Multiparametric Toolbox 3.0
Function Reference

December 19, 2013

Contents

3

Function Reference

MPT_INIT

Initializes MPT toolbox for the first time after installation.

SYNTAX

```
mpt_init
```

DESCRIPTION

The function `mpt_init` sets default options for MPT toolbox and adds all required directories to Matlab path. It is recommended to store the Matlab path for future such that this script does not need to be run again. The options will be stored internally by Matlab and it is no longer needed to initiaze them using `mpt_init` function everytime the toolbox is started. The options can be later changed using `mptopt` function. Note that any changes in the options will be replaced by default options whenever `mpt_init` is invoked.

SEE ALSO

mptopt

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPTOPT

Global option handler for MPT.

SYNTAX

```
mptopt('Parameter1',Value1,'Parameter2',Value2,...)
```

DESCRIPTION

The function/class `mptopt` plays the role of global option handler throughout MPT. Any specific option should be changed using `mptopt` function. The syntax follows the "parameter-value" scheme where the `parameter` is the desired option and `value` is the assignment to this option. The changes take effect immediatelly and are valid globally through the whole MPT toolbox. Global options are kept in memory after initialization and any changes in the option are preserved between sessions.

INPUT

Parameter1	The name of the desired option to be changed provided as string. The list of available options can be obtained by typing `properties('mptopt')` at the Matlab prompt. Class: `char`
Value1	The value to be assigned to `Parameter1`. Class: `double or char`

EXAMPLE(s)

Example 1

This example shows how to change LP solver to "CDD".
To list which solvers are installed, invoke `solvers_list` option from `mptopt` class.
This is achieved by assigning the output from `mptopt` class to some variable and referring to `solvers_list` field as follows:

```
s = mptopt
```

```
s =

Global settings for MPT:
version: @version@
solvers_list: [struct]
rel_tol: 1e-06
abs_tol: 1e-08
lex_tol: 1e-10
zero_tol: 1e-12
region_tol: 1e-07
report_period: 2
```

```
verbose: 0
infbound: 10000
colormap: matrix of size [10 x 3]
lpsolver: LCP
qpsolver: LCP
milpsolver: GLPK
miqpsolver:
lcpsolver: LCP
plpsolver: PLCP
pqpsolver: PLCP
plcpsolver: PLCP
modules: [struct]
```

```
s.solvers_list.LP
```

```
ans =

'LCP'
'CDD'
'GLPK'
'LINPROG'
'QPOASES'
'CLP'
'SEDUMI'
```

If CDD solver is present in the list, it means that it exist on the Matlab path. To change CDD solver as default for solving LP, it can be done twofold.
Assignment with the help of the output variable `s`:

```
s.lpsolver = 'CDD';
```

or calling `mptopt` via standard syntax

```
mptopt('lpsolver','CDD');
```

Example 2

Options for any module are stored here as well. For instance, the geometry module of MPT that contains operations on polyhedra has the options stored under

```
s = mptopt;
```

```
s.modules.geometry
```

```
ans =

sets: [1x1 struct]
unions: [1x1 struct]
```

These options can be changed from any instance of the script and take effect globally. To reset to default options use the function

```
mpt_init
```

```
MPT searches for solvers on the path ...

LINPROG ......................................... linprog.m
QUADPROG ........................................ quadprog.m
GLPK ................................................. glpkcc
CDD .................................................. cddmex
CLP ................................................... mexclp
QPOASES ............................................ qpOASES
LCP ...................................................... lcp
SEDUMI ............................................ sedumi.m
QPSPLINE ........................................ QPspline.m
PLCP ............................................... mpt_plcp.m
MPQP ............................................... mpt_mpqp.m
MPLP ............................................... mpt_mplp.m

MPT toolbox @version@ initialized...
Copyright (C) 2003-2013 by M. Kvasnica,C.N. Jones,and M. Herceg
For news,visit the MPT web page at http://control.ee.ethz.ch/~mpt/
LP solver: LCP
QP solver: LCP
MILP solver: GLPK
parametric LP solver: PLCP
parametric QP solver: PLCP

These default options can be changed. See "help mptopt" for more details.
```

SEE ALSO

mpt_init

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPTDOC

Display documentation for Multi-Parametric Toolbox in Matlab help browser.

SYNTAX

```
mptdoc
```

DESCRIPTION

The function `mptdoc` opens the documentation for Multi-Parametric Toolbox inside the Matlab help browser.

SEE ALSO

mpt_init, mptopt

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

MPT_SCALE_MATRIX

Scales matrix row-wise and column-wise

SYNTAX

`[An,D1,D2] = mpt_scale_matrix(A)`

DESCRIPTION

Scales matrix A by finding diagonal matrices D_1 and D_2 in $A_n = D_1AD_2$ such that infinity norm of each row and column approaches 1.
The problem is given as

$$\min \quad \|D_1AD_2\|)$$
$$\text{s.t.} \quad A_n = D_1AD_2$$

Scaling matrix is used in solving linear equations of the type $Ax = b$ for badly scaled matrix A as follows:

$$
\begin{aligned}
Ax &= b \\
D_1Ax &= D_1b &&\text{multiply from left by } D_1 \\
D_1AD_2D_2^{-1}x &= D_1b &&\text{insert } D_2D_2^{-1} \\
D_1AD_2y &= D_1b &&\text{substitute } y = D_2^{-1}x \\
A_ny &= b_n &&\text{substitute } A_n = D_1AD_2,\ b_n = D_1b
\end{aligned}
$$

First solve $A_ny = b_n$, then obtain $x = D_2y$.

INPUT

A   Input matrix do be scaled. The matrix can be also rectangular.
Class: `double`

OUTPUT

An   Scaled matrix A such that $A_n = D_1AD_2$.
Class: `double`

D1   Diagonal matrix D_1 such that $A_n = D_1AD_2$.
Class: `double`

D2   Diagonal matrix D_2 such that $A_n = D_1AD_2$.
Class: `double`

SEE ALSO

mptopt, mpt_solve

LITERATURE

Details of the method are in file drRAL2001034.ps.gz
http://www.numerical.rl.ac.uk/reports/reports.html

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_INEQ2EQ

Detects inequality constraints which form equalities

SYNTAX

`[An, bn, Ae, be, ind_e] = mpt_ineq2eq(A, b, tol)`

DESCRIPTION

For a system of inequalities $Ax \leq b$, this function detects and returns those inequality constraints which form equalities. For instance: `A = [1; -1; 1]; b = [1; -1; 2];` The output will lead: `An = [-1]; bn = [2]; Ae = [1]; be = 1;` such that the original problem can be rewritten as:

$$
\begin{aligned}
A_n x &\leq b_n \\
A_e x &= b_e
\end{aligned}
$$

The algorithm works up to specified tolerance `tol`.

INPUT

A   Matrix of inequality constraints in $Ax \leq b$
     Class: `double`

b   Right hand side of inequalities in $Ax \leq b$
     Class: `double`

tol Working precision of the algorithm.
     Class: `double`
     Default: `MPTOPTIONS.abs_tol`

OUTPUT

An   Matrix of new inequality constraints $A_n x \leq b_n$
       Class: `double`

bn   Right hand side of new inequality constraints in $A_n x \leq b_n$
       Class: `double`

Ae   Matrix of equality constraints $A_e x = b_e$
       Class: `double`

be   Right hand side of equality constraints in $A_e x = b_e$
       Class: `double`

ind_e Rows of A, b that create a pair of double sided inequalities, sorted in columns.

Class: `double`

EXAMPLE(s)

Example 1

A system of inequalities $x \leq 1$, $-x \leq -1$, $x \leq 2$ contains one equality constraint $x = 1$ written as double-sided inequality.
The corresponding matrix form of inequalities $Ax \leq b$ is built by

`A = [1; -1; 1];`

`b = [1; -1; 2];`

To detect the equality, we use `mpt_ineq2eq` function

`[An,bn,Ae,be] = mpt_ineq2eq(A,b)`

```
An =

1


bn =

2


Ae =

1


be =

1
```

SEE ALSO

Polyhedron

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
        mailto:michal.kvasnica@stuba.sk

© **2006** Johan Loefberg: ETH Zurich
        mailto:loefberg@control.ee.ethz.ch

FOREACH

Applies a given function to each element of in an array of sets.

SYNTAX

```
A = obj.forEach(@myfun)

A = obj.forEach(@(x) myfun(x))

A = obj.forEach(@(x) x.myfun)
```

DESCRIPTION

The method `forEach` is an overloaded method of standard `arrayfun` or `cellfun` that applies to arrays of sets. To apply method MYFUN to all elements of the array of objects OBJ, you can use any of the following ways:

- `obj.forEach(@myfun)`

- `obj.forEach(@(x) myfun(x))`

- `obj.forEach(@(x) x.myfun)`

Note that `forEach` will not return any outputs if not asked to. Outputs can be requested as well:

```
[A, B, C] = obj.forEach(@myfun)
```
If MYFUN generates all its outputs as scalars, then A, B, and C will be arrays with as many elements as there are elements in the array OBJ.
If outputs generated by MYFUN have different sizes for different elements of OBJ, then you must set the "UniformOutput" option to false:

```
[A, B, C] = obj.forEach(@myfun, 'UniformOutput', false)
```
In this case A, B, and C will be cell arrays, with Ai, Bi and Ci containing the output of MYFUN applied to the i-th element of OBJ.
In short, the `forEach` method works exactly as `arrayfun` and `cellfun`, hence see "help arrayfun" for a more throught description.

