithm is a set of points in the d-dimensional space so that each of the points is assumed to have ϵ relative accuracy in each of its co-ordinates. The output is a numerical polytope in the $[E, F, V, \delta]$-form. The main result with regard to this algorithm is that it always results in a *consistent* numerical polytope. The user of the routine can define a desired initial tolerance δ but the routine may increase this if it is not possible to ensure the desired tolerance.

The next algorithm is a subroutine of CONVH and is called P_CONV. Its purpose is to include a given single point x into the convex hull of a set of points represented by a polytope P. It also starts with a given desired tolerance δ which it may increase if it turns out to be impossible to achieve the desired adjacency tolerance. P_CONV always produces a consistent numerical polytope of the standard form $[E, F, V, \delta]$.

Linear transformations of polytopes of the standard numerical form can be performed by the LTRAN algorithm. The linear transformation is given by a matrix A so that the linear mapping is $x \rightarrow Ax + a$ where a is an offset vector. The algorithm is able to transform and embed a lower dimensional polytope into a higher dimensional space or map a higher dimensional polytope into a lower dimensional one, depending on the dimensions of A. If the input of LTRAN is a consistent numerical polytope then its output is also a consistent numerical polytope. LTRAN calls CONVH to perform the intensive computations.

Given a polytope $P \subset \Re^d$, its dual is defined by

$$P_{dual} = \{x \mid x \in \Re^d,\ xy \leq 1, y \in P\} \tag{2.7}$$

If P contains the origin (zero vector) as one of its internal points, then the dual of a polytope is also a polytope. In this case the vertices of P can be brought into one-to-one correspondence with the facets of P_{dual}. Vica-versa, the vertices of P_{dual} can be brought into one-to-one correspondence with the facets of P. This fact can be used to convert the convex-hull computation for a set of vertices into the convex-hull computation for a given set of hyperplanes in order to obtain the set of vertices. The algorithm FCONVH is doing just that: given a set of inequalities, it produces the set of vertices for the polyhedron defined by the inequalities, using CONVH. A desired tolerance can be given at its input and it always produces a consistent numerical polyhedron of the standard form.

Given a polytope P and a half-space H defined by an inequality of the form $x^T a \leq c$, computing the intersection of the polytope P and the half-space H is called *polytope updating*. The algorithm UPDATE performs polytope updating by a single half-space. FCONVH computes the intersection of a polytope with a given finite set of half-spaces.

Chapter 3

Reference Manual

3.1 Principle Routines

CONVH **CONVH**

Purpose

CONVH computes the convex hull of a set of points in m-dimensional space. Let S be a finite set of points in $\Re^m$, $m \geq 1$. The convex hull of the points in S is defined by

$$convh(S) \overset{def}{=} \left\{ y \in \Re^m \mid \begin{array}{r} \exists x_i \in S, i = 1, \ldots, N \ : \ y = \sum_{i=1}^{N} \lambda_i x_i, \ with\ some \\ \lambda_i \geq 0, i = 1, \ldots, N; \sum_{i=1}^{N} \lambda_i \ = \ 1 \end{array} \right\}$$

The convex hull is to be computed to a given accuracy $\delta > 0$ and points δ-near to the convex hull are allowed to be neglected. If δ is set relatively high this will reduce computational complexity.

Algorithm

The algorithm is based on straightforward inclusion of the points of the set one-by-one into the convex hull. Initialization is based on first computing the dimension of the given set of points (as they may lie in a lower dimensional linear manifold) and computing a possible large simplex with vertices from the given set of points (which can be lower dimensional than m).

After initialization the remaining points of the given set S are included one-by-one using the P_CONV routine. This routine calls various routines and among them CONVH itself. This way the CONVH can call itself both directly as well as through P_CONV, but this always happens for a lower dimensional problem and therefore the depths of self-calls is finite.

Usage

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

where

```
V      -  a matrix of row-vectors of the set of points
H      -  Polytope in GBT 7.0 -format
acc    -  fitting accuracy of hyperplanes to vertices
```

Example

The following code defines 7 random points in 4D and computes their convex hull:

```
S=rand(7,4); P=convh(S); view2d(P,[1 2 3 4]);
```

Then the wire-frame view of the convex hull is displayed from 6 directions in Figure 3.1.

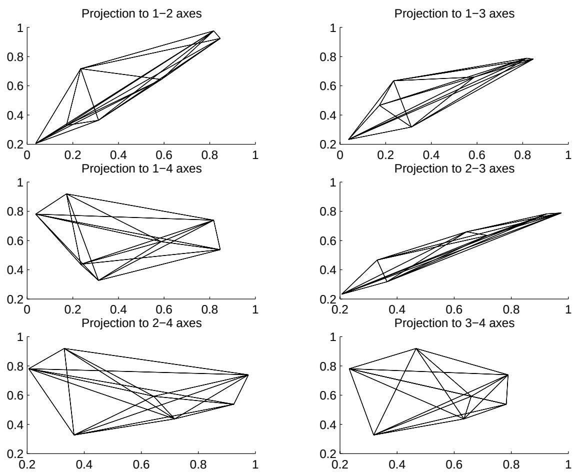

Figure 3.1: The convex hull of 7 random points in 4D.

DEFBOX DEFBOX

Purpose

DEFBOX defines an axis aligned box in GBT 7.0 format. To define the box its lowest (in terms of its co-ordinates) and highest corner point has to be given as input to DEFBOX.

Algorithm

Straightforward definition of the faces from the given corner points of the box. The plane-vertex fitting precision is defined by default as $\epsilon = 2.2204 \times 10^{-16}$.

Usage

```
P=defbox(Lcorner,Hcorner)
```

where

```
Lcorner     -  the lowest corner of the box
Hcorner     -  the highest corner of the box
```

Example

The following code defines a 5D box with low corner at [-1,2,1,3,-2] and its high corner at [1 3 2 4 0]:

```
Box=defbox([-1,2,1,3,-2],[1 3 2 4 0]);view2d(Box,[1 2 3 4]);
```

The wire-frame view of the resulting box is shown Figure 3.2.

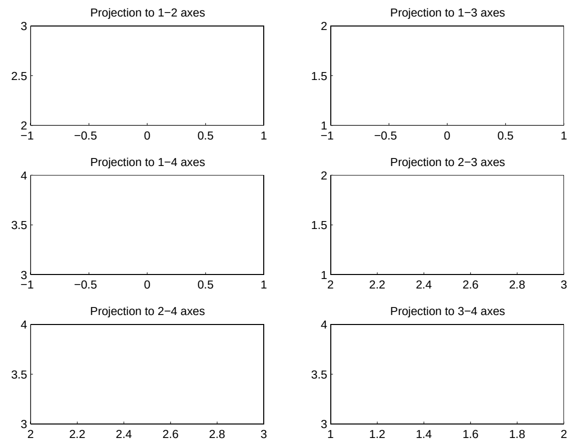

Figure 3.2: View of the wire-frame of a 5D box from 6 directions. The edges of the box are actually on the figure frames/axes.

DEFELL DEFELL

Purpose

DEFELL defines an ellipsoid with given "covariance" matrix and centre in GBT 7.0 format.

Algorithm

Straightforward definition of covariance matrix and centre. DEFELL calls no other routines from GBT 7.0. With P covariance matrix and centre c the ellipsoid defined is

$$E = \{x \mid (x-c)^T P^{-1}(x-c) \ \leq \ 1 \}$$

P has to be positive definite matrix.

Usage

```
P=defell(Cov,cent)
```

where

```
Cov   -  pos. def. covariance matrix of the ellipsoid
cent  -  centre of the ellipsoid
```

DEFPIPE DEFPIPE

Purpose

DEFPIPE defines a parallelepiped in GBT 7.0 format from a given double inequality $b_1 \leq Ax \leq b_2$.

Algorithm

Straightforward application of UPDATE is to the set of inequalities obtainable from the double-inequality. The plane-vertex fitting precision is defined by default as $\epsilon = 2.2204 \times 10^{-16}$.

Usage

```
P = defpipe(b1,A,b2);
```

where

```
b1  -  a column vector (n x 1)
A   -  a matrix (n x n)
b2  -  a column vector (n x 1)
```

Example

The following code defines a 4D parallelepiped:

```
A=[ 2 1 0 0;1 2 1 0;0 1 2 1;0 0 1 2];

P=defpipe([-1,2,1,3]',A,[1 3 2 4]');view2d(Box,[1 2 3 4]);

view2d(P,[1 2 3 4]);
```

Then the wire-frame view of the resulting 4D parallelepiped is shown Figure 3.3.

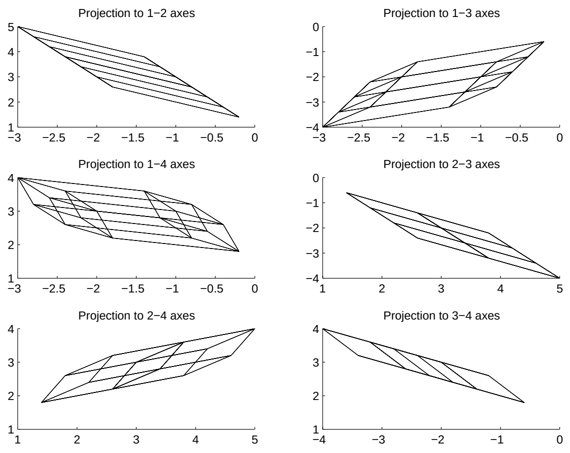

Figure 3.3: View of the wire-frame of a 4D parallelepiped defined by $b_1 \leq Ax \leq b_2$.

DEFSIMP DEFSIMP

Purpose

DEFSIMP defines an regular, i.e. equilateral simplex with unit length of its edges and centred to the origin.

Algorithm

A direct algorithm which builds up the simplex in a recursive procedure. The plane-vertex fitting precision achieved is $\epsilon = 2.2204 \times 10^{-16}$.

Usage

```
        S = defsimp(d);
```

where

```
d   -  the dimension of the simplex required
S   -  name of the regular simplex to be computed
```

Example

The following code defines a 7D simplex:

```
P=defsimp(7); view2d(P,[1 2 3 4]);
```

The wire-frame view of the resulting 7D simplex is shown Figure 3.4 from 6 directions.

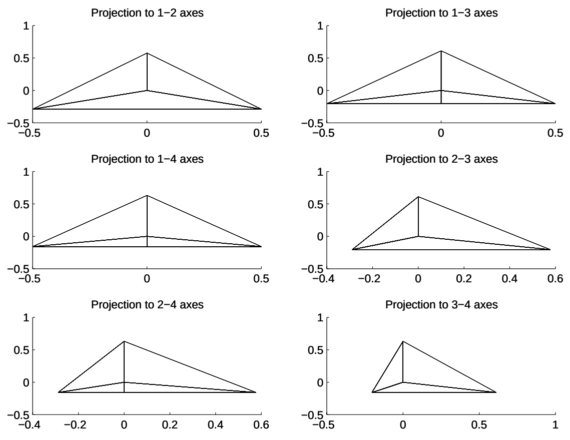

Figure 3.4: Wire-frame views of a 7D regular simplex.

