# Sample routines from the Geometric Bounding Toolbox 7.3

Prof. Sandor M. Veres, sandy1@sysbrain.com, UK

### Purpose

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

### Usage

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

where the variables used are:

`V[0] - set of points`

### Brief Description of the Algorithm (not included)

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

### Example

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

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

### Detailed Description of the Algorithm (not included)

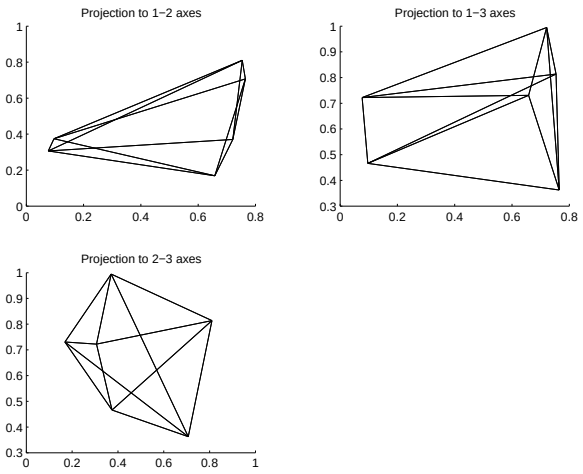


Figure 1: Demo figure to routine CONVH .

---

DEFBOX DEFBOX

---

### Purpose

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

### Usage

```
function P=defbox(vertex1,vertex2);
```

where the variables used are:

`vertex1[-1] - lowest position corner of the box`

### Brief Description of the Algorithm

The algorithm is based on building up lists of vertices in a way as binary number are build from 0 and 1, except these are replace by lower-corner and upper conrner components respectively. The number of vertices thus becomes $2^d$ where $d$ is the dimension of the space. The $2d$ facet ineqaalities are formed by unit vectors and components of the lower and upper corners given.

### Example

The following code defines a box with diagonal points [0.4,0.5,1] and [1.2,1.4,2] and displays its projected wireframe views:

```
P=defbox([0.4,0.5,1],[1.2,1.4,2]);view2d(P,[1 2 3]);% no.figs=1
```

**Detailed Description of the Algorithm (not included)**

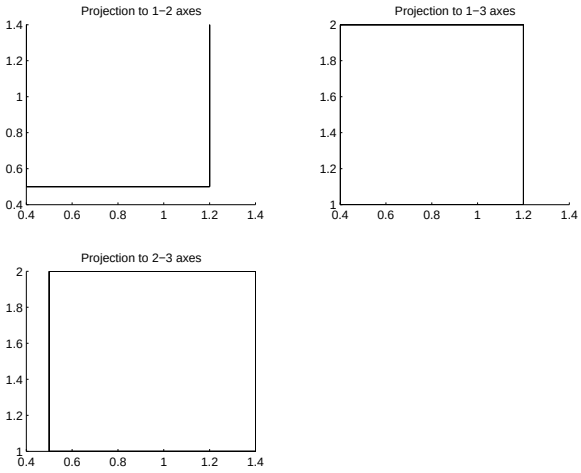


Figure 2: Demo figure to routine DEFBOX .

**Purpose**

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

**Usage**

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

where the variables used are:

`P[[1 eps;1 1]] - polytope`

**Brief Description of the Algorithm (not included)**

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

**Example**

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

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

**Detailed Description of the Algorithm (not included)**

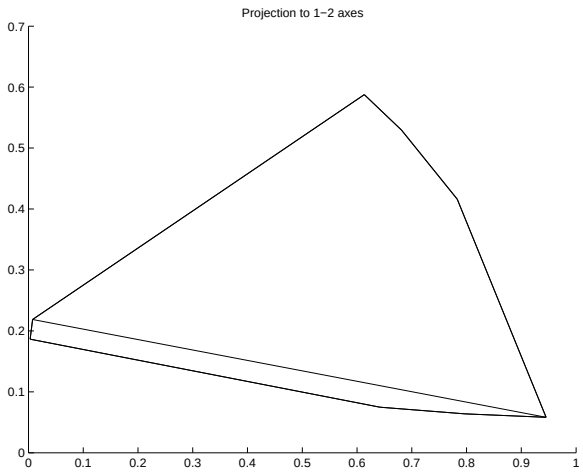


Figure 3: Demo figure to routine DIAMETER .