INPUT

@myfun A handle to a function that is to be applied to each
        element in the array of sets for object `obj`.
        Class: `function_handle`

OUTPUT

> **A** Array or cell array of outputs returned by function
> `@myfun` that correspond to individual outputs from the
> array of sets `obj`.
> Class:

EXAMPLE(s)

Example 1

Create array of 5 random polyhedra

```
for i=1:5; P(i)=ExamplePoly.randVrep; end
```

For each polyhedron in the array execute the function `outerApprox`

```
B = P.forEach(@outerApprox)
```

```
Array of 5 polyhedra.
```

In B we have computed the outer bounding boxes for each polyhedron in the array. It is the same results if asked

```
Bnew = P.outerApprox
```

```
Array of 5 polyhedra.
```

Example 2

Create two unions of polyhedra in different dimensions

```
U(1) = PolyUnion([ExamplePoly.randVrep;ExamplePoly.randVrep]);
```

```
U(2) = PolyUnion([ExamplePoly.randVrep('d',3);ExamplePoly.randVrep('d',3)]);
```

For each polyunion in the array extract the convexity property that is computed by `isConvex` method.

```
ts = U.forEach(@isConvex)
```

```
ts =

0    0
```

The same result can be obtained by

```
tn = U.isConvex
```

```
tn =

0    0
```

SEE ALSO

Polyhedron, ConvexSet

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

ITERABLEBEHAVIOR

Basic class which provides `forEach` method to iterate over an array.

SYNTAX

```
S = IterableBehavior ( arg )
```

DESCRIPTION

The `IterableBehavior` class has been introduced to deal with arrays of sets. The only purpose of this class is to have defined `forEach` method that executes a particular method for each element in the array. The `IterableBehavior` class cannot be instantiated or subclasses.

INPUT

arg   Any input argument is accepted but has no influence on
the object creation.
Class:

OUTPUT

S   Object of the `IterableBehavior` class.
Class: `IterableBehavior`

SEE ALSO

Polyhedron, ConvexSet

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

TOC

Export of PWA/PWQ function to C-code

SYNTAX

```
controller.toC('function')
```

```
controller.toC('function','filename')
```

DESCRIPTION

The function `toC()` exports given piecewise affine (PWA) or piecewise quadratic (PWQ) function to C-language including a binary search routine. The PWA/PWQ function must be attached to the `BinTreePolyUnion` object.

If the file name is not provided, the default output name is `mpt_getInput`.

The export routine generates two files on the output:

- `mpt_getInput.c` - which contains the PWA/PWQ function including the binary search

- `mpt_getInput_mex.c` - mex interface for evaluation in Matlab

The file `mpt_getInput_mex` can be compiled inside Matlab and used for fast evaluation of PWA/PWQ function. The compilation is invoked by `mex` routine as follows:

```
mex mpt_getInput_mex
```

The function `toC()` can export the floating point numbers to single or double precision. The default setting is `double` but this can be modified in global options

```
modules.geometry.unions.BinTreePolyUnion.toC.
```

INPUT

function	Name of the attached PWA/PWQ function to export. Class: `char`
filename	Base name of the file to be generated. Class: `char`

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

HASFUNCTION

Returns true if the union contains given function names.

SYNTAX

```
ts = hasFunction(U, FuncName)
```

```
ts = U.hasFunction(FuncName)
```

DESCRIPTION

Returns true or false if the function stored under `FuncName` is attached to union `U`.

INPUT

U	Union of sets derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `Union`
FuncName	Name of the function to test. Class: `char`

OUTPUT

ts	Logical statement if the set contains the function `FuncName` or not. Class: `logical`

SEE ALSO

listFunctions, addFunction, removeFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

GETFUNCTION

Extract function handle from the union of convex sets.

SYNTAX

```
F = getFunction (U, FuncName )
```

```
F = U. getFunction ( FuncName )
```

DESCRIPTION

Extract `Function` object that is attached to a union `U` under the name `FuncName`.

INPUT

U	Union of sets derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `Union`
FuncName	Name of the function to extract. Class: `char`

OUTPUT

F	Function object stored under the `FuncName` string. Class: `Function`

SEE ALSO

hasFunction, addFunction, removeFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

VERTCAT

Vertical concatenation for union objecs.

SYNTAX

```
U = [U1; U2]

S = vertcat(U1,U2)
```

DESCRIPTION

Overloaded method for vertical concatenation of unions. It is not possible to concatenate objects of different type to the same array (e.g. `PolyUnion` and `Union`). Similarly, it is not possible to concatenate into matrices, only vectors are allowed.

INPUT

U1   Object of the `Union` class.
Class: `Union`

U2   Object of the `Union` class.
Class: `Union`

OUTPUT

U   The array of the `Union` objects.
Class: `Union`

EXAMPLE(s)

Example 1

We have two polyhedra in `U1` object.

```
P(1) = ExamplePoly.randHrep;

P(2) = ExamplePoly.randHrep;

U1 = Union(P);
```

The object `U2` contains one polyhedron.

```
U2 = Union(ExamplePoly.randHrep)
```

```
Union of 1 convex sets.
Functions : none
```

Concatenate unions vertically

23

```
U = [U1; U2]
```

Array of 2 Unions.

SEE ALSO

horzcat

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

REMOVEFUNCTION

Remove function from all sets in the union based on the function name.

SYNTAX

```
U = removeFunction(U,name)
```

```
U.removeFunction(name)
```

DESCRIPTION

Removes `Function` object from the union of convex sets identified by the string `name`. The functions stored with the union can be retrieved using `listFunctions` method. The name of the function must match with one of the names stored in the array.

INPUT

U	Object of the `Union` class that holds objects derived from the `ConvexSet` class. Class: `Union`
name	Name of the function to remove from the union. String must match one of the names as retrieved by `listFunctions` method. For multiple names, provide `pos` as a cell array of strings. Class: `char`

OUTPUT

U	Modified object of the `Union` class without the function handles that has been removed. Class: `PolyUnion`

EXAMPLE(s)

Example 1

Create an union of two Yalmip sets that two functions.
Define sets in 2D

```
x = sdpvar(2,1);
```

```
F1 = [ [1,-2; 0.4,5]*x<= [1;2]];
```

```
F2 = [ [5,-3; 1.3,-2; 0.8 -9; 1 -0.3]*x<= [1;2;1;1.5]];
```

```
Y(1) = YSet(x,F1);
```

```
Y(2) = YSet(x,F2);
```

Define the functions

```
f1 = AffFunction([0.1,-3],2);
```

```
f2 = AffFunction([-2,0.5],0.4);
```

Add these functions to the sets

```
Y.addFunction(f1,'f1');
```

```
Y.addFunction(f2,'f2');
```

Create the `Union` object.

```
U = Union(Y)
```

```
Union of 2 convex sets.
Functions : 2 attached "f1","f2"
```

Remove the function "f2" from the set

```
U.removeFunction('f2')
```

```
Union of 2 convex sets.
Functions : 1 attached "f1"
```

Union of sets now contains only 'f1' function

```
U.listFunctions
```

```
ans =
```

```
'f1'
```

SEE ALSO

addFunction, removeAllFunctions

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

26

DISPLAY

Overload display for `Union` class.

SYNTAX

```
display(U)
```

```
U.display
```

DESCRIPTION

Default display for `Union` class.

INPUT

U   `Union` object.
    Class: `Union`

SEE ALSO

PolyUnion

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

FPLOT

Plot single function over the sets of the Union object.

SYNTAX

```
h = Set.fplot()

h = Set.fplot('name', 'Prop1', value1, 'Prop2', value2)

h = fplot(Set, 'name', 'Prop1', value1, 'Prop2', value2)
```

DESCRIPTION

Plot single function over an union of convex sets. If there are more functions attached to a set, then the string `name` identifies the function to be plotted. If the function is vector valued, i.e. its range is greater than 1, than the first element of the function is plotted by default. For vector valued functions, use the `position` property to indicate that you want a different element of the function value to plot.

Figure properties, such as color, line width, etc, can be specified with "Property" - "Value" pairs.

INPUT

U   Union object that contains sets derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ...
Class: `Union`

name   If there are more functions attached to the set, this string indicates the `name` of the function to plot.
Class: `char`

Prop1   Specification of figure properties.
Class: `char`

Allowed values: `position` For vector valued functions, the `position` indicates which element of the function value to plot.

`Grid` With how many gridpoints to grid the circle/-sphere. Default is 20.