DIAMETER DIAMETER

Purpose

DIAMETER computes the end points of a diameter of a polytope and its lengths.

Algorithm

A direct algorithm based on finding the largest distance between vertices.

Usage

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

where

```
  P    -  the polytope of which the diameter is required
 diam  -  the length of the diameter in Euclidean norm
  ds   -  row-vector of the starting point of the diameter
  de   -  row-vector of the ending point of the diameter
```

Example

The following code defines a 5D random polytope and computes its diameter:

```
V=rand(12,5);P=convh(V); [dia,ds,de]=diameter(P);
```

```
view2d(P,[1 2]); l1=line([ds(1);de(1)],[ds(2);de(2)]);

set(l1,'LineWidth',2);set(l1,'LineStyle',':');
```

Then the wire-frame view of the polytope is shown projected onto the 1-2 plane in Figure 3.5.

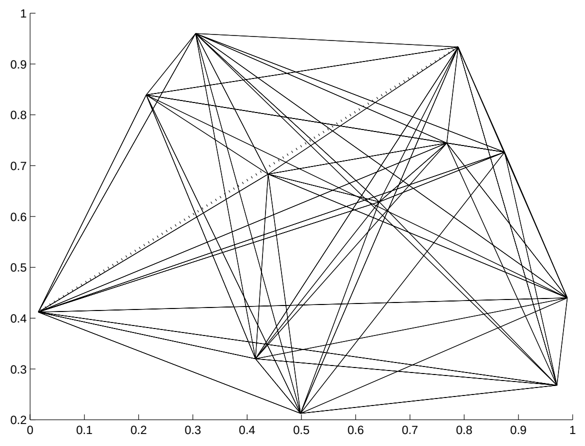

Figure 3.5: Wire-frame view of a 5D polytope and its diameter indicated by the dotted line.

DIF DIF

Purpose

DIF computes the set-difference of two sets of points in the d-dimensional Euclidean space with a given tolerance δ for the discrimination of points by ℓ_1-distance.

Algorithm

For $A \ominus B$ DIF checks point-by-point in B whether it is equal to a point in set A with ℓ_1 accuracy $< \delta$.

Usage

```
U=dif(U1,U2);
```

where

```
 U1,U2   -  matrices of row vectors
  U      -  the set difference
```

Example

The following code defines two sets of points with 2 points shared and checks the number of points of the difference set:

```
A=rand(4,4);B=[A(2:3,:); rand(2,4)]; C=dif(A,B); m=size(C,1)

m =
     2
```

DIM DIM

Purpose

DIM returns the true dimension of a polytope. As the dimension of the space of the polytope representation can be greater that the true dimension of a polytope, this routine returns the dimension of the smallest linear manifold containing the polytope.

Algorithm

A direct extraction from the polytope representation. A single point is defined as zero dimensional (0D), a linear interval is 1D, a square is 2D, etc.

Usage

```
            [dactual,d,mu] = dim(P)
```

where

```
  dactual - actual dimension with fitting precision delta
     d    - formal dimension of the polytope description (dactual<=d)
     mu   - worst-case accuracy of facet-vertex fits
```

Example

The following code defines a 4D random polytope and computes its diameter:

```
V=rand(4,4);P=convh(V); [d,dactu,delt]=dim(P)
```

In a run of this code the following result was obtained:

```
d =
         4

dactu =
         3

delt =
        4.5797e-016
```

DIREXT DIREXT

Purpose

DIREXT returns the extremum points of a polytope along a given direction both ways. Direction is defined by a straight line along which the two extrema of the polytope are to be found. The width of the polytope in the given direction is also computed.

Algorithm

All vertices of the polytope are projected onto a straight line parallel with the given direction and the extrema are found along the line. The width is the distance between the maximum and minimum point on this line.

Usage

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

where

```
G      -   polytope
di     -   direction vector to search for extremal point
vmax   -   extremal point in positive direction +di
vmin   -   extremal point in negative direction -di
width  -   the width of the object in direction  di .
```

Example

The following code defines a 3D random polytope and computes its extrema and width in direction [1 1 1]':

```
V=rand(8,3);P=convh(V);di=[1 1 1]';

[vmax,vmin,width]=dirext(P,di); view2d(P,[1 2]);hold on;

plot(vmax(1),vmax(2),'x'); plot(vmin(1),vmin(2),'x');
```

A wire-frame view of the 3D polytope P is displayed in Figure 3.6.

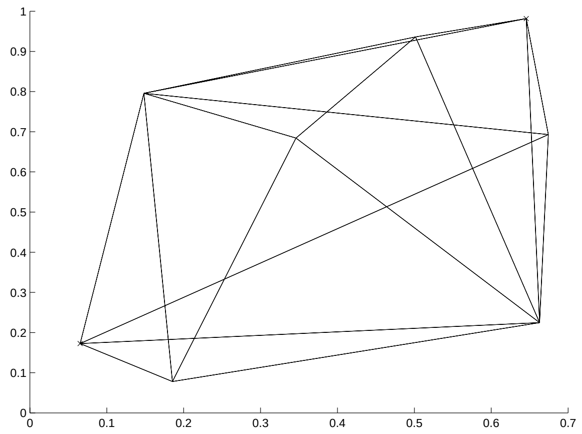

Figure 3.6: Wire-frame view of a 3D polytope and its extreme points in the [1 1 1] direction indicated by $\times$.

ELLADDE **ELLADDE**

Purpose

ELLADDE calculates an outer-bounding ellipsoid of the sum of two ellipsoids.

Usage

```
[e,p,v]   = elladde(e1,e2,rin)
```

where

```
e        -   the bounding ellipsoid
p        -   vector containing
                     the calculated parameter value,
                     the lower bound,
                     the upper bound for parameters
                     producing tight ellipsoids, and
v        -   vector containing
                     the corresponding values
                     (in case of rin numerical, it is set to 0-s.)
e1,e2    -   two ellipsoids of equal dimensions
rin      -   either a scalar parameter > 0, or
                     the string: 'vol', 'tr1', or 'tr2'.
```

Example

The following code generates two random ellipses and computes an external approximation to their Minkowski sum:

```
A=rand(2,3);e1=defell(A*A');
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],ax,'red');
e=elladde(e1,e2,'tr1');
ax=view2del(e,[1 2],ax,'r','--')
```

The ellipses and their external sum with trace criterion (denoted by dashed line) is displayed in the figure.

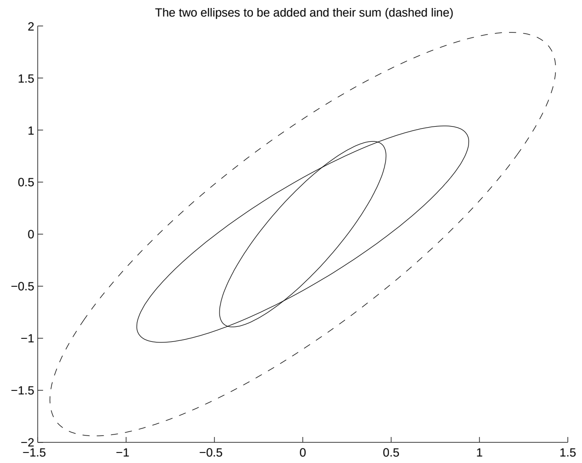

Figure 3.7: The two ellipses and their "external" sum (dashed line).

ELLADDI **ELLADDI**

Purpose

ELLADDI calculates an inner-bounding ellipsoid of the sum of two ellipsoids.
Usage

```
[e,S,v] = elladdi(e1,e2,S0)
```

where

```
e       -   the bounding ellipsoid,
S       -   matrix containing the calculated
              parameter matrix, (in case of S0 a matrix, S=S0.)
v       -   scalar containing det(Q) (in case of S0 a matrix, v=0.)
e1,e2   -   two ellipsoids of equal dimensions
S0      -   matrix parameter (positive definite)
```

Examples

The following code generates two random ellipses and computes an internal approximation to their sum, based on the volume criterion:

```
A=rand(2,3);e1=defell(A*A');
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],ax,'red');
e=elladdi(e1,e2);
ax=view2del(e,[1 2],ax,'r','--')
```

The ellipses and their "internal" sum with volume criterion (denoted by dashed line) is displayed in the figure.

ELLAPPRX **ELLAPPRX**

Purpose

ELLAPPRX calculates tight polytopes inside and outside an ellipsoid.
Usage

```
[Hout, Hin] = ellapprx(e,nord)
```

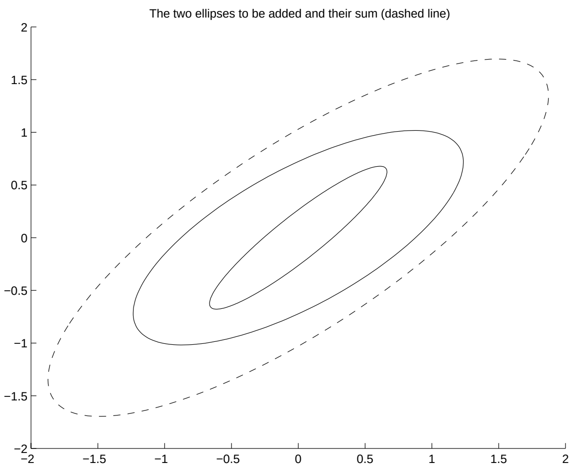

Figure 3.8: The two ellipses and their "internal" sum (dashed line).

where

```
  e     -   ellipsoids to be approximated
 nord   -   order of polytope approximations for method='polell'
  Hout  -   list of outer tangent hyperplanes
  Hin   -   list of inner hyperplanes
```

Example

The follwing code defines a random 3D ellipsoid and computes inner and outer bounding tight polytopes to it with approximation order 6 :

```
A=rand(3,3);E=defell(A*A');[Po,Pi]=ellapprx(E,6);ax=view2d(Po,[1 2]);
view2del(E,[1 2],ax);view2d(Pi,[1 2],'r',ax);
```

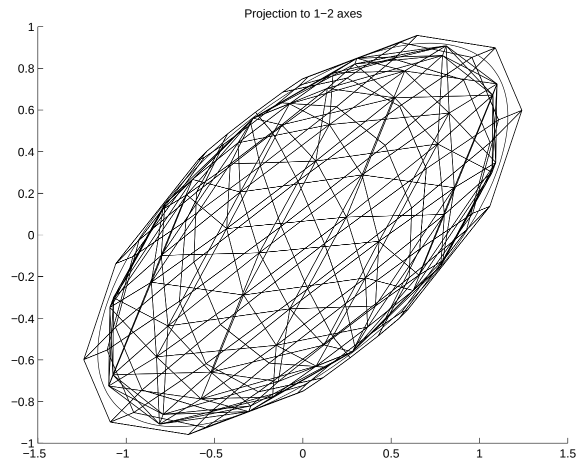

Figure 3.9: A random 3D ellipsoid and its 6th order inner and outer approximations by two polytopes.