---

FACES FACES

---

**Purpose**

The purpose of **FACES** is to extract half-space inequalities of polytope.

**Usage**

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

where the variables used are:

`P[1 0;1 1] - polytope`

**Brief Description of the Algorithm**

The facet inequalities are simply extracted from the format of polytope representation.

**Example**

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

```
P=convh(rand(5,3));view2d(P,[1 2 3]);F=faces(P)
F =
-0.59161     0.44284    -0.67372    -0.15707
```

| | | | |
|---|---|---|---|
| -0.1548 | -0.089397 | -0.98389 | -0.365 |
| 0.54343 | 0.28325 | 0.79022 | 0.75074 |
| 0.53067 | 0.11588 | 0.83962 | 0.66544 |
| 0.089009 | -0.87972 | -0.46709 | -0.45808 |
| 0.07815 | -0.73161 | 0.67723 | -0.055499 |

**Detailed Description of the Algorithm (not included)**

### Purpose

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

### Usage

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

where the variables used are:

`H[[1 1;-1 1]] - list of half-spaces`

### Brief Description of the Algorithm

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

### Example

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

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

### Detailed Description of the Algorithm (not included)

### Purpose

The purpose of **FIRSTEQU** is to combine points with first one.

### Usage

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

where the variables used are:

```
V - list of  points
```

### Brief Description of the Algorithm

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

### Example

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

```
V=zeros(4,1); V(1,:)=[1,1];
V(2,:)=[1 2];V(3,:)=[1 1+1e-15];V(4,:)=[ 3 1];
[v,nV]=firstequ(V)
v =
0
nV =
```

### Detailed Description of the Algorithm (not included)

### Purpose

The purpose of **FV** is to produce facet-vertex Boolean table of adjacency.

### Usage

```
function [FV,nh,ne,acc]=fv(H);
```

where the variables used are:

`H[[0 eps;1 1;-1 1;-1 -1;1 -1]] - polytope`

### Brief Description of the Algorithm

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

### Example

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

```
P=convh(rand(5,3));fv(P)
fv(P) =
2  3  4
1  3  4
1  3  5
1  4  5
2  3  5
2  4  5
```

### Detailed Description of the Algorithm (not included)

**Purpose**

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

**Usage**

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

where the variables used are:

`P[[0 eps; 1 1; -1 1;-1 -1; 1 -1]] - polytope`

**Brief Description of the Algorithm**

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

**Example**

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

```
P=convh(rand(5,3));fvfitacc(P)
fvfitacc(P) =
2.2204e-016
```

**Detailed Description of the Algorithm (not included)**

**Purpose**

The purpose of **HADJUST** is to adjust a new hyperplane to a polytope.

**Usage**

```
function [h1,fitacc]=hadjust(h,P);
```

where the variables used are:

**Brief Description of the Algorithm**

The half-space inequality $h$ to update a polytope is adjusted so that first the DELTA-near vertices set $V1$ of the polytope is computed. The normal vector $n$ of $h$ is orthogonally projected onto the orthogonal complement of the manifold spanned by $V1$ to obtain a new normal $n1$. The adjusted inequality is then given by

$$h1 = [n1 \text{mean}(V1n^T1)]$$

**Example**

The following code generates a random polygon and modifies one facet inequality slightly to form a new half-space. This half-space is then updated with the polytope to illlustrate adjustment:

```
P=convh(rand(5,2));F=faces(P);h=F(1,:);
DELTA=1e-9;h2=h+rand(1,3)*1e-11;h1=hadjust(h2,P);
h_h1=norm(h-h1)
h_h1 =
0
```

**Detailed Description of the Algorithm (not included)**

### Purpose

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

### Usage

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

where the variables used are:

`G1[[1 eps;1 1]] - first polytope`

### Brief Description of the Algorithm

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

### Example

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

```
P1=convh(rand(10,3));P2=convh(rand(16,3)-0.1);
P=intersct(P1,P2);ax=view2d(P1,[1 2 3]); % no.figs=2
view2d(P2,[1 2 3],'b',ax);figure;view2d(P,[1 2 3]);
```

### Detailed Description of the Algorithm (not included)

Figure 4: Demo figure to routine INTERSCT .