`Color` The color of the plot specified by real RGB vector or a string name of the color (e.g. 'gray');

`Wire` Highlight the edges of the set. Default is false.

`LineStyle` Specify the type of the line to plot edges of the set. Accepted values are '-',':','-.','–', and 'none'.

`LineWidth` Specify the width of the line. Default is 1.

`Alpha` Transparency of the color. The value must be inside [0,1] interval. Default is 1.

`Contour` Add contour graph. Default is false.

`ContourGrid` With how many grid points to plot the contour graph. Default is 30.

`show_set` Plot the domain of the function. Default is false.

`showgrid` Show the grid inside the set. Default is false.

`colormap` Color map to use given as a string or a function handle. Default is 'mpt'.

`colororder` Either 'fixed' or 'random'. Default is 'fixed'.

`value1` Assigns value to `Prop1`.

OUTPUT

`h` Handle related to graphics object.
Class: `handle`

EXAMPLE(s)

Example 1

We have two linear functions "alpha", "beta" over a union of convex sets. Construct the set first set as the circle

```
x = sdpvar(2,1);

F = [0.5*x'*x<=0.2];

S = YSet(x,F);
```

Construct quadratic and linear function.

```
Q = QuadFunction(eye(2),[0,1]);

L = AffFunction([-3,1],1);
```

Add functions to the set with names "alpha" and "beta".

29

```
S.addFunction(Q,'alpha');
```

```
S.addFunction(L,'beta');
```

Construct a hyperplane that separates the set `S` and the point `x=[0;1]`.

```
x = [0;1]; h = S.separate(x);
```

Construct a polyhedron out of the hyperplane.

```
P = Polyhedron('He',h,'lb',[-1;-1],'ub',[1;1]);
```

Add functions "alpha" and "beta" to the polyhedron

```
P.addFunction(Q,'alpha');
```

```
P.addFunction(L,'beta');
```

Create union object from the sets `S` and `P`.

```
U = Union(S); U.add(P)
```

```
Union of 2 convex sets.
Functions : 2 attached "alpha","beta"
```

Plot the function "beta" over the union because this is contained in both sets

```
U.fplot('beta');
```
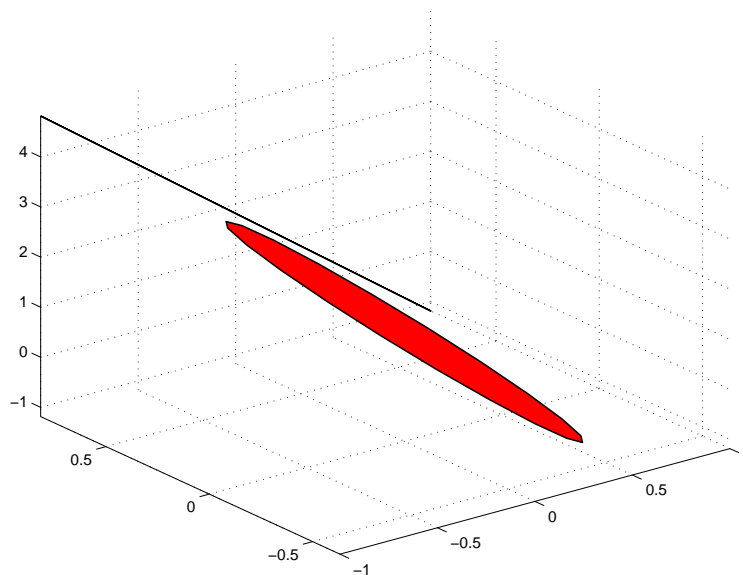

SEE ALSO

plot

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

PLOT

Plot the union of convex sets.

SYNTAX

```
h = plot(U, 'Prop1', value1, 'Prop2', value2)

h = U.plot('Prop1', value1, 'Prop2', value2)

h = plot(U1, 'Prop1', value1, ..., U2, 'Prop2', value2, ...)
```

DESCRIPTION

Plot the union of general convex sets up to dimension three.
Figure properties, such as color, line width, etc, can be specified with "Property" - "Value" pairs.

INPUT

U	Union object that contains sets derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `Union`
Prop1	Specification of figure properties. Class: `char`

Allowed values: `Grid` With how many gridpoints to grid the circle/-sphere for `YSet` objects. Default is 40.

`Color` The color of the plot specified by real RGB vector or a string name of the color (e.g. 'gray');

`Wire` Highlight or not the edges of the set. Default is false.

`LineStyle` Specify the type of the line to plot edges of the set. Accepted values are '-',':','-.','--', and'none'.

`LineWidth` Specify the width of the line. Default is 1.

`Alpha` Transparency of the color. The value must be inside [0,1] interval. Default is 1.

`Marker` Type of markings to use. Allowed values are ".", "o", "x", "+", "*", "s", "d", "v", "^", "¡", "¿", "p", "h" or "none". Default is "none".

`MarkerSize` The size of the marker. Default is 6.

`ColorMap` Color map given either as a M-by-3 matrix, or as a string. Default is 'mpt'. Other available options are 'hsv', 'hot', 'gray', 'lightgray', 'bone', 'copper', 'pink', 'white', 'flag', 'lines', 'colorcube', 'vga', 'jet', 'prism', 'cool', 'autumn', 'spring', 'winter', 'summer'.

`ColorOrder` Either 'fixed' for fixed ordering of colors, or 'random' for a random order. Default is 'fixed'.

`ShowIndex` This option is valid only for bounded polyhedra in 2D. If true, display an index of the plotted element. The default choice is false.

value1   Corresponding value to `Prop1`.

OUTPUT

h   Handle related to graphics object.
Class: `handle`

EXAMPLE(s)

Example 1

Plot union of two sets in 2D.
We have a half circle describe the the following inequalities

```
x = sdpvar(2,1);
```

```
F = [x(1)<=1; x(2)>=1; 0.2*x'*x <= 0.9];
```

Construct the set

```
S = YSet(x,F);
```

Construct outer approximation of the set `S`

```
B = S.outerApprox;
```

Store the sets inside the union object

```
U = Union(S); U.add(B)
```

```
Union of 2 convex sets.
Functions : none
```

Plot the sets in the Union object with the 0.4 transparency.

```
U.plot('Alpha',0.4);
```

```
Plotting...
30 of 40
```
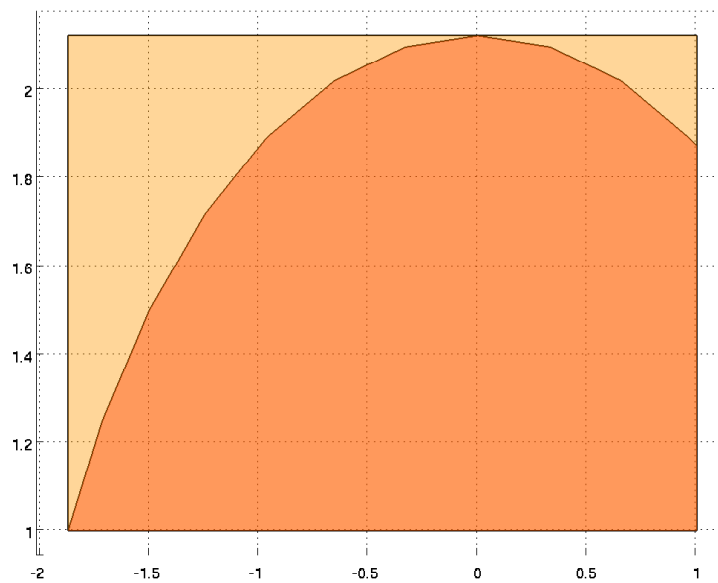

SEE ALSO

fplot

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

FEVAL

Evaluates a given function defined over a union of convex sets.

SYNTAX

```
fval = U.feval(x)

fval = U.feval(x, function_name)

[fval, feasible, idx, tb_value] = U.feval(x, function_name)

[fval, feasible, idx, tb_value] = feval(U, x, function_name)
```

DESCRIPTION

Evaluates function for a given value of the point `x` over the union of convex sets `U` characterized by the name `function_name`. If the string `function_name` is omitted, it is assumed that only one function is attached to the union. The dimension of the vector `x` must be the same as the dimension of all sets in the union. The evaluation is based on the following approach:

1. Find the index of the set where the point `x` lies.

2. Evaluate the function over the set determined by given index.

If the point lies outside of the union, the result is `NaN`.
Notes:

- `U` must be a single union. Arrays of unions are not accepted. Use `array.forEach(@(e) e.feval(...))` to evaluate arrays of unions.

- `function_name` must refer to a single function. If omitted, `U.feval(x)` only works if the union has a single function.

Outputs

1. `x` is not contained in any set of the union:

   `fval` $= (m \times 1)$ vector of `NaNs`, where m is the range of the function, `feasible = false`, `idx = []`, `tb_value = []`.

2. `x` is in a single set:

   `fval` $= (m \times 1)$ vector of function values, `feasible = true`, `idx` $=$ index of the set which contains `x`, `tb_value = []`.