ELLDIFE ELLDIFE

Purpose

ELLDIFE calculates an outer bounding ellipsoid of the difference of two ellipsoids.

Usage

```
[e,S,v] = elldife(e1,e2,S0)
```

where

```
e        -   the bounding ellipsoid,
S        -   matrix containing the calculated
                    parameter matrix,
                    (in case of S0 a matrix, S=S0.)
v        -   scalar containing det(Q).
                    (in case of S0 a matrix, v=0.)
e1,e2    -   two ellipsoids of equal dimensions
S0       -   matrix parameter (positive definite)
```

Example

The following code generates two random ellipses and computes an external approximation to their difference, based on the volume criterion:

```
A=rand(2,3);e1=defell(A*A');
A1=rand(2,2);e2=defell(A1'*A1);
ax=view2del(e1);
title('The two ellipses and their difference (dashed line)');
ax=view2del(e2,[1 2],ax,'red');
e=elldife(e1,e2);
ax=view2del(e,[1 2],ax,'r','--')
```

The ellipses and their "external" difference with volume criterion (denoted by dashed line) is displayed in the figure.

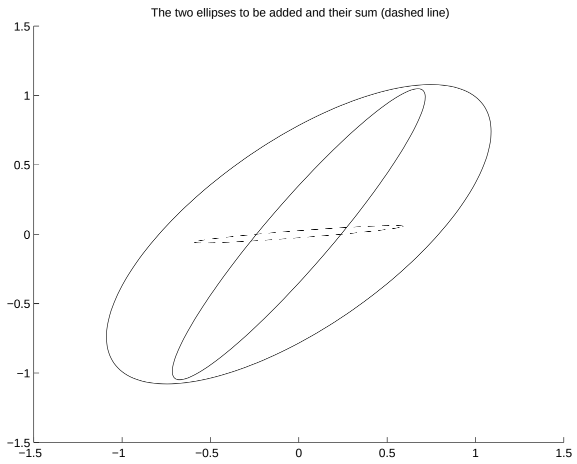

Figure 3.10: The two ellipses and their external approximation to their difference (dashed line).

ELLDIFI ELLDIFI

Purpose

ELLDIFI calculates an inner bounding ellipsoid of the difference of two ellipsoids.
Usage

```
[e,p,v] = elldifi(e1,e2,rin)
```

where

```
e       -   the bounding ellipsoid
p       -   vector containing
                the calculated parameter value,
                the lower bound,
                     the upper bound for parameters
                     producing tight ellipsoids, and
   v        -   vector containing
                     the corresponding values.
                     (in case of rin numerical, it is set to 0-s.)
  e1,e2    -   two ellipsoids of equal dimensions
  rin      -   either a scalar parameter < -1,
                     or the string: 'vol', 'tr1', or 'tr2'.
```

Example

The following code generates two random ellipses and computes an internal approximation to their difference, based on the trace criterion:

```
A=rand(2,3);e1=defell(25*A*A');
A1=rand(2,2);e2=defell(A1'*A1);
ax=view2del(e1);
title('The two ellipses and their difference (dashed line)');
ax=view2del(e2,[1 2],ax,'red');
e=elldifi(e1,e2,'tr1');
ax=view2del(e,[1 2],ax,'r','--')
```

The ellipses and their "internal" difference with trace criterion (denoted by dashed line) is displayed in the figure.

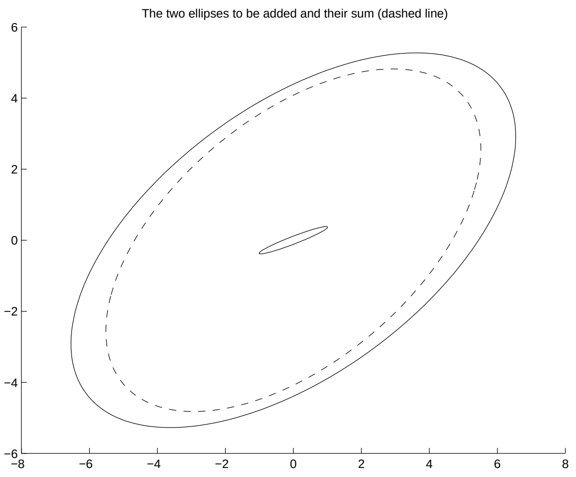

Figure 3.11: The two ellipses and internal approximation to their difference (dashed line).

ELLICOV ELLICOV

Purpose

ELLICOV extracts the covariance matrix from and ellipsoid representation.
Usage

```
                C = ellicov(E)
```

where

```
    E  -  ellipsoid in GBT format
    C  -  covariance matrix of ellipsoid defined by  [x-x0]'C^-1[x-x0]<1
```

ELLINT ELLINT

Purpose

ELLINT calculates an ellipsoid around the intersection of two ellipoids.
Usage

```
                E = ellint(e1,e2,method,nord)
```

where

```
    e1, e2   -  two ellipsoids to be intersected
    E        -  resulting ellipsoid
    method   -  = 1   for searching between convex
                 combinations of the two ellipsoid
                = 2 (default)  for computing first approximate
                 polytope hulls and fitting an ellipsoid around
                 the intersection
     nord   -  order of polytope approximations for method='polell'
```

Example

The following code defines two ellipses and computes an ellipse around their intersection:

```
E1=defell([1 0.3;0.3 1],[1;1]);E2=defell(eye(2)); E3=ellint(E1,E2);
ax=view2dell(E1);view2del(E2,[1 2],ax,'g');view2del(E3,[1 2],ax,'r');
```

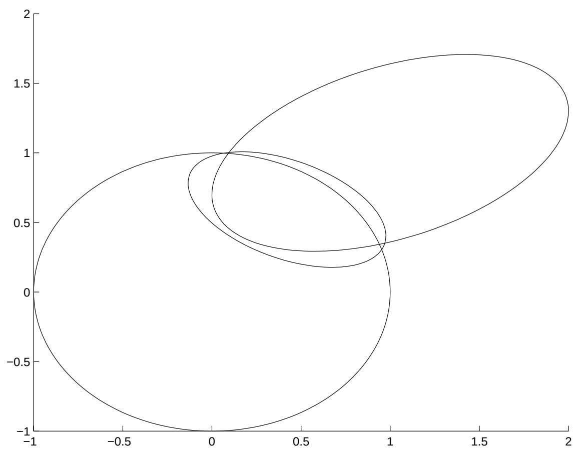

Figure 3.12: Two ellipses and a tight ellipse aroud their intersection.

ELLRANGE ELLRANGE

Purpose

ELLRANGE calculates the ranges of projections of an ellipsoid onto each of the coordinate axis.

Usage

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

where

```
 P     -  ellipsoid
vmin   -  column-vector of the lower limits
vmax   -  column-vector of the upper limits
```

ELLVOLUM ELLVOLUM

Purpose

ELLVOLUM calculates the volume of an ellipsoid.

Usage

```
                v=ellvolum(E)
```

where

```
   v  -  the volume of ellipsoid E
```

FACES FACES

Purpose

FACES returns the set of facet inequalities of a polytope and their number.

Algorithm

Direct operation for extracting the sub-matrix of facet inequalities.

Usage

```
[F,nh] = faces(P)
```

where

```
     P    -  polytope in GBT format
     F    -  list of hyperplane equations. Each row F(i) has the form
             [a(i)' b(i)]  with the meaning  a(i)'x <= b(i) where a(i) is
             unit vector
     nh   -  number of hyperplane faces
```

Example

The following code defines a 4D random polytope and computes its facets:

```
V=rand(6,4);P=convh(V);view2d(P,[1 2 3 4]);F=faces(P)
```

A wire-frame view of the 4D polytope P is displayed in Figure 3.13 from 6 directions in the 4D space. The list of the facets are:

```
F =
    0.0918    0.3062    0.2591   -0.9114   -0.0426
   -0.1263    0.0058    0.4420   -0.8881   -0.1666
   -0.1106    0.2058    0.0433   -0.9714   -0.3547
    0.5606    0.1956    0.0907   -0.7995    0.2484
    0.2043   -0.6760   -0.1191    0.6979    0.2001
   -0.0175    0.0242   -0.2982   -0.9540   -0.4937
    0.0202    0.0549    0.0003    0.9983    0.5577
   -0.0892   -0.1801   -0.0524    0.9782    0.3156
   -0.3071   -0.0202   -0.3080    0.9002    0.1428
```

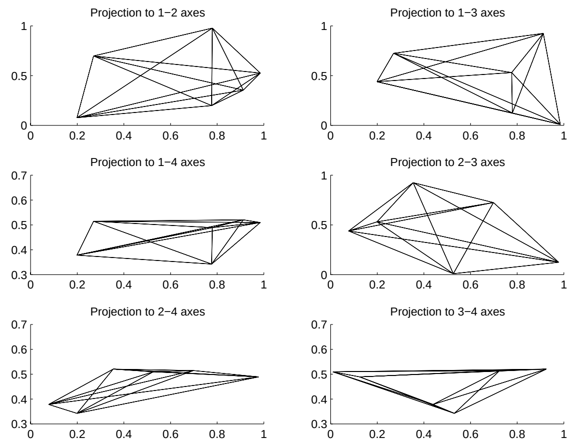

Figure 3.13: Wire-frame view of the 4D polytope.

FACET FACET

Purpose

FACET computes a facet of a polytope in lower dimension.

Algorithm

Given the index of the facet required, FACET computes the neighbouring facets and defines a same dimensional polytope representation. of the facet. This full dimensional representation is reduced to lower dimension by MINDIM.

Usage

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

where

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

Example

The following code defines a 3D random polytope and computes its facets:

```
V=rand(8,3);P=convh(V);

F=facet(P,1);view2d(P,[1 2 3]);figure(2);view2d(facet(P,2),[1 2]);
```

A wire-frame view of the 3D polytope P is displayed in Figure 3.14 from 3 directions in the 3D space. The first facet polytope F (in Figure 3.15) has the following matrix representation:

```
F=
     0         3.0000    0.0000
     0.8250   -0.5652   -0.1426
     0.2104    0.9776    1.2116
    -0.9143    0.4050    0.0782
     0.0741    0.3604   -1.0000
     0.4231    1.1483   -1.0000
     0.5894    1.1125   -1.0000
```

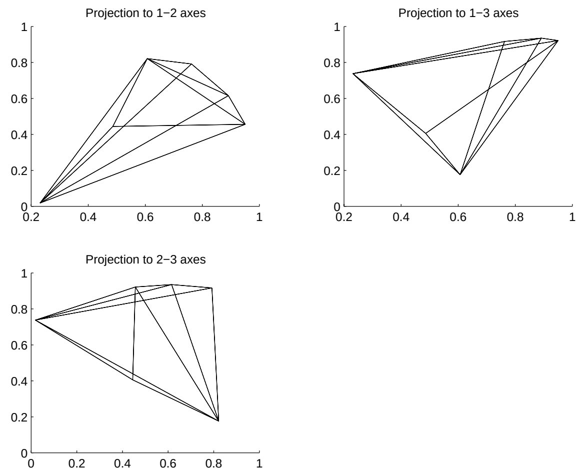

Figure 3.14: Wire-frame view of the 3D polytope P.