Figure 5: Demo figure to routine INTERSCT .

---

INTEST INTEST

---

**Purpose**

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

**Usage**

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

where the variables used are:

```
V[0] - array of points
```

**Brief Description of the Algorithm**

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

**Example**

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

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

**Detailed Description of the Algorithm (not included)**

**Purpose**

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

**Usage**

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

where the variables used are:

`P[[0 eps;1 1;-1 1;-1 -1; 1 -1]] - polytope`

**Brief Description of the Algorithm**

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

**Example**

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

```
x=rand(1,2);P=defbox(x,x+[0.5,0.3]);
A=rand(2,2)
A =
0.65011     0.55267
0.98299     0.40007
a=rand(2,1)
a =
0.73336
0.37589
P1=ltran(P,A,a);
ax=view2d(P);view2d(P1,[1 2],'b',ax);
```

**Detailed Description of the Algorithm (not included)**

### Purpose

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

### Usage

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

where the variables used are:

```
V - set of points
```

### Brief Description of the Algorithm

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

### Example

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

```
V=rand(3,3)
V =
0.19389     0.63179     0.93158
0.90481     0.23441      0.3352
0.56921     0.54878     0.65553
[a,b,fitacc]=planfit(V)
a =
0.53823    -0.24662     0.80591
b =
0.69932
fitacc =
1.1102e-016
```

### Detailed Description of the Algorithm (not included)

**Purpose**

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

**Usage**

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

where the variables used are:

`O1[[1 eps;1 0]] - first polytope`

**Brief Description of the Algorithm**

The algorithm first lists the vertices of each polytope. All combinations of sums of vertices from the first and second polytope are listed in a set of points $V1+2$. The convex hull of $V1+2$ is computed to obtain the polytope representing the Minkowski sum.

**Example**

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

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

**Detailed Description of the Algorithm (not included)**

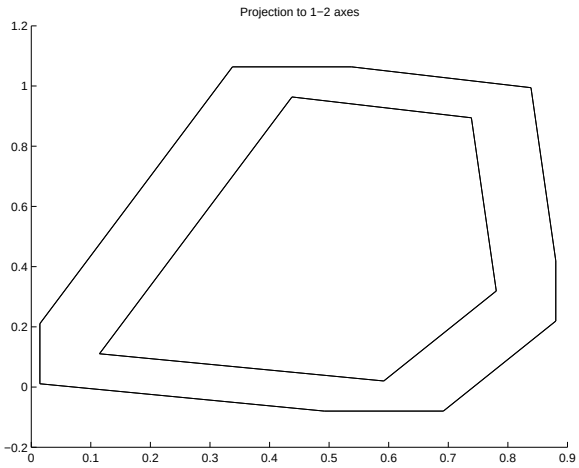


Figure 6: Demo figure to routine POLADD .

---

POLDUAL POLDUAL

---

**Purpose**

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

**Usage**

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

where the variables used are:

`P[[0 eps;1 1;-1 1;-1 -1; 1 -1]] - polytope`

**Brief Description of the Algorithm**

The dual of a polytope $P \subset \Re^d$ is defined by $Pdual = \{x \in \Re^d \mid x^T y \leq 1 \, \forall y \in P\}$ . The dual of a polytope $P$ is only defined if the origin is contained in $P$, otherwise the output 'D' is an empty matrix. If the vertices of polytope $P$ are $yk, k \in K$ then the dual can be computed as the intersection of half-spaces

$$\cap k \in K\{x \in \Re^d \mid x^T yk, \; k \in K\}$$

This fact is used to define the facet list if the dual polytope. If $h^T x \leq h1$ is a facet-inequality than $h/(h1)^{-1}$ is a vertex of the dual polytope. This fact is used to list the set of vertices of the dual polytope. The worst-case facet-vertex fitting accuracy of the dual polytope is re-computed by the 'fv' routine.