3. `x` is contained in multiple sets (either at the boundary or in strict interior if there are overlaps), no tie-breaking (default):

   `fval` $= (m \times j)$ matrix of function values (j denotes the number of sets which contain x), each column contains the value of `function_name` in the corresponding set, `feasible = true`, `idx` $= (1 \times j)$ vector of indices of sets which contain `x`, `tb_value = []`.

35

4. x is contained in multiple sets (either at the boundary or in strict interior if there are overlaps), tie-breaking enabled (see below):

    fval $= (m \times 1)$ vector containing the function value in the set in which value of the tie-breaking function is smallest (if there are multiple sets with the same tie-breaking value, the first such set is considered), feasible = true, idx = index of the set which contains x and, simultaneously, has the **smallest** value of the tie-breaking function, tb_value = scalar value of the tie-breaking function in set indexed by idx.

Tie-breaking

The purpose of tie-breaking is to automatically resolve situations where the evaluation point x is contained in multiple sets. With tie-breaking enabled Union/feval() evaluates the tie-breaking function to decide which set containing x should be used for evaluation of the original function.

The tie-breaking function can be specified by U.feval(x, 'tiebreak', tb_fun), where tb_fun can be either a string or a function handle. A string value must refer to an another function which exists in the union U.

A typical case where tie-breaking is useful is evaluation of discontinuous MPC feedback laws:

uopt = U.feval(x, 'primal', 'tiebreak', 'obj')

Here, if x is contained in multiple sets, then the function primal is only evaluated in the set which contain x and simultaneously has the **smallest** value of the tie-breaking function obj.

A special case of tie-breaking is the "first-set" rule where we are only interested in evaluating a function in the first set which contains x (despite the fact there can be multiple such sets). This is achieved by

fval = U.feval(x, 'function_name', 'tiebreak', @(x) 0)

Notes:

- Tie-breaking functions must be scalar-valued.

- No tie-breaking is done by default.

Evaluation in particular sets

fval = U.feval(x, 'myfun', 'regions', indices) evaluates function myfun over all sets indexed by indices. The output fval is always an $(m \times j)$ matrix, where j is the cardinality of indices.

Note that once the regions option is enabled, Union/feval() will not perform point location. Instead, it will evaluate the function in all sets indexed by indices, regardless of whether they contain x or not.

The regions option allows to quickly evaluate multiple functions as follows:

[first_value, idx] = U.feval(x, 'first_function')

second_value = U.feval(x, 'second_function', 'regions', idx)

In the second call, Union/feval will only evaluate second_function in sets specified by the indices option, hence skipping expensive point location.

INPUT

| U | Union of convex sets derived from the ConvexSet class, e.g. Polyhedron, YSet, ... Class: Union |

x	A point at which the function should be evaluated. The point must be given as a column real vector with the same dimension as the convex set. Class: `double`
`function_name`	Name of the function to evaluate. The string must match one of the stored function names. If there is only one function attached, this argument can be omitted. Class: `char`

OUTPUT

`fval`	Function value at the point `x` over the union of convex sets `U`. Class: `double`
`feasible`	Logical value indicating if the point `x` is contained in the union or not. Class: `logical` Allowed values: 1 0
`idx`	Vector of indices of sets that contain the point `x`. Class: `double`
`tb_value`	Value of the tie-breaking function if the point belongs to multiple sets. Class: `double`

EXAMPLE(s)

Example 1

PWA function over two Yalmip sets and polyhedron

```
x = sdpvar(2,1);

A = [1 -0.2; 0.4 -1];

F = [ norm(A*x-[1;1])<=2; [1 -2]*x<=0.4 ];

G = [ [1 -2]*x>=0.4; -1.5<= x <=1.5 ];
```

Create `YSet` objects out of Yalmip constraints.

```
Y(1) = YSet(x,F);

Y(2) = YSet(x,G);
```

Add two functions "a" and "b" to the sets.

```
Y.addFunction(AffFunction([1,-0.4],-1),'a');
```

37

```
Y.addFunction(AffFunction(3*eye(2),[-1;1]),'b');
```

Create the `Union` object.

```
U = Union(Y);
```

Add an affine set to the union

```
P = Polyhedron('Ae',[0 1],'be',-1.5)
```

```
Affine set with 1 equations in R^2
```

```
P.addFunction(AffFunction([-0.6,0.5],0.8),'a');
```

```
P.addFunction(AffFunction(eye(2)),'b');
```

```
U.add(P);
```

Evaluate function "a" for the point $v = (1, 1)$

```
v = [1;1];
```

```
y1 = U.feval(v,'a')
```

```
y1 =

-0.4
```

Plot the sets to see the union.

```
U.plot('color','lightgreen');
```

```
Plotting...
26 of 40
```
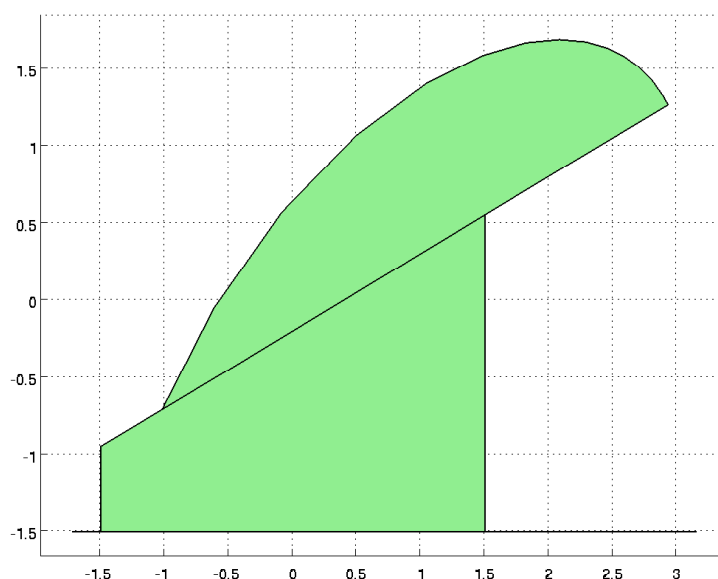
Assume that we provide a point that lies in two sets, for instance [1;-1.5]

```
y2 = U.feval([1;-1.5],'b')
```

```
y2 =

2                    1
-3.5                 -1.5
```

The output is a matrix because this point is contained in two sets. In particular, it lies exactly on the boundary of those sets. If we want just one output, we specify a tie-breaking function.
Evaluate the function 'b' with respect to a tie-breaking function 'a'.

```
y3 = U.feval([1;-1.5],'b','tiebreak','a')
```

```
y3 =

1
-1.5
```

SEE ALSO

fplot

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

HORZCAT

Horizontal concatenation for union objecs.

SYNTAX

```
U = [U1 , U2]

S = horzcat(U1,U2)
```

DESCRIPTION

Overloaded method for horizontal concatenation of unions. It is not possible to concatenate objects of different type to the same array (e.g. `PolyUnion` and `Union`). Similarly, it is not possible to concatenate into matrices, only vectors are allowed.

INPUT

U1   Object of the `Union` class.
     Class: `Union`

U2   Object of the `Union` class.
     Class: `Union`

OUTPUT

U   The array of the `Union` objects.
    Class: `Union`

EXAMPLE(s)

Example 1

We have two polyhedra in `U1` object.

```
P(1) = ExamplePoly.randHrep;

P(2) = ExamplePoly.randHrep;

U1 = Union(P);
```

The object `U2` contains one polyhedron.

```
U2 = Union(ExamplePoly.randHrep)
```

```
Union of 1 convex sets.
Functions : none
```

Concatenate unions horizontally.

41

```
U = [U1,U2]
```

Array of 2 Unions.

SEE ALSO

vertcat

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

REMOVE

Remove set from Union object.

SYNTAX

```
U = remove(U,index)
```

```
U.remove(index)
```

DESCRIPTION

Removes the `Set` from the union based on the `index`. The elements of the `Union` object are stored under `U.Set` property as a cell array. The `index` must correspond to elements in this array to remove.

INPUT

U	The object of the `Union` class. Class: `Union`
index	An index set derived that specifies which sets to remove from the union. Class: `double`

OUTPUT

U	Union of the sets. Class: `Union`

EXAMPLE(s)

Example 1

Construct `Union` object from 3 polyhedra.

```
P(1) = Polyhedron('ub',-2);
```

```
P(2) = Polyhedron('lb',-1,'ub',1);
```

```
P(3) = Polyhedron('lb',2);
```

Create union out of these polyhedra.

```
U = Union(P)
```

```
Union of 3 convex sets.
Functions : none
```

43

Remove the second polyhedron from the union.

```
U.remove(2)
```

```
Union of 2 convex sets.
Functions : none
```

SEE ALSO

Union, add

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
     mailto:herceg@control.ee.ethz.ch

UNION

Represents a general union of convex sets in MPT

SYNTAX

```
U = Union(Set)
```

DESCRIPTION

The `Union` object represent collection of various convex sets. The only restriction for the sets is to be convex, i.e. they have to be derived from the `ConvexSet` class.
You can associate functions to any of the set via `addFunction` method of the `ConvexSet` class. Function handles and all properties of the sets can be accessed via `Union.Set` property based on the index.
For a list of available methods type "methods('Union')".