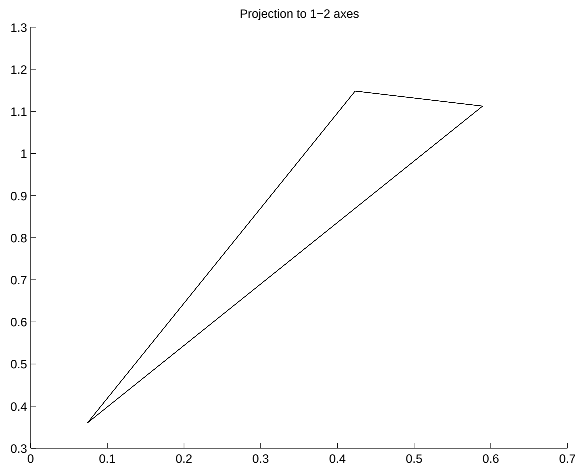

Figure 3.15: Wire-frame view of the first first facet of P, which is a 2D polytope.

FCONVH FCONVH

Purpose

FCONVH computes the intersection of a polytope and a set of half-spaces.

Algorithm

This is a direct algorithm based on the use of UPDATE. If no initial polytops is defined then the enclosding hypercube with edge size 2*KAPPA is used as initial polytope.

Usage

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

where

```
H      -  =[A b] a set of half spaces defined by Ax<=b
P      -  a polytope
P1     -  is the intersection polytope
acc    -  actual vertex-facet fitting accuracy
```

Example

The following code defines a random polygon and updates it with a half-plane:

```
V=rand(6,2)-0.5;P=convh(V);h=[1 1 0]; P1=fconvh(h,P);
aou=view2d(P);view2d(P1,[1 2 ],aou,'g');hold on;plot(0,0,'+');
```

Polygon P is updated by a half-plane through the origin in Figure 3.16.

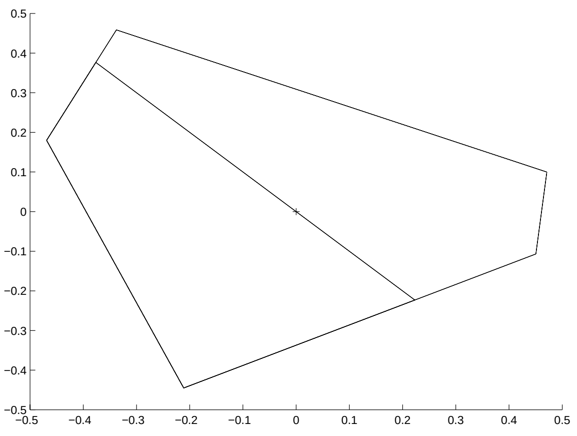

Figure 3.16: The polygon updated by a half-plane through the origin.

FV FV

Purpose

FV computes lists of vertex indices fitting to each facet.
Usage

```
[FV,nv,nh]=fv(P)
```

where

```
P   -  a polytope
FV  -  vertex-facet adjacency table
nv  -  number of vertices
nh  -  number of facets
```

FVFITACC FVFITACC

Purpose

FVFITACC calculates the worst-case facet-vertex inaccuracy for a polytope.
Usage

```
mu=fvfitacc(P)
```

where

```
mu - the maximum inaccuracy of facet-vertex fits in polytope P
     measured in terms of  abs(a*v'-b) for a vertex v and facet [a b]
```

INELL INELL

Purpose

INELL tests whether a point is in an ellipsoid
Usage

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

where

```
bool - Boolean variable indicating whether point p is inside ellipsoid E
```

INTERSCT INTERSCT

Purpose

INTERSCT computes the intersection of two polytopes.

Algorithm

First the faces of the polytopes are extracted and one of the polytopes is updated by the facets of the polytope with the smaller number of facets.

Usage

```
 P = intersct(P1,P2);
```

where

```
  P1, P2  -  polytopes
     P    -  the calculated intersection of P1 and P2
```

Example

The following code defines two 4D random polytopes and computes their intersection:

```
V=rand(6,4);P1=convh(V);V=rand(6,4);P2=convh(V); P=intersct(P1,P2);
ax=view2d(P1,[1 2],'r');view2d(P2,[1 2],'g',ax);view2d(P,[1 2],'k',ax);
```

A wire-frame view of the two 4D polytopes P1 and P2 are displayed in Figure 3.17 together with there intersection as projected onto the 1-2 plane:

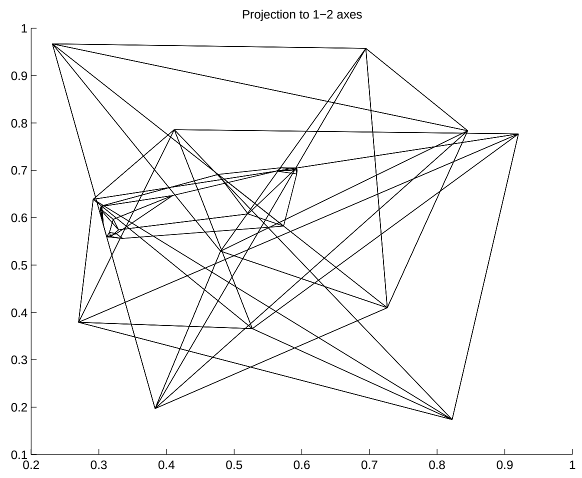

Figure 3.17: Wire-frame view of two 4D polytopes and their intersection projected ont the 1-2 plane.

INTEST INTEST

Purpose

INTEST tests whether a given set of points is inside a given polytope with accuracy δ .

Algorithm

A direct algorithm which checks for each facet-plane which side the points falls into.

Usage

```
T=intest(V,P);
```

where

```
V     -  set of points
P     -  polytope
```

Example

The following example defines an 8D random polytope and tests whether the origin falls inside:

```
V=rand(12,8)-0.5;P=convh(V);
orig_in=intest(zeros(8,1),P);
```

INTPOINT INTPOINT

Purpose

INTPOINT finds an internal point within a given polytope with given margin δ from each facet.

Algorithm

The centre of mass is tested whether it is to a distance of at least δ from all facets. If not than 1000 random points are generated around the centre of mass to find a suitable point. If none of the randomly generated points has a δ-margin, the routine returns an empty matrix.

Usage

```
        p=intpoint(P);
```

where

```
P     -  is a polytope in GBT 7.0  format
p     -  point inside the polytope
```

Example

The following code defines a random 7D simplex and produces and internal point with margin $\delta = 0.001$:

```
V=rand(7,6);P=convh(V);p=intpoint(P);view2d(P,[1 2]);

hold on;plot(p(1),p(2),'x');
```

The wire-frame view of the resulting 7D simplex is shown in Figure 3.18 projected onto the 1-2 plane.

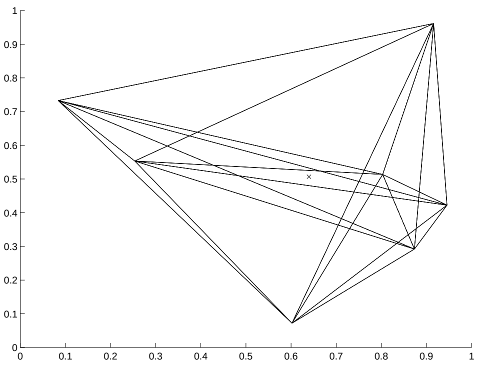

Figure 3.18: Wire-frame view of a 7D simplex and one of its internal points projected onto the 1-2 plane.

LTRAN LTRAN

Purpose

LTRAN can be used to compute an affine mapping of a polytope which will be another polytope. It is allowed that the polytope operated on is not full dimensional. Also, depending on the rank of the linear mapping, the resulting polytope may be not full dimensional. For a polytope P in the m-dimensional space, the affine map of P by a linear transformation $\bar{A}$ defined by matrix A ($A \in \Re^n \times \Re^m$), and shift vector $a \in \Re^n$, is defined by

$$\bar{A}P \overset{def}{=} \{y \in \Re^n \mid y = Ax + a,\ x \in P\}$$

The resulting polytope will be represented in the n-dimensional space but will not necessarily be full dimensional, depending on whether A is of full rank or whether P is m-dimensional or not.

Algorithm

First the vertices of P are extracted and each of them is mapped by matrix A. Then the convex hull of the mapped set is computed by CONVH. Also the lowest upper bound for the accuracy of vertex-facet adjacency is computed by CONVH and is passed on to LTRAN.

Usage

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

where

```
Pt    - the transform of P
A     - transformation matrix
a     - offset vector (column vector with the same number
        of rows as A )
P     - polytope to be transformed
acc   - resulting accuracy of facet-vertex fits
```

Example

The following code defines a 4D polytope which is mapped into 2D by a given affine transformation matrix.

```
[P,acc]=convh([1 3 1 2;0 1 0 0;2 3 -1 -2;-1 1 1 1;-1 -2 2 1]);
figure(1);view2d(P,[1 2 3 4]); A=[ 1 0 0.1 -1;-1 0 1 -1];a=[-1 1];
[Pa,acc]=ltran(P,A,a); figure(2);view2d(Pa,[1 2]);
```

The result of this code is displayed in Figures 3.19 and ??.

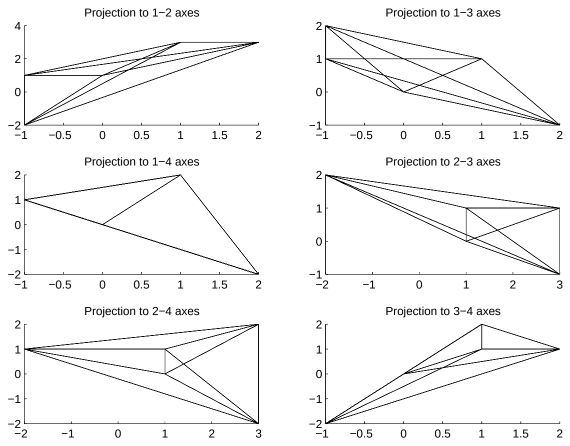

Figure 3.19: 2D wire-frame projections of the 4D polytope P.

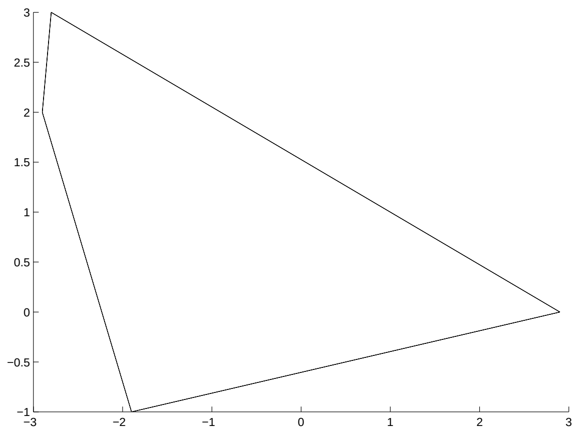

Figure 3.20: The 2D polytope Pa obtained by the affine transformation defined by A applied to P.