**Example**

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

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

**Detailed Description of the Algorithm (not included)**

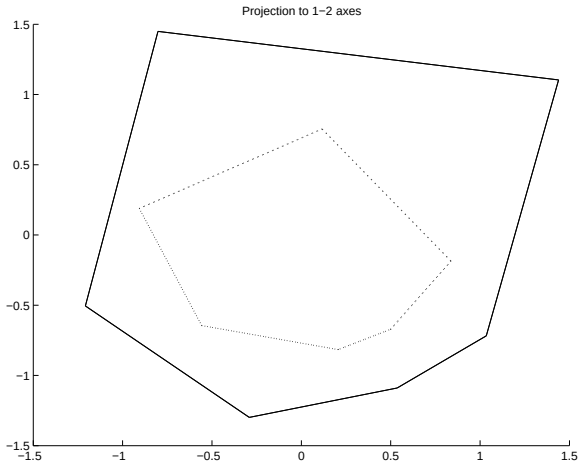


Figure 7: Demo figure to routine POLDUAL .

**Purpose**

The purpose of **P_CONV** is to compute point inclusion into convex hull.

**Usage**

```
function [0,acc]=p_conv(xn,P);
```

where the variables used are:

`xn[0] - point`

**Brief Description of the Algorithm**

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

**Example**

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

```
V=rand(6,2);P=convh(V);P1=p_conv([1 1],P); % no.figs=1
figure;a=view2d(P);view2d(P1,[1 2],'r',a);
```

**Detailed Description of the Algorithm (not included)**

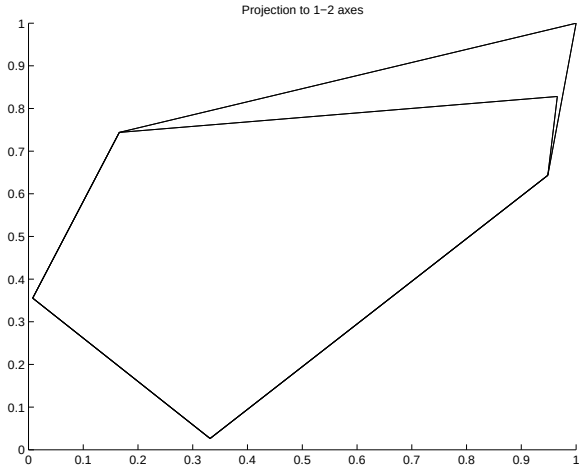


Figure 8: Demo figure to routine P_CONV .

**Purpose**

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

**Usage**

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

where the variables used are:

`A[[1 eps;1 0]] - first polytope`

**Brief Description of the Algorithm**

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

$$h^T i \leq ci - \max k \in K(h^T ivk),\ i \in I$$

where $h^T ix \leq ci,\ i \in I$ are all the half-space inequalities defining polytope $A$. The routine 'fconvh' is used to compute the intersection of the new set of half-space inequalities.

**Example**

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

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

**Detailed Description of the Algorithm (not included)**

Figure 9: Demo figure to routine SUBTRACT .

---

UNI UNI

---

**Purpose**

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

**Usage**

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

where the variables used are:

`U1[1] - matrix of row vectors`

**Brief Description of the Algorithm**

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

**Example**

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

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

```
2  1
1  2
1  1
```

**Detailed Description of the Algorithm (not included)**

### Purpose

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

### Usage

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

where the variables used are:

`h[[1 0]] - half-space inequality`

### Brief Description of the Algorithm

This is a direct algorithm to obtain the intersection of a half-space and a polytope. If no vertex of 'P' falls DELTA-outside of 'h' then the intersection of 'P' and 'h' remains 'P'. Otherwise it is tested by the 'hadjust' routine whether the new hyperplane 'h' has to be aligned with some vertices of polytope 'P'. Then the vertices of 'P' inside, outside and on the boundary of the half-space defined by 'h' are found. New vertices are only computed by 'vtxfit' if two vertices are connected by an edge and one is inside, the other is outside 'h'. The facets, which are fully on or outside the intersection of 'P' and 'h', are erased. The facet-inequality 'h' is added to the list, vertices outside 'P' are erased from the vertex list.