INPUT

Set Any object derived from the `ConvexSet` class.
     Class: `ConvexSet`

OUTPUT

U Object of the `Union` class.
   Class: `Union`

EXAMPLE(s)

Example 1

Construct union of two `YSet` objects.
Define the objects in YALMIP

```
x = sdpvar(2,1);

F1 = [0<= x <=1];

F2 = [ norm(x-[1;1]) <= 1];

Y(1) = YSet(x,F1);

Y(2) = YSet(x,F2);
```

Create the union

```
U=Union(Y)
```

```
Union of 2 convex sets.
Functions : none
```

Plot the union
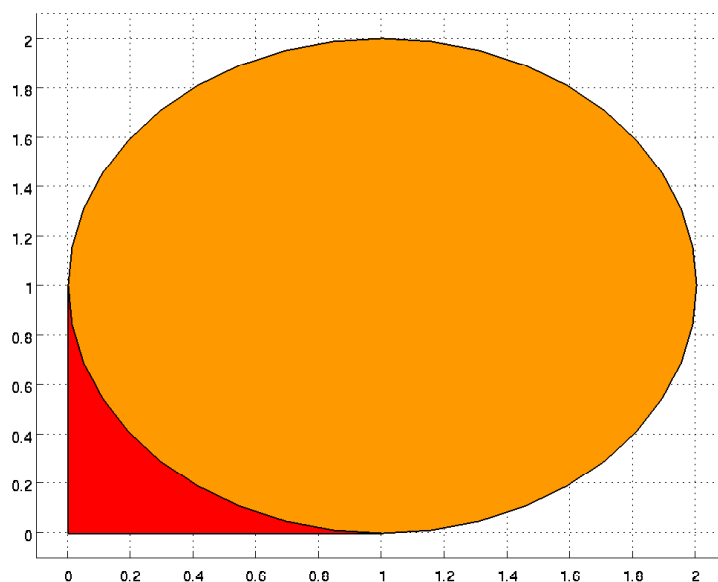
`U.plot`

Example 2

Construt the union of two polyhedra in 1D.
Define the polyhedra

`P(1) = Polyhedron('lb',-5,'ub',1);`

`P(2) = Polyhedron('lb',0);`

Create the union

`U = Union(P)`

```
Union of 2 convex sets.
Functions : none
```

Plot the polyhedra

`U.plot('linewidth',3)`
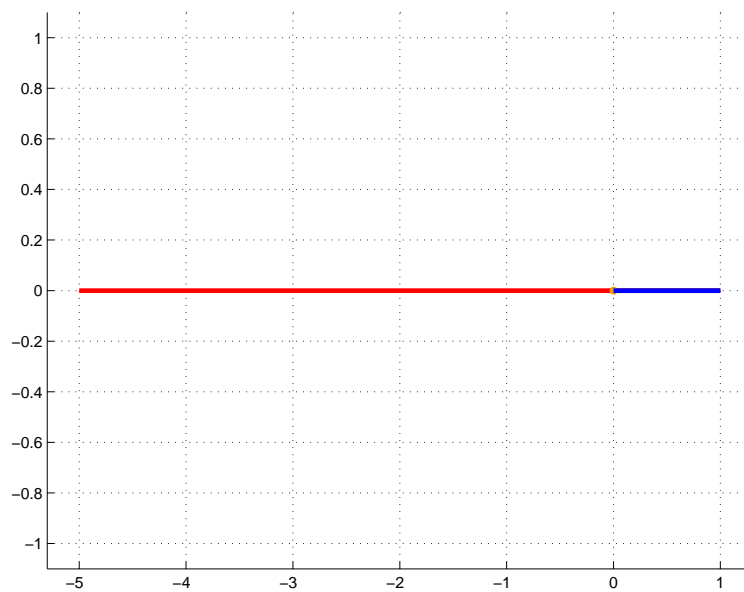
SEE ALSO

YSet, Polyhedron, PolyUnion

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

COPY

Create a copy of an object derived from the `Union` class.

SYNTAX

```
Unew = U.copy
```

```
Unew = copy(U)
```

DESCRIPTION

Create a copy of the union `U` and return a new object `Unew` with same properties as the original union object. Changing one object does not affect the second one because the objects are not connected by reference.

Note that this method is different versus the equality assignment `Unew = U` which points to the same object.

INPUT

U   Any object derived from the `Union` class, e.g. `PolyUnion`, etc.
    Class: `Union`

OUTPUT

Unew   New object which is an exact copy of the original object U.
       Class: `Union`

SEE ALSO

PolyUnion

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
        mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
        mailto:herceg@control.ee.ethz.ch

LISTFUNCTIONS

Extract list of functions stored with the union.

SYNTAX

```
U.listFunctions
```

DESCRIPTION

Extract list of function names stored with the union of convex sets. If there are no functions associated, the result is empty.

INPUT

P   Object derived from `Union` class.
      Class: `Union`

EXAMPLE(s)

Example 1

Create union of polyhedra with two affine function "width", "height".

```
P(1) = Polyhedron('ub',-1);
```

```
P(1).addFunction(AffFunction(-1,1),'weight');
```

```
P(1).addFunction(AffFunction(-2,0.5),'height');
```

```
P(2) = Polyhedron('lb',-1,'ub',1);
```

```
P(2).addFunction(AffFunction(0,1),'weight');
```

```
P(2).addFunction(AffFunction(-3,0.5),'height');
```

```
P(3) = Polyhedron('lb',1);
```

```
P(3).addFunction(AffFunction(1,1),'weight');
```

```
P(3).addFunction(AffFunction(-4,0.5),'height');
```

Create the `PolyUnion` object without specifying any properties.

```
U = PolyUnion(P)
```

```
PolyUnion in the dimension 1 with 3 polyhedra.
Functions : 2 attached "height","weight"
```

Get the function names

```
U.listFunctions
```

49

```
ans =

'height'    'weight'
```

SEE ALSO

Union, PolyUnion

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

CONTAINS

Test if a point is contained inside the union of convex sets.

SYNTAX

```
[isin, inwhich, closest] = contains(U, x, fastbreak)

[isin, inwhich, closest] = U.contains(x)

[isin, inwhich, closest] = U.contains(x, fastbreak)
```

DESCRIPTION

Check if the point `x` is contained in any of the sets in the union `U`. The result it a logical statement if $x \in U$ and false otherwise. If the point is contained inside the union, indices of sets in which the point lie are returned. If the point does not lie in the union, the index of the region with the least distance to the point x is returned. All sets in the union must have the same dimension, otherwise the evaluation cannot be done.

INPUT

U	Single union object that holds sets derived from the `ConvexSet` class. If `U` is an array, use `U.forEach()`. Class: `Union`
x	A point in the same dimension as all the sets in the union. Class: `double`
fastbreak	Do a quick stop in the consecutive search when `x` is contained in the first set it founds. Class: `logical` Allowed values: `true` `false` Default: `false`

OUTPUT

isin	True if $x \in U$ and false otherwise. Class: `logical` Allowed values: `true` `false`
inwhich	Indices of sets that contain x. If the `fastbreak` option is turned on, a single index is returned. Class: `double`

closest   If the point is not contained inside the union, this output indicates the index of the set that is the closest to the point x. Note: since this computation is expensive, do not ask for the third output unless you really need it. Class: `double`

EXAMPLE(s)

Example 1

Create two yalmip polytopic sets in 2D.

`x = sdpvar(2,1);`

`Y(1) = YSet(x,[randn(8,2)*x<=ones(8,1)]);`

`Y(2) = YSet(x,[randn(8,2)*x<=ones(8,1)]);`

Create an union of the sets

`U = Union(Y);`

Check if the point $x = (1,0)$ is contained in the union.

`x = [1; 0];`

`[isin,inwhich,closest] = U.contains(x)`

```
isin =

1


inwhich =

1     2


closest =

[]
```

SEE ALSO

feval

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

REMOVEALLFUNCTIONS

Remove all functions that are associated to this union of convex sets.

SYNTAX

```
U = U.removeAllFunctions
```

```
U = removeAllFunctions(U)
```

DESCRIPTION

Removes all `Function` objects associated to this union.