MAXFACE MAXFACE

Purpose

Given the number of vertices n in the d-dimensional space, MAXFACE computes the vector containing the maximum number of possible lower-dimensional faces for each dimension from 1 to $d-1$.

Algorithm

Applies direct formulas for the maximum number of faces achievable by constructing cyclic polytopes as defined in [16] .

Usage

```
f = maxface(n,d);
```

where

```
n    -   number of vertices
d    -   dimension
f    -  (d-1)-dimensional vector containing the maximal number
         of faces
```

Example

The following command applies MAXFACE for 200 vertices in 6D:

```
f=maxface(200,6)
```

The result is:

```
f =
     1293900
     1313400
     3841700
     3822000
     1274000
```

Here f(1) contains the maximum number of 1D faces, i.e. edges, f(2) indicates the maximum number of 2D faces, etc. f(5) indicates the maximum possible number of 5D facets.

MAXVERT MAXVERT

Purpose

Given the number n of half-spaces in the d-dimensional space, MAXVERT computes the maximum number of possible vertices which is actually achievable for a cyclic polytope [16].

Algorithm

By duality it applies MAXFACE and takes the (d-1)-component of the resulting vector [16].

Usage

```
            nov = maxvert(n,d);
```

where

```
   n    -  number of half-spaces
   d    -  dimension
   nov  -  maximum number of vertices
```

Example

The following command applies MAXFACE for 200 vertices in 6D:

```
nov=maxvert(200,6)
```

The result is:

```
nov =
      1274000
```

which is the last component of the vector obtained in the example provided for MAXVERT.

MCVOL **MCVOL**

Purpose

MCVOL estimates the volume of the intersection of a set of ellipsoids by the relative number of uniform rectangular grid points falling into the intersection.

Usage

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

where

```
E  -  list of ellipsoids in standard GBT format
n  -  degree of approximation n>10, the number of grid
      points in each coordinate axis
```

Example

The following example defines 5 random ellipsoids in 4 dimensional space and estimates the volume of their intersection:

```
E=[];for i=1:5,
    A=rand(4,4);P=25*A'*A;          % generating random covariance matrix
    E=[E;defell(P,rand(4,1)-0.5)]; % building list of elllipsoids
end;
vol=mcvol(E,10);
```

P_CONV P_CONV

Purpose

P_CONV computes the convex hull of a polytope and a given point: i.e. the routine"includes" a new point into the convex hull.

Algorithm

A direct algorithm is applied which computes first the "visible facets" of the polytope from the point. Then cones are computed for each visible facet with a peak at the point given. All the cones obtained are united with the polytope to obtain the new convex hull.

Usage

```
            [O,acc]=p_conv(xn,P);
```

where

```
  xn    -   point to be included in the convex hull
  P     -   polytope in GBT 7.0  format
  O     -   convex hull polytope of "xn" and "P"
  acc   -   accuracy of facet-vertex fits at the output in O
            note that also O(1,3)=acc is defined for d>1 .
```

Example

The following code defines a random 4D simplex and "includes" [1 1 1 1] into the convex hull.

```
V=rand(7,4);P=convh(V);p=[1 1 1 1 ];P1=P_CONV(p,P);view2d(P1,[1 2 3 4]);
```

Then the wire-frame view of the resulting 4D convex hull is shown in Figure 3.21 from 6 directions projected onto 2D planes.

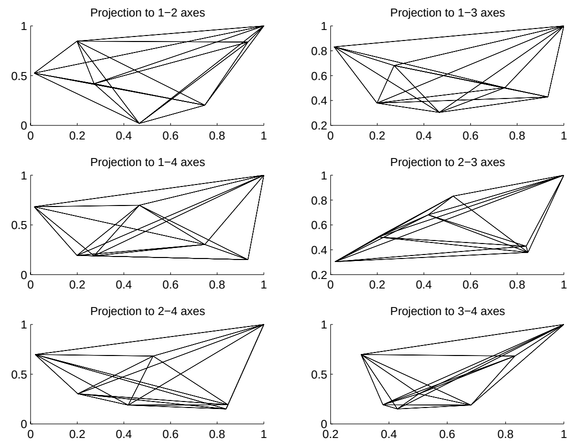

Figure 3.21: Wire-frame view of a 7D simplex and one of its internal points projected onto the 1-2 plane.

POLADD POLADD

Purpose

POLADD computes the algebraic sum (i.e. Minkowski sum) of two polytopes. For polytopes P_1 and P_2 the sum can be defined as

$$P_1 \oplus P_2 \overset{def}{=} \{x + y \mid x \in P_1,\ y \in P_2\}$$

Algorithm

The algorithm takes the sums of vertices of the two polytopes in evry combination and then the convex hull algorithm is used to the set of points obtained.

Usage

```
[P,acc] = poladd(P1,P2)
```

where

```
P1, P2   -   first and second geometric object
   P     -   the sum of P1 and P2
  acc    -   resulting accuracy of facet-vertex fits
             P(1,3)=acc by definition for d>1
```

Example

The following code defines a rectangle and a quadrilateral polygon obtained as the convex hull of four given points. The sum of these two polygons is then computed by POLADD:

```
P1=defbox([-1 -2],[0 3]);
 P2=convh([1 3;0 1;2 3;-1 1]);
 P=poladd(P1,P2);
ax=view2d(P,[1 2],'r');
view2d(P1,[1 2],'b',ax);
  view2d(P2,[1 2],'g',ax);
```

The result of this code is displayed in Figure 3.22.

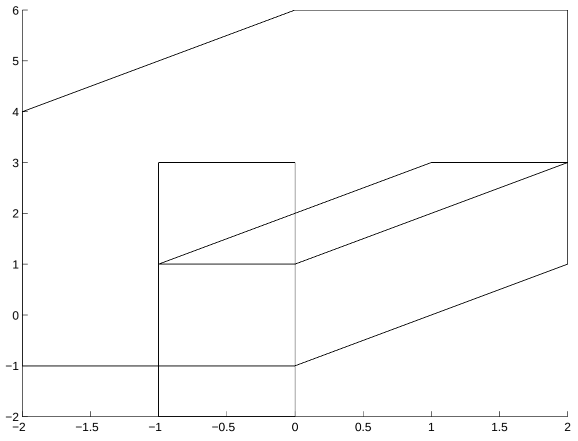

Figure 3.22: Addition of two polygons: the [-2 -1;0 -1;2 1;2 6;0 6;-2 4] polygon is the sum of the polygon [-1 -2;0 3] and the polygon [1 3;0 1;2 3;-1 1] .

POLCEN POLCENT

Purpose

POLCENT can be used to compute the ℓ_p Chebyshev centre of a given polytope. The ℓ_p-centre of a polytope P is defined by

$$centre_p(P) \overset{def}{=} arg \min_{y \in \Re^m} \sup_{x \in P} ||y - x||_p$$

Algorithm

This is an algorithm by A. Kuntsevich [13].

Usage

```
[c,f] = polcent(P,typ,p)
```

where

```
P    -   a polytope
c    -   centre of ellipsoid or polytope
typ  -  'mass'  - centre of mass for the vertices taken with unit mass
        'cheb'  - the Chebishev centre in lp-sense
          The default of 'type' is 'cheb'.
p    -   a real number p >= 1 , p < inf
```

Example

The following code defines a 4D polytope an POLCENT computes its centre according to the four most important interpretations:

```
[P,acc]=convh([1 3 1 2;0 1 0 0;2 3 -1 -2;-1 1 1 1;-1 -2 2 1]);
c1=polcent(P,'cheb',1); c2= polcent(P,'cheb',2); c12=polcent(P,'cheb',12);
cmass=polcent(P,'mass'); i=2;j=4;
view2d(P,[i j]); hold on;
plot(c1(i),c1(j),'r.');
plot(c2(i),c2(j),'r+');
plot(c12(i),c12(j),'ro');
plot(cmass(i),cmass(j),'r*')
```

The result of this code is displayed in Figure 3.23.

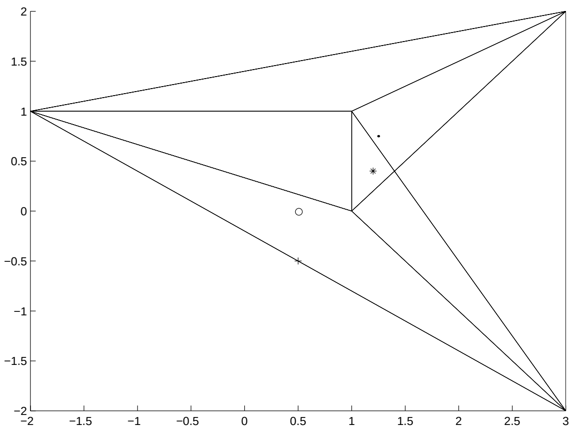

Figure 3.23: Wire-frame of P is projected to the 2D-plane defined by the 2-4 axes. The different types of centres of polytope P are wide apart: ℓ_1-centre: $\bullet$; ℓ_2-centre: $+$; ℓ_{12}-centre (near to ℓ_∞ centre): $\circ$; centre of mass: $*$.

POLDUAL POLDUAL

Purpose

POLDUAL computes the dual of a polytope which contains the origin of the co-ordinate system inside.

Algorithm

The algorithm converts the vertices into hyper-planes and the facets into vertices in the dual space.

Usage

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

where

```
  P   -  a polytope in GBT 7.0  format
  D   -  dual of  P  in GBT 7.0  format
acc   -  vertex-facet fitting actual accuracy obtained
         D(1,3)=acc by definition
```

Example

The following code defines a box and then computes its dual:

```
P=defbox([-1 -2 -1 -2],[1 1 1 2]);

D=poldual(P); view2d(D,[1 2 3 4]);
```

The result of this code is displayed in Figure 3.24.

POLVOL POLVOL

Purpose

POLVOL computes the volume of a given polytope.

Algorithm

The algorithm computes the volume recursively relying on the lower-dimensional volumes of its facets. A detailed description of the algorithm by Lasserre can be found in [17] .

Usage

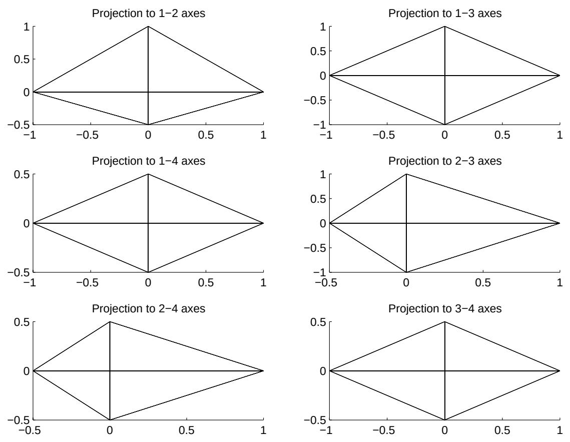

Figure 3.24: Wire-frame projections of the diamond shape dual polytope to the box defined.