### Example

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

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

### Detailed Description of the Algorithm (not included)

**Purpose**

**Detailed Description of the Algorithm (not included)**

**Purpose**

The purpose of **VERTICES** is to produce list of vertices of polytope.

**Usage**

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

where the variables used are:

`P[[1 eps;1 0]] - polytope`

**Brief Description of the Algorithm**

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

**Example**

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

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

**Detailed Description of the Algorithm (not included)**

**Purpose**

**Detailed Description of the Algorithm (not included)**

**Purpose**

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

**Usage**

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

where the variables used are:

`P - polytope`

**Brief Description of the Algorithm**

The plot of the 3D view is based on creating a patch object of faces of the polytope. If 'axin, style and pcolor' are defined then reasonable defaults are taken. The object handles of the axes(axout), patch(p) and light(ligh) appear as outputs of the routine so that they can be used altered or manipulated by the user. See the MATLAB manual on graphics and patch objects.

**Example**

The following code generates a random 3D polytope and displays its perspective view:

```
P=convh(rand(7,3));view3d(P);%no.figs=1
```

**Detailed Description of the Algorithm (not included)**

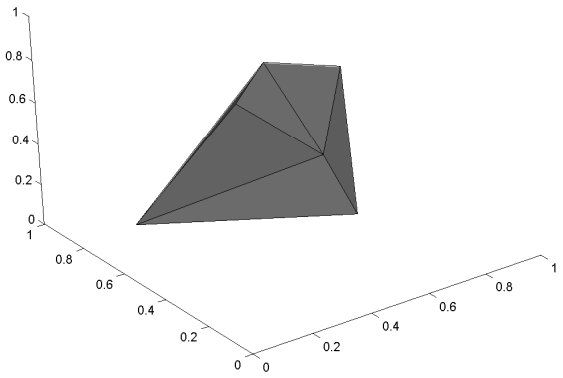


Figure 10: Demo figure to routine VIEW3D .

---

VREDUCE VREDUCE

---

**Purpose**

The purpose of **VREDUCE** is to eliminate multiple points from array of points.

**Usage**

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

where the variables used are:

`V - array of points`

**Brief Description of the Algorithm**

This algorithm is based on repeated application of the routine 'firstequ' to obtain a reduced set of points by combining the DELTA near ones. Note that the result may depend on the order of the points listed.

**Example**

The following code generates 1000 points in a 4D hypercube of edge size 10*DELTA and reduces the number of points by making the DELTA-near ones equal:

```
DELTA=1e-10;V=6*DELTA*(rand(200,4)-0.5);
nV=vreduce(V);no_of_new_points=size(nV,1)
```

```
no_of_new_points =
152
```

**Detailed Description of the Algorithm (not included)**

**Purpose**

**Detailed Description of the Algorithm (not included)**

### Purpose

The purpose of **VVF** is to produce a vertex-vertex and vertex-facet adjacency tables.

### Usage

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

where the variables used are:

`H - polytope`

### Brief Description of the Algorithm

In the algorithm all facet-inequalities ar tested against all vertices to see whether they fit within a DELTA error bound. To obtain 'VV', for each vertex the indices of all those vertices are listed which share $d-1$ adjacent facets. To obtain 'VF', for each vertex the indices of all adjacent facets are listed. The maximum length of lists is monitored and this determines the number of columns in 'VV' or in 'VF'. If a list for a vertex is shorter than the number of columns then the rest of the entries is filled with zeros.

### Example

The following code generates a random 3D polytope and displays its vertex-vertex and vertex-facet adjacency tables:

```
P=convh(rand(5,3));view2d(P,[1 2 3]);
[VV,VF]=vvf(P)
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  4  5
1  3  4  6
1  2  5  6
4  5  6  0
```

### Detailed Description of the Algorithm (not included)

**Purpose**

**Detailed Description of the Algorithm (not included)**