INPUT

U	Object of the `Union` class that holds objects derived from `ConvexSet` class. Class: `Union`

OUTPUT

U	Modified object of `Union` class without the function handles. Class: `Union`

EXAMPLE(s)

Example 1

Create union of two polyhedra given by Yalmip constraints.

```
x = sdpvar(3,1);
```

```
F1=[randn(12,3)*x<=ones(12,1)];
```

```
F2=[randn(12,3)*x<=ones(12,1)];
```

```
Y(1) = YSet(x,F1);
```

```
Y(2) = YSet(x,F2);
```

Add function handles to each set.

```
for i=1:2,
Y(i).addFunction(AffFunction(randn(1,3)),'a');
Y(i).addFunction(AffFunction(randn(1,3)),'b');
end
```

Create the union of polyhedra

```
U = Union(Y)
```

```
Union of 2 convex sets.
Functions : 2 attached "a","b"
```

Remove all functions handles from the union

```
U.removeAllFunctions
```

```
Union of 2 convex sets.
Functions : none
```

SEE ALSO

`addFunction`, `removeFunction`

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    `mailto:herceg@control.ee.ethz.ch`

ADD

Insert set to Union object.

SYNTAX

```
U = add(U,Set)
```

```
U.add(Set)
```

DESCRIPTION

Insert the `Set` derived from from the `ConvexSet` inside the `Union` object. The `Set` can be also an array. Each element of the array is stored under `Union.Set` property as a cell array because objects derived from the `ConvexSet` class cannot be concatenated.
If the `Set` is empty, it is not added to the union.

INPUT

U The object of the `Union` class.
   Class: `Union`

Set A convex set derived from `ConvexSet` class.
    Class: `ConvexSet`

OUTPUT

U Union of the sets.
   Class: `Union`

EXAMPLE(s)

Example 1

Construct empty `Union` object.

```
U = Union
```

```
Empty Union.
```

Insert random polyhedron inside the union.

```
U.add(ExamplePoly.randHrep);
```

The union now consist of one set.

```
U.Set,U.Num
```

```
ans =

[1x1 Polyhedron]


ans =

1
```

Example 2

Union of `YSet` object and `Polyhedron` object.
Define `YSet` object.

```
x = sdpvar(2,1);

F = [0<= x <= 1];

Y = YSet(x,F);
```

Define `Polyhedron` object.

```
P = Polyhedron('lb',[2;0],'ub',[3;3])

Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities   4 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

Create the union

```
U = Union(Y); U.add(P)

Union of 2 convex sets.
Functions : none
```

Empty polyhedron object is not added

```
U.add(Polyhedron)

Union of 2 convex sets.
Functions : none
```

SEE ALSO

Union, remove

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ISOVERLAPPING

Test if the union of polyhedra contains overlaps.

SYNTAX

```
ts = U.isConnected
```

```
ts = isConnected(U)
```

DESCRIPTION

Return true if the union U of polyhedra contains overlaps and false otherwise. Once this method has been called, the information about the overlaps can be retrieved from U.Internal.Overlaps property.

This function considers following two cases to detect overlaps:

1. If two full-dimensional polyhedra overlap, then the intersection of these polyhedra must be full-dimensional.

2. If low-dimensional and full-dimensional polyhedra overlap, then the intersection of these polyhedra must not be empty.

Note that this function is computationally demanding and is suitable for unions with small number of polyhedra.

INPUT

U   Union of polyhedra in the same dimension.
     Class: PolyUnion

OUTPUT

ts   True if union of polyhedra has overlaps and false otherwise.
      Class: logical
      Allowed values: true
                       false

EXAMPLE(s)

Example 1

Create three full-dimensional polyhedra that overlap.

```
P(1) = ExamplePoly.randHrep;
```

```
P(2) = ExamplePoly.randHrep;
```

```
P(3) = ExamplePoly.randHrep;
```

Create union out of these polyhedra without specifying the properties

```
U = PolyUnion(P)
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Functions : none
```

Plot the polyhedra to see that they overlap.

```
P.plot;
```
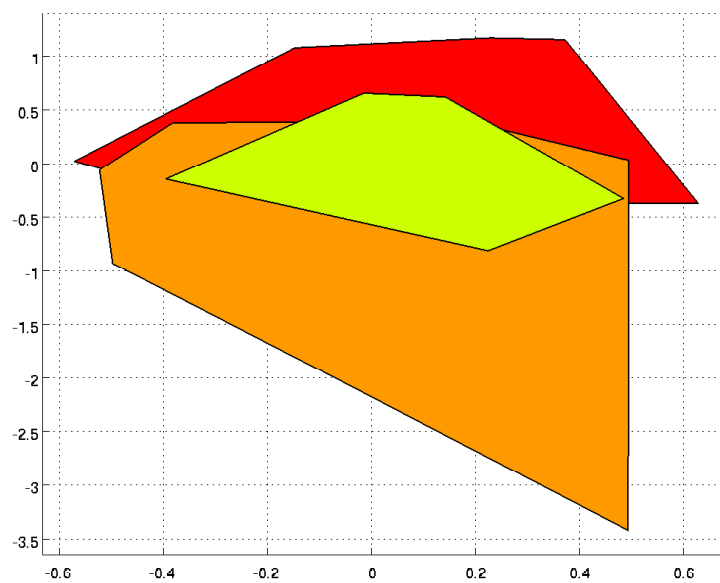

Check if the union is overlapping

```
U.isOverlapping
```

```
ans =
```

```
1
```

The information about the overlaps can be accessed in

```
U.Internal.Overlaps
```

```
ans =
```

```
1
```

Example 2

Full-dimensional and low-dimensional polyhedra that overlap.
Generate affine set in dimension 3.

```
A = Polyhedron('Ae',[1 -0.5 2; -1 -4 0],'be',[-1;0.8]);
```

Generate box in 3D

```
B = Polyhedron('lb',[-1;-1;-1],'ub',[1;1;1]);
```

Create union out of these polyhedra

```
U = PolyUnion([A,B]);
```

Plot to see the sets if they overlap

```
U.plot('LineWidth',2,'alpha',0.1)
```
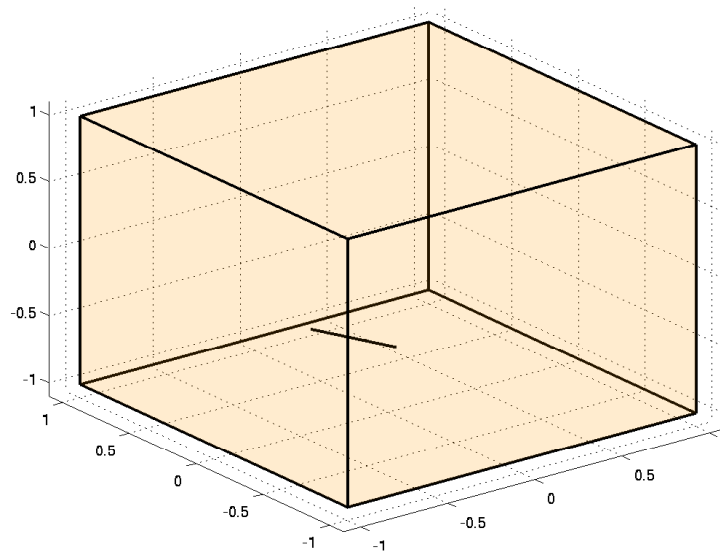

Check if the union is overlapping

```
U.isOverlapping
```

```
ans =

1
```

The result of the operation is stored under

```
U.Internal.Overlaps
```

59

```
ans =

1
```

SEE ALSO

isConvex, isConnected, isFullDim, isBounded

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ISBOUNDED

Test if the union is built from bounded polyhedra.

SYNTAX

```
ts = U.isBounded
```

```
ts = isBounded(U)
```

DESCRIPTION

Return true if all polyhedra in the union `U` are bounded and false otherwise. Once this method has been called, the information about the boundedness can be retrieved from `U.Internal.Bounded` property.

INPUT

U   Union of polyhedra in the same dimension.
Class: `PolyUnion`

OUTPUT

ts   True if all polyhedra in the union are bounded and false otherwise.
Class: `logical`
Allowed values: `true`
                `false`

EXAMPLE(s)

Example 1

Union of bounded and unbounded:

```
P(1) = Polyhedron('V',randn(5,2));
```

```
P(2) = Polyhedron('V',randn(5,2),'R',[0 1]);
```

Create union

```
U = PolyUnion(P)
```

```
PolyUnion in the dimension 2 with 2 polyhedra.
Functions : none
```

Check if the union is bounded

```
U.isBounded
```

`ans =`

`0`

The boundedness property can be retrieved from

`U.Internal.Bounded`

`ans =`

`0`

SEE ALSO

`isConvex`, `isOverlapping`, `isConnected`, `isFullDim`

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

MERGE

Greedy merging of polyhedra

SYNTAX

```
U.merge
```

```
merge(U)
```

DESCRIPTION

Simplifies the union of polyhedra by merging the neighboring polyhedra if their union is convex. The algorithm cycles through the regions and checks if any two regions form a convex union. If so, the algorithm combines them in one region, and continues checking the remaining regions. To improve the solution, multiple merging loops can be enabled in options.

INPUT

U   Union of polyhedra in the same dimension.
    Class: `PolyUnion`

EXAMPLE(s)

Example 1

Create a random V-polyhedron that contains the origin.