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

where

```
P     -    polytope in GBT 7.0  format
```

Example

The following code defines a random 4D polytope displays its wire-frame view in Figure 3.25 and computes the projection of the [1 1 1 1] onto its surface:

```
V=[-1 1 -2 -1; 1 0 -1 1;2 -2 1 -1 ; 1 -1 2 1; 3 1 1 -1 ; -1 -1 -1 1];
P=convh(V); view2d(P,[1 2 3 4]); vol=polvol(P)
```

where

```
vol =
      4.5000
```

The following code defines a 4D polytope, displays its wire-frame view in Figure 3.25 and computes its volume:

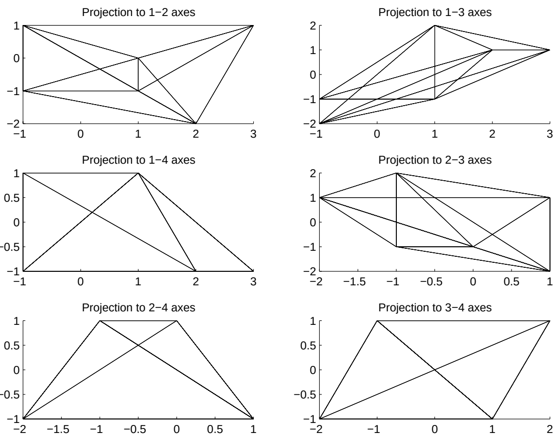

Figure 3.25: Wire-frame projections of the 4D polytope for volume computation.

PROJPONT PROJPONT

Purpose

PROJPONT computes the projection of an external point onto the surface of a polytope.

Algorithm

The algorithm computes all visible faces of all dimensions from the point. Then the projections to the visible-faces are computed and the projection point falling onto the surface of the polytope is selected.

Usage

```
p=projpont(p,P);
```

where

```
p  -   point to be projected
P  -   polytope onto which projection is made
```

Example

The following code defines a random 4D polytope, displays its wireframe view in Figure 3.26 and computes the projection of the [1 1 1 1] onto its surface:

```
V=rand(7,4); P=convh(V); view2d(P,[1 2 3 4]); p=projpont([1 1 1 1],P)
```

The result of the projection is the point p shown:

```
p =
    0.9687
    0.9659
    1.0641
    0.9449
```

The result of this code is displayed in Figure 3.26.

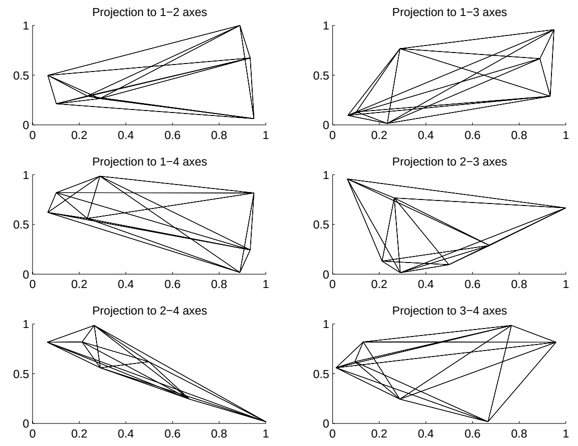

Figure 3.26: Wire-frame projections of the random 4D polytope.

SEL **SEL**

Purpose

SEL finds an ellipsoid of the minimal volume containing a set of given points.

Usage

```
E=sel(C,method,options)
```

where

```
C        -  set of points, each row represents a point
method   -  method to be used. Choose
            1 : minimize the non-smooth penalty function (direct
                problem)
                using the algorithm with space dilatation
```

```
              2 : the method of successive space dilatation
              3 : minimize the non-smooth penalty function (dual problem)
                  using the algorithm with space dilatation
              The default is '2'
  options  -  an array of control parameters of iterative algorithms
             options(1) : the termination tolerance for the coefficients
                          of matrix E ( if method==1 or method==3)
             options(2) : maximal number of iterations
                            (the default is [1.e-4,150])
  E        -  resulting ellipsoid in GBT format
```

Example

The following code generates 10 random vectors in 2D and computes the samllest volume ellipsod around them.

```
V=rand(8,2);E=sel(V);

figure;view2del(E);hold on;plot(V(:,1),V(:,2),'gx');
```

The figure displays the 8 points and the fitted ellipsoid.

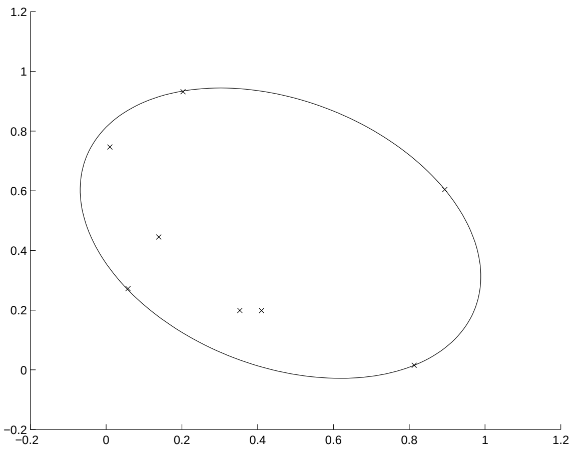

Figure 3.27: A tight ellipse around the 10 points.

SETDELTA SETDELTA

Purpose

SETDELTA is an interactive m-file to specify the edge-size of the hypercube which will contain all polytopes to be computed with.

Usage

```
setdelta
```

prompts for κ, i.e. the global variable **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]: KAPPA=
```

after which the global variable δ, i.e. **DELTA** is set and displayed:

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

DELTA =

  1.0000e-011
```

SUBTRACT SUBTRACT

Purpose

SUBTRACT computes the algebraic difference of two polytopes. For polytopes P_1 and P_2 the algebraic difference is defined by

$$P_1 \ominus P_2 \overset{def}{=} \{x \in P_1 \mid \forall y :\in P_2 \; x + y \in P_1\}$$

Algorithm

The algorithm first extracts the vertices of P_2 and using these computes a system of linear equations for the difference $P_1 \ominus P_2$. The dual of the convex hull algorithm is then applied to obtain the polytope in GBT 7.0 format.

Usage

```
D = subtract(A,B)
```

where

```
A, B  -  two polytopes for which the difference is sought
 D    -  the difference polytope (can be empty, i.e. [ ] )
```

Example

The following code defines two random polygons of different sizes and computes their difference:

```
V2=10*rand(9,2);P2=convh(V2);V1=2*rand(5,2)-1;P1=convh(V1);

P=subtract(P2,P1,0.00001);

view2d(P2,[1 2]);view2d(P1,[1 2],'b',1);view2d(P,[1 2],'g',1);
```

Polygons P_2 and P_1 and their difference are shown in Figure 3.28.

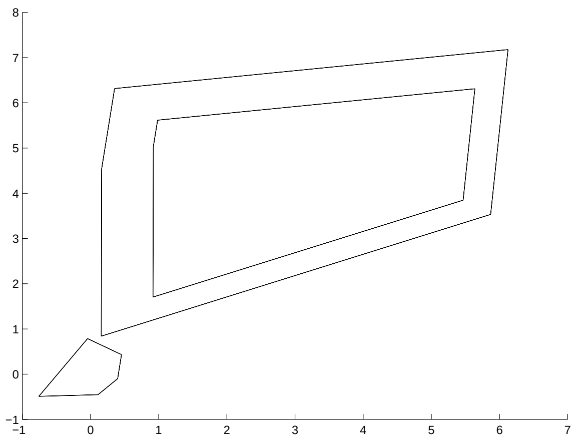

Figure 3.28: Two polygons and their difference.

UNI UNI

Purpose

UNI computes the union set of two sets of vectors by reducing the number of multiple copies. Two vectors are considered equal if their ℓ_1-norm distance is less than a given δ accuracy limit.

Algorithm

The algorithm examines each new addition to the set by testing its distance from existing members for it being greater than δ in the ℓ_1-sense.

Usage

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

where

```
U1,U2   -  matrices of row vectors
  U     -  the union
```

Example

The following code defines two random set of vectors with 3 vectors shared between them. Then the union set is computed and its number of vectors is checked:

```
V1=rand(10,4);V2=[V1(1:3,:);rand(6,4)]; V=uni(V1,V2,0.0001); nv=size(V,1)
```

UPDATE UPDATE

Purpose

UPDATE computes a new polytope which is the intersection of half-spaces or the intersection of half-saces and a polytope. The half-spaces are to be described by linear inequalities.

Algorithm

The algorithm is based on analysing which vertices of the polytope fall inside or outside of the given half-space.

Usage

```
[P1,acc]=update(h,P);
```

where

```
h     -   a row vector representing a linear inequaility
P     -   Polytope in GBT 7.0 - format
acc   -   fitting accuracy of hyperplanes to vertices
```

Example

The following code defines 7 random inequalities in 4D and computes the intersection of the corresponding half-spaces and an initial box with edge size 2000:

```
IQ=[rand(7,4)-0.5 ones(7,1)]; v=1e+3*ones(1,4);P=defbox(-v,v);

for i=1:7, P=update(IQ(i,:),P);end;

view2d(P,[1 2 3 4]);
```

Then the wire-frame view of the resulting polytope is displayed from 6 directions in Figure 3.29.

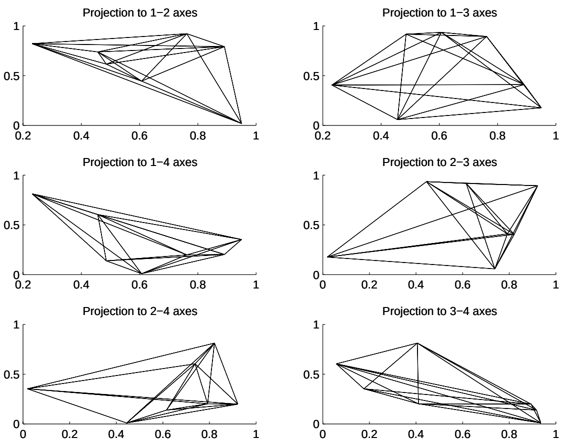

Figure 3.29: The convex hull of 7 random points in 4D.

VERTICES VERTICES

Purpose

VERTICES extracts the vertices of a polytope and lists them in an array so that each row contains the co-ordinates of on vertex.

Algorithm

Straightforward sub-matrix operation based on the matrix representation of a polytope in GBT 7.0 .

Usage

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

where

```
P   -  polytope in format of GBT 7.0
V   -  array of vertex row-vectors
nv  -  number of vertices
```

Example

The following code defines a random polygon and extracts its vertices.

```
F=[rand(6,2)-0.5 ones(6,1)];P=fconvh(F); [V,nv]=vertices(P);

view2d(P,[1 2]); hold on; for i=1:nv, plot(V(i,1),V(i,2),'+');end;
```

The vertices of the polytope with random faces in Figure 3.30 are indicated by + signs.