```
P = 5*ExamplePoly.randVrep;
```

Triangulate the polyhedron

```
T = P.triangulate;
```

Plot the triangular regions

```
T.plot
```
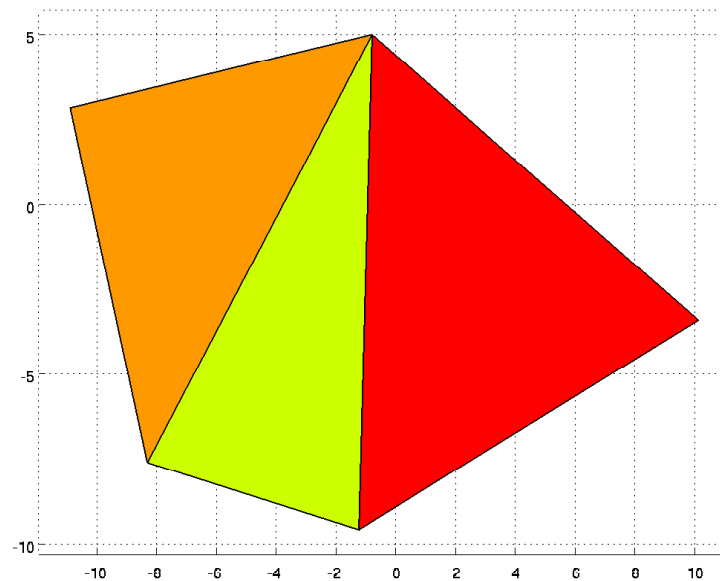
Create the union of polyhedra by specifying some properties.

```
U = PolyUnion('Set',T,'convex',true,'overlaps',false,'fulldim',true,'bounded',true)
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Properties of the union:
Convex: 1
Overlaps: 0
Connected: 1
Bounded: 1
FullDim: 1
Functions : none
```

Merge the polyhedra back.

```
U.merge
```

```
PolyUnion in the dimension 2 with 1 polyhedra.
Properties of the union:
Convex: 1
Overlaps: 0
Connected: 1
Bounded: 1
FullDim: 1
Functions : none
```

Plot the merged union

```
U.plot
```

64
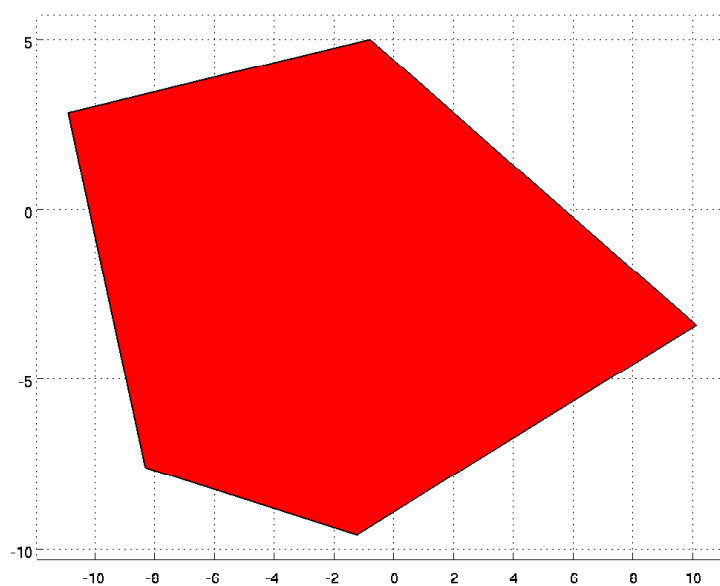
SEE ALSO

reduce

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

© **2005** Frank J. Christophersen: ETH Zurich
    mailto:fjc@control.ee.ethz.ch

© **2005** Tobias Geyer: ETH Zurich
    mailto:geyer@control.ee.ethz.ch

POLYUNION

Represents a union of polyhedra in the same dimension

SYNTAX

```
PU = PolyUnion(P)

PU = PolyUnion('Set',P,'Domain',D,'Convex',true,'Overlaps',
    false,...)
```

DESCRIPTION

The `PolyUnion` object represent collection of polyhedra in the same dimension. The only restriction to construct the `PolyUnion` object is that the sets have a common dimension. The union thus can contain bounded, unbounded, and lower-dimensional polyhedra. Empty sets are removed from at the time of construction of `PolyUnion` object.

You can associate various properties with the `PolyUnion` object:

- Domain of the union can be set via the `Domain` property. If not provided, the domain will be equal to the underlying set of the union.

- To mark the union as **convex**, you can set the `Convex` property `true` or `false`. If you specify that the union is convex, the remaining algorithms will rely on this information which will can speed up the computations significantly.

- To mark that there are **overlaps** in the union, you can set the `Overlaps` property as `true` or `false`. If the union contains non-overlapping polyhedra, the union forms a **partition**.

- To mark that the union is **connected** in the union, you can set the `Connected` property as `true` or `false`. If the union is convex, if implies that the union is also connected.

- To mark that the union comprises only of **bounded** polyhedra, you can set the `Bounded` property as `true` or `false`.

- To mark that the union contains only **full-dimensional** polyhedra, you can set the `FullDim` property as `true` or `false`.

By specifying any of the above properties you promise that these properties are automatically satisfied by the provided set, otherwise unexpected results may occur. These properties can be accessed via `PU.Internal` field.

You can associate functions to any of the set via `addFunction` method of the `ConvexSet` class. Function handles and all properties of the sets can be accessed via `PolyUnion.Set` property based on the index.

For a list of available methods type "methods('PolyUnion')".

INPUT

P	Non-empty polyhedra in the same dimension. Class: `Polyhedron`
Domain	Specifies domains of the union.

Class: `Polyhedron`

`Convex`	Set this property to `true` if you are sure that the union of the polyhedra is convex. Class: `logical` Allowed values: `true` `false`
`Overlaps`	Set this property to `false` if you are sure that the union of the polyhedra is non-overlapping. Class: `logical` Allowed values: `true` `false`
`Connected`	Set this property to `true` if you are sure that the union of the polyhedra is connected. If the union is convex, then this property is automatically set as `true`. Class: `logical` Allowed values: `true` `false`
`Bounded`	Set this property to `true` if you are sure that all the polyhedra in the union are bounded. Class: `logical` Allowed values: `true` `false`
`FullDim`	Set this property to `true` if you are sure that all the polyhedra in the union are full-dimensional. Class: `logical` Allowed values: `true` `false`

OUTPUT

PU   Object of the `PolyUnion` class.
     Class: `PolyUnion`

EXAMPLE(s)

Example 1

Construt the convex and non-overlapping union of two polyhedra in 2D.
Define the polyhedra

```
P(1) = Polyhedron('lb',[-1;0],'ub',[0;1]);
```

```
P(2) = Polyhedron('lb',[0;0],'ub',[1;1]);
```

Assign linear functions to these sets

```
P(1).addFunction(AffFunction(eye(2),[1;0]),'f');
```

```
P(2).addFunction(AffFunction(-eye(2),[1;0]),'f');
```

Create the union and set convexity and overlaps properties because these are known.

```
U = PolyUnion('Set',P,'Convex',true,'Overlaps',false);
```

Plot the polyhedra

```
U.plot
```
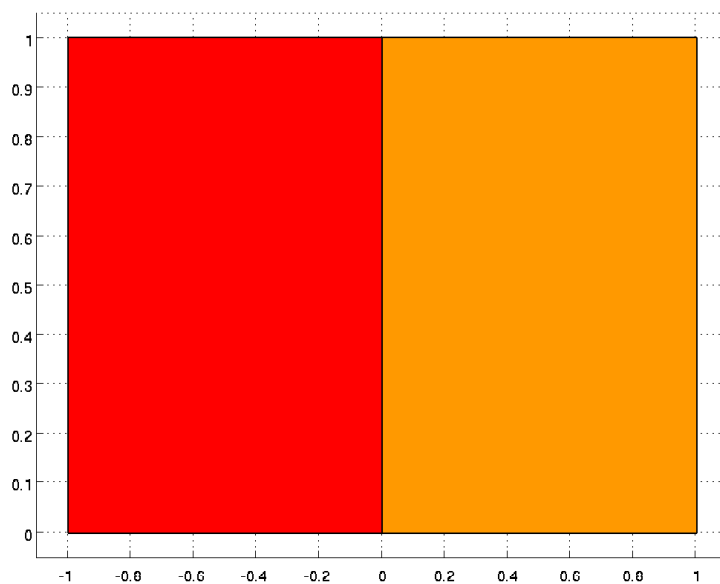

We can plot also functions over the union

```
U.fplot
```
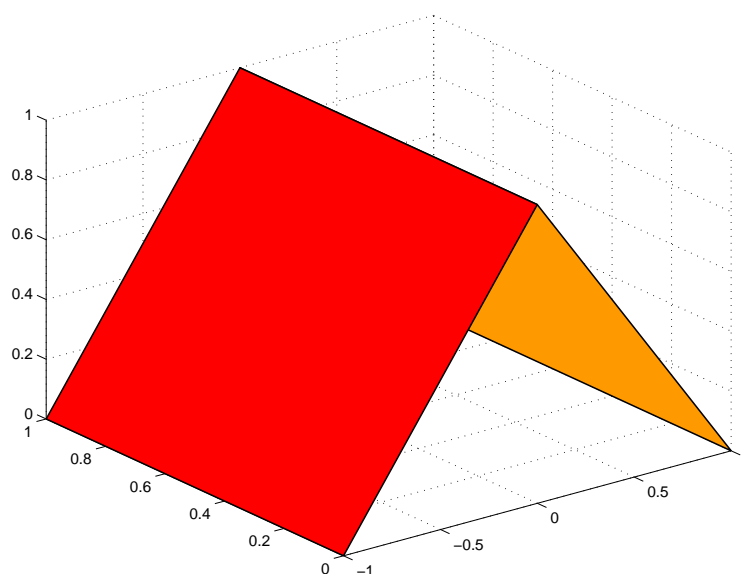
Example 2

Create three random polyhedra and merge them to `PolyUnion` object.