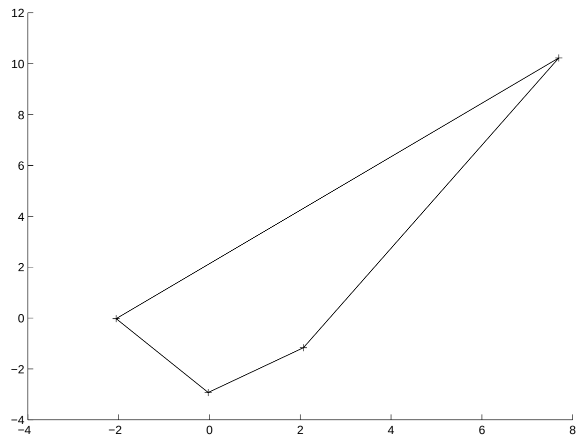

Figure 3.30: Wire-frame projections of a random polytope.

VIEW2D VIEW2D

Purpose

VIEW2D displays projections of wire-frame views of a polytope onto 2D-planes defined by pairs of axes. The number of view displayed can be one of $\begin{pmatrix} 2 \\ 2 \end{pmatrix} = 1$, $\begin{pmatrix} 3 \\ 2 \end{pmatrix} = 3$, $\begin{pmatrix} 4 \\ 2 \end{pmatrix} = 6$, depending on whether 2, 3 or 4 indices of axes are defined.

Algorithm

First the vertex-facet adjacency table is computed. Then the projection of 1D edges are displayed in the suitable figure. Settings of color and overlay of objects are taken into account.

Usage

```
          aout=view2d(P,a,c,hold,ax,acc);
```

where

```
   P    -  the name of the object
   a    -  a vector of indices of axes for which
           all 2D projections are to be displayed
   c    -  colour of object ('b','r','g', etc.)
   ax   -  axes handle to be used
   aout -  axes handle
```

Example

The following code defines a random polygon and extracts its vertices.

```
V=rand(6,3)-0.5;P=convh(V);h=[1 1 1 0]; P1=fconvh(h,P,0.00001);

figure(1);clf;view2d(P,[1 2 3]);

figure(2);clf;aou=view2d(P,[1 2 3]);view2d(P1,[1 2 3],'g',1,aou);

V1=rand(10,5);P1=convh(V1);figure(3);clf;view2d(P1,[1 2 3 4]);
```

A random 3D polytope is cut by a half-space going through the origin and the new and old polytopes are superimposed in Figures 3.31 and 3.32. Figure 3.33

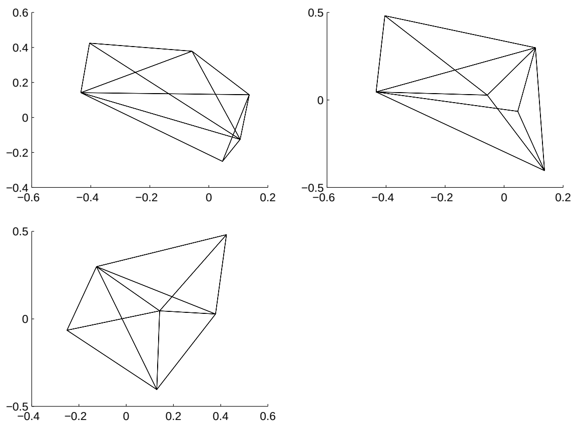

Figure 3.31: Wire-frame projections of a 3D random polytope.

displays 6 projected wire-frame views of a randomly generated 5D polytope.

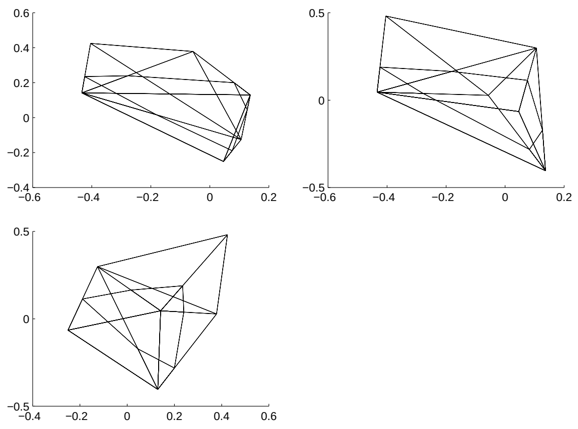

Figure 3.32: The polytope is cut by plane [1 1 1 0] and the new polytope is superimposed.

VIEW3D VIEW3D

Purpose

VIEW3D displays 3D views of a 3D polytope.

Algorithm

First the visible facets are computed and the associated facet-vertex adjacency table. Then a patch object is constructed for the view of visible facets. Finally colors and shading are introduced depending on the light directions and rendering options which can be defined separately for the 3D axes. There is full compatibility with the facilities of 3D viewing in MATLAB as the view is a patch object placed in an axes.

Usage

```
          [axout,p,ligh]=view3d(P,axin,style,pcolor)
```

where

```
P      -  is the name of the 3D polytope to be diplayed
axin   -  existing axes where the polytope should be displayed
style  -  'plane' of 'interp' (default)
pcolor -  3-vector to define the basic colour of the object
          (standard  format of colors in MATLAB )
p      -  object handle of the patch object of the polytope
ligh   -  handle of the light source introduced
```

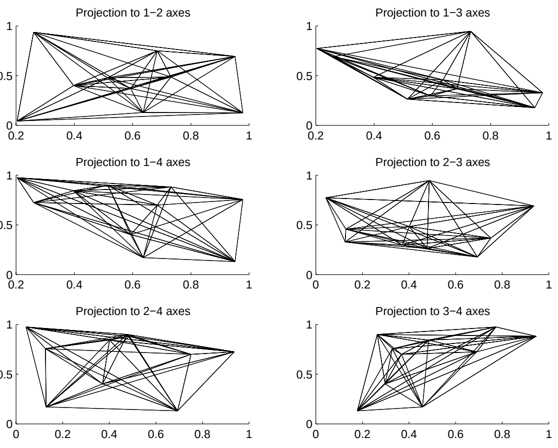

Figure 3.33: A randomly generated 5D polytope's wire-frame projections onto 6 axis-planes.

Example

The following code defines a random polytope box and displays its wire-frame view as shown in Figure 3.36.

```
figure;P1=convh(rand(4,3));P2=convh(rand(8,3));

ax=view3d(P1);view3d(P2,ax);
```

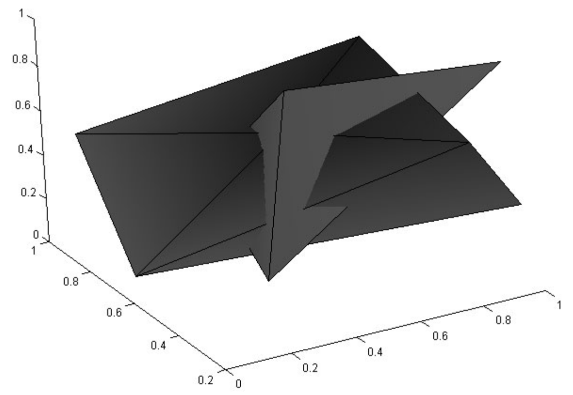

Figure 3.34: 3D view of two random polytopes superimposed.

newpage ___

VIEW2DEL **VIEW2DEL**

Purpose

VIEW2DEL displays 2D views of an ellipsoid.

Algorithm

First the surface of a sphere is transformed into an ellipsoid surface and iyt is deisplayed with the SURFL or MESH routines, depending on a type of plot reuqired. The handles of the axes, surface object and light are returned and can be further mulipluted by the user in a standard way as it is usual in MATLAB graphics. Angle of view can be set by VIEW and light sources can be defined by LIGHT as standard in MATLAB.

Usage

```
        [axout,p,ligh]=view2del(E,a,axin,col)
```

where

```
E     = ellipsoid
a     = list of axis indices
axin  = handle of axes to be used
col   = colour of ellipsoid projection required
```

Example

The following code defines two ellipses and computes an ellipse around their intersection and displays them all using VIEW2DEL:

```
E1=defell([1 0.3;0.3 1],[1;1]);E2=defell(eye(2)); E3=ellint(E1,E2);
ax=view2dell(E1);view2del(E2,[1 2],ax,'g');view2del(E3,[1 2],ax,'r');
```

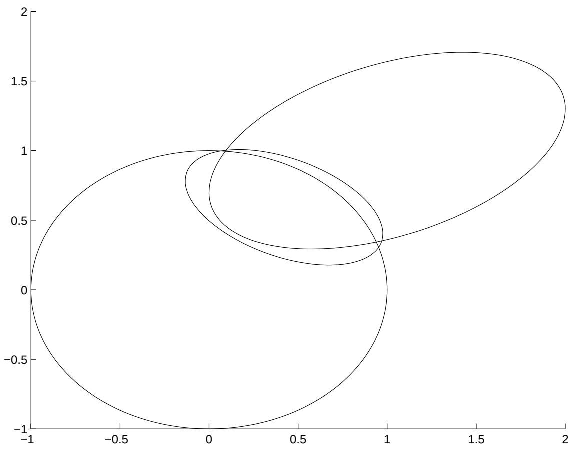

Figure 3.35: Two ellipses and a tight ellipse aroud their intersection.

VIEW3DEL VIEW3DEL

Purpose

VIEW3DEL displays 3D views of a 3D ellipsoid.

Algorithm

First the surface of a sphere is transformed into an ellipsoid surface and it is displayed with the SURFL or MESH routines, depending on a type of plot required. The handles of the axes, surface object and light are returned and can be further altered by the user in a standard way as it is usual in MATLAB graphics. Angle of view can be set by VIEW and light sources can be defined by LIGHT as standard in MATLAB.

Usage

```
        [axout,p,ligh]=view3del(P,axin,style,resol)
```

where

```
P      -  is the name of the 3D polytope to be diplayed
axin   -  existing axes where the polytope should be displayed
style  -  'smooth' (default) of 'mesh'
resol  -  number of divisions used in each dimension to approximate
axout  -  axes handle
p      -  object handle of the patch object of the ellipsoid
ligh   -  handle of the light source introduced
```

Example

The following code defines a random polytope box and displays its wire-frame view as shown in Figure 3.36.

```
A=rand(3,3)-0.5;E1=defell(A*A');ax=view3del(E1);

for i=1:3,
   A=rand(3,3)-0.5;E=defell(A*A');ax=view3del(E,ax);
end;
```

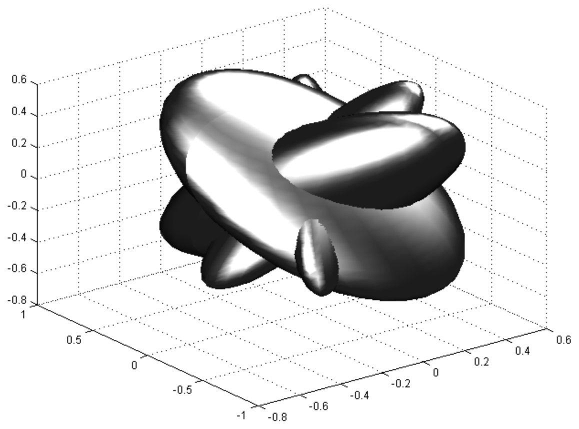

Figure 3.36: 3D view of four random ellipsoids centered and superimposed.

VREDUCE VREDUCE

Purpose

VREDUCE reduces the number of points in space by grouping into one the δ-near ones.

Algorithm

The first vector is compared with the rest and the δ-near ones in ℓ_2-sense are discarded. The first vector is included in the reduced set. From the remaining points again the first is taken and compared with the rest and the procedure is repeated. This operation continues until the remaining set becomes empty and the new reduced set is built up.

Usage

```
nV=vreduce(V);
```

where

```
V    -   given set of  points - one point in each row
nV   -   new set of points
```

Example

The following code defines a large set of random points, reduces them to with accuracy 0.01 and takes the approximate convex hull of the points:

```
N=200;V=0.3*(rand(N,3)-0.5)+0.5*ones(N,3);

V1=vreduce(V,0.01);P1=convh(V1,0.0001);

figure(1);clf;outh=view2d(P1,[1 2 3]);

axes(outh(1));hold on;for i=1:N, plot(V(i,1),V(i,2),'.');end;
axes(outh(2));hold on;for i=1:N, plot(V(i,1),V(i,3),'.');end;
axes(outh(3));hold on;for i=1:N, plot(V(i,2),V(i,3),'.');end;

figure(2);clf;outh=view2d(P1,[1 2 3]);N1=size(V1,1);

axes(outh(1));hold on;for i=1:N1, plot(V1(i,1),V1(i,2),'.');end;
axes(outh(2));hold on;for i=1:N1, plot(V1(i,1),V1(i,3),'.');end;
axes(outh(3));hold on;for i=1:N1, plot(V1(i,2),V1(i,3),'.');end;
```

Figure 3.37 displays the original cloud of points and the reduced complexity polytope fitted to their convex hull. Figure 3.38 only displays the reduced set of points from which the reduced complexity polytope was obtained.

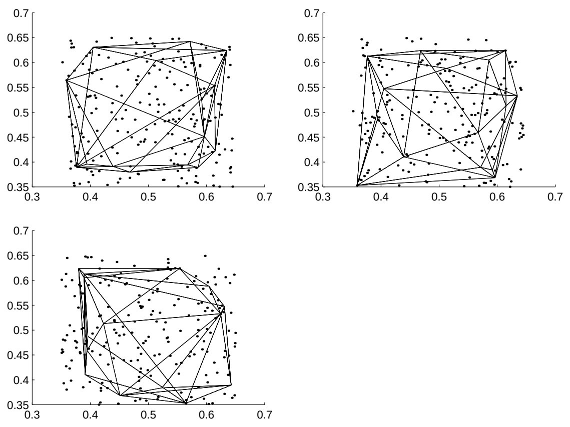

Figure 3.37: The original large set of points and the approximate polytope fitted.

VVF **VVF**

Purpose

VVF computes the vertex-vertex and vertex-facet adjacency tables of a polytope with given vertex-facet fitting accuracy $\delta > 0$. The tables of adjacency use indices of the vertices and hyper-planes in the order as they occur in the matrix representation of the polytope. Hence the width of the the tables depend on the maximum number of adjacent vertices and facets, respectively.

Algorithm

The Boolean-table of vertex-facet adjacency is computed first. Next the tables of vertex-vertex adjacency and tables of vertex-facet adjacency are constructed by straightforward operations.

Usage

```
[VV,VF,nv,nh]=vvf(H);
```

where

```
VV    -  vertex-vertex adjacency table
VF    -  vertex-facet adjacency table
nv    -  number of vertices
nh    -  number of facets
```

Example

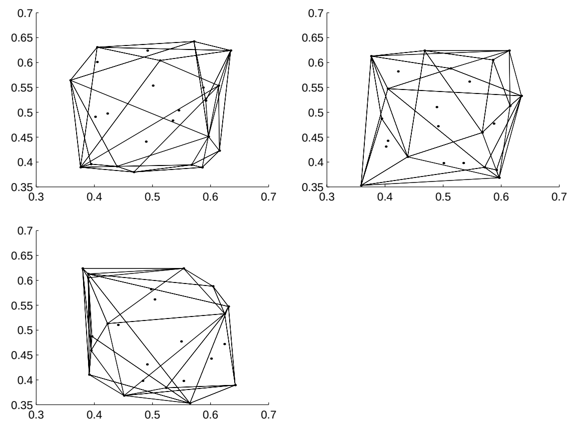

Figure 3.38: The reduced set of points and the polytope fitted.

The following code defines a polytope and produces the adjacency tables.

```
V=[-0.1 -0.3 0.4;0.2 0.5 0.1;-0.1 0.2 0.15; 0.1 0.05 -0.2;-0.3 -0.4 -0.25]

P=convh(V); [VV,VF,nv,nf]=VVF(P,0.000001)

figure(1);clf;view2d(P,[1 2 3]);
```

Figure 3.39 displays the polytope considered. The results of the VVF routine in

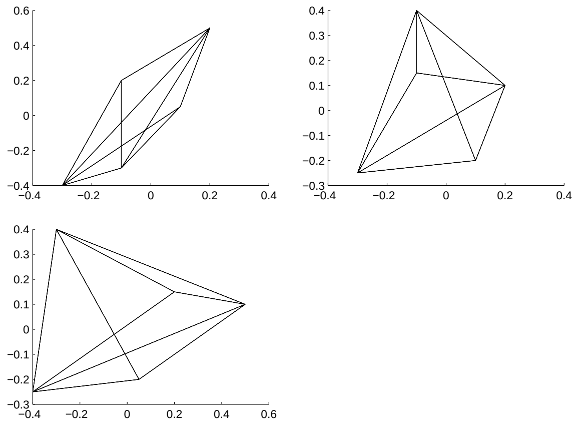

Figure 3.39: Wire-frame view of the polytope in the example for VVF.

this example are:

```
VV =
     2     3     4     0
     1     3     4     5
     1     2     4     5
     1     2     3     5
     2     3     4     0
VF =
     1     2     3     0
     2     3     5     6
     1     3     4     6
     1     2     4     5
     4     5     6     0
nv =
     5
nf =
     6
```

Here the matrix representation of the polytope P was:

```
P =
         0    6.0000    0.0000         0
   -0.9526    0.1361    0.2722    0.1633
   -0.2857    0.4286    0.8571    0.2429
   -0.6467    0.5605   -0.5174    0.0992
    0.7482   -0.6506   -0.1301    0.0683
    0.9433   -0.3018    0.1383    0.0516
   -0.5357    0.5476   -0.6428    0.1024
   -0.1000    0.2000    0.1500   -1.0000
    0.2000    0.5000    0.1000   -1.0000
   -0.3000   -0.4000   -0.2500   -1.0000
   -0.1000   -0.3000    0.4000   -1.0000
    0.1000    0.0500   -0.2000   -1.0000
```

3.2 Auxiliary Routines

COMBIN **COMBIN**

Purpose

Auxiliary routine to PROJPONT. COMBIN computes the binomial coefficients $\left(\begin{array}{c} n \\ k \end{array}\right)$, $n \geq k$.

Usage

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

where

```
C   -  matrix with a combination in each row
nc  -  the number of combinations = coefficient "n over k"
 n  -  positive integer
 k  -  nonnegative integer
```

FIRSTEQU FIRSTEQU

Purpose

Auxiliary routine to VREDUCE. Given a set of points in a matrix, it compares the first point (first row) with all other points (other rows) and it finds those which are less than δ distance from the first point in ℓ_2-sense. The δ-near points to the first are combined into a single point and the rest of the points are left unchanged.

Usage

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

where

```
  V    -  set of points
 del   -  tolerance
  v    -  combined point

  nV   -  rest of the points
```

Bibliography

[1] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The “quickhull” algorithm for convex hulls. *ACM Trans. on Mathematical Software*, 22:469–483, 1996.

[2] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry*, 8:295–313, 1992.

[3] D. Q. Mayne and W. R. Schroeder. Nonlinear control of constrained dynamic systems. *Int. J. Control*, 60:1035–1043, 1994.

[4] D. Q. Mayne and W. R. Schroeder. Robust time-optimal of constrained linear systems. *Automatica*, 33: , 1997.

[5] D. Q. Mayne, R. W. Grainger, and G. C. Goodwin. Nonlinear filters for linear signal models. *Proc. IEE, Series D.*, : , 1998.

[6] P. O. M. Scokaert and D. Q. Mayne. Min-max feedback control for constrained linear systems. *IEEE Trans. Automatic Control*, : , 1998.

[7] P. Caravani and E. De Sandtis. A polytopic game. *Automatica*, 36:973–981, 2000.

[8] V. A. Yemelichev, M. M. Kovalev, and M. K. Kravtsov. *Polytopes, Graphs and Optimization.* Cambridge University Press, Cambridge, 1984.

[9] J. P. Norton and S. H. Mo. Parameter bounding for time-varying systems. *Mathematics and Computers in Simulation*, 32:527–534, 1990.

[10] E. Walter and H. Piet-Lahanier. Exact recursive polyhedral description of the feasible parameter set for bounded-error models. *IEEE Trans. on Automatic Control*, AC-34:911–915, 1989.

[11] S. M. Veres. Limited complexity and parallel implementation of polytope updating. *Proc. American Control Conference, Chicago*, pages 1061–1062, 1992.

[12] M. Milanese, J. P. Norton, H. Piet-Lahanier, and E. Walter (Edts.). *Bounding Approaches to System Identification.* Plenum Press, New York, 1996.

[13] S. M. Veres, A. V. Kuntsevich, I. Vályi, S. Hermsmeyer D. S. Wall, and S. Sheng. Geometric Bounding Toolbox for MATLAB^{TM}, 1993-99. Version 6.1, MATLAB/Simulink Connections Catalogue, MathWorks Inc.

[14] The MathWorks Inc. *MATLAB - The Language of Technical Computing*. Version 5.2. The MathWorks, Natick, MA, 1984-1999.

[15] S. M. Veres. The format of a complete polytope description (type 6) in the Geometric Bounding Toolbox for MATLAB^{TM}, 1996. Research Memorandum 48, School of Electronic and Electrical Engineering, The University of Birmingham, Edgbaston, B15 2TT, UK.

[16] V. A. Yemelichev, M. M. Kovalev, and M. K. Kravtsov. *Polytopes, Graphs and Optimization*. Cambridge University Press, Cambridge, 1984.

[17] J. B. Lasserre. An analytical expression and algorithm for the volume of a convex polyhedron in $\Re^n$. *J. Optimiz. Theory Appl.*, 39:463–477, 1983.