```
P(1) = ExamplePoly.randVrep;
```

```
P(2) = ExamplePoly.randVrep;
```

```
P(3) = ExamplePoly.randVrep;
```

We don't know any of the particular properties of the union, we can call the short syntax:

```
U = PolyUnion(P)
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Functions : none
```

Plot the sets

```
U.plot
```
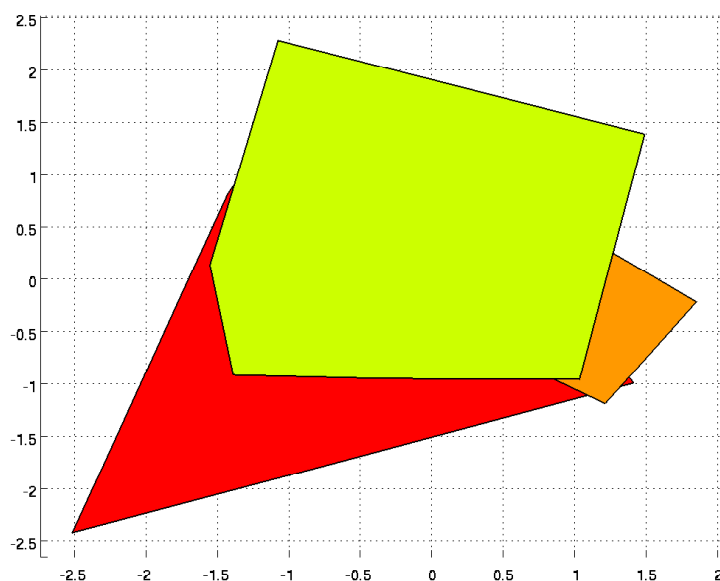

We can access the sets via `U.Set` property and add eventually function handles to each set.

```
U.Set(1).addFunction(AffFunction(eye(2)),'g');
```

```
U.Set(2).addFunction(AffFunction(2*eye(2)),'g');
```

```
U.Set(3).addFunction(AffFunction(3*eye(2)),'g');
```

Plot the function over the sets

`U.fplot`
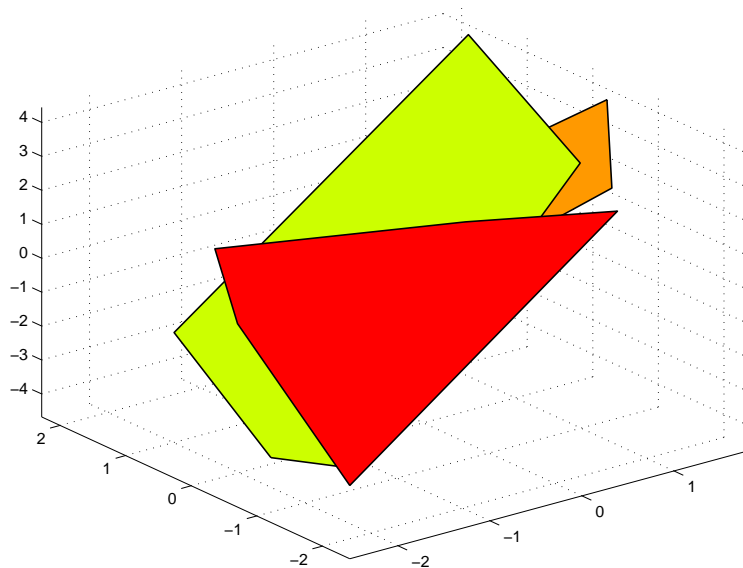
SEE ALSO

,

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

GETFUNCTION

Extract function from PolyUnion object.

SYNTAX

```
Un = U.getFunction(name)

Un = getFunction(U,name)
```

DESCRIPTION

Extract single function from the `PolyUnion` object, if it has some functions associated to the sets. The extraction is based on the string under which the functions are stored. This method is useful for retrieving particular data from the result returned by parametric solver which contains typically functions such as "primal", "obj" corresponding to primal solution and objective function.

INPUT

U   Union of polyhedra in the same dimension.
Class: `PolyUnion`

name   Name of the function to extract given as string.
Class: `string`

EXAMPLE(s)

Example 1

Create a partition by triangulation of polyhedron P.

```
P = 5*ExamplePoly.randVrep;
```

Triangulate the polyhedron

```
T = P.triangulate;
```

Assign two functions to each polyhedron and give them names "a" and "b".

```
for i=1:numel(T),
T(i).addFunction(AffFunction(randn(1,2)),'a');
T(i).addFunction(Function(@(x)norm(x,Inf)),'b');
end
```

Create the union of polyhedra by specifying some properties.

```
U = PolyUnion('Set',T,'convex',true,'overlaps',false,'fulldim',true,'bounded',true)
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Properties of the union:
Convex: 1
```

71

```
Overlaps: 0
Connected: 1
Bounded: 1
FullDim: 1
Functions : 2 attached "a","b"
```

Extract only function "b" from the union.

```
Un = U.getFunction('b')
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Properties of the union:
Convex: 1
Overlaps: 0
Connected: 1
Bounded: 1
FullDim: 1
Functions : 1 attached "b"
```

We can plot the function over partition `Un`.

```
Un.fplot
```


AUTHOR(s)
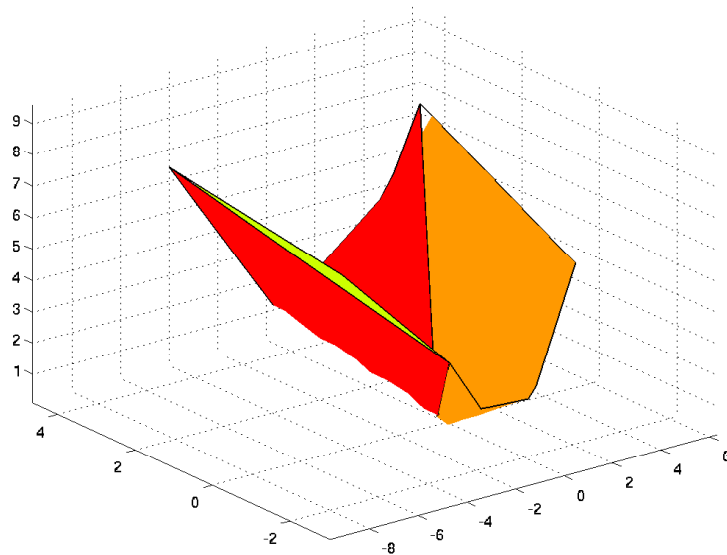
© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MINUS

Pontryagin/Minkowski difference for union of polyhedra

SYNTAX

```
U - W
```

```
U.minus(W)
```

DESCRIPTION

Computation of Pontryagin or Minkowski difference for the union of polyhedra in the same dimension. The algorithm for efficiently computing the Minkowski difference between a union of polytopes and a polytope is based on a talk by S. Rakovic and D. Mayne entitled *Constrained Control Computations* It was the keynote addressed at the GBT Meeting, London, November, 2002.

The algorithm proceeds in the following way:

1. Compute the convex hull of the union.

2. Compute the Minkowski difference of each of the polyhedron from the convex hull.

3. Compute the set difference between the convex hull and the union.

4. Compute the set difference between the Minkowski difference for each polyhedron and the set obtained in the previous step.

The result is a non-overlapping union of the polyhedra.

INPUT

U   Union of polyhedra in the same dimension.
    Class: `PolyUnion`

W   Polyhedron to be summed with the union that is in the
    same dimension as the union.
    Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create three polyhedra that form bounded union.

```
P(1) = Polyhedron('V',[-5 0;-4 0;-5 5;-4 5]);
```

```
P(2) = Polyhedron('V',[-4 1;-1 1.5;-4 4;-1 2.5]);
```

```
P(3) = Polyhedron('V',[-1 0; -1 4]);
```

```
U = PolyUnion('Set',P,'bounded',true)
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Properties of the union:
Bounded: 1
Functions : none
```

Compute Minkowski difference for full-dimensional and bounded polyhedron W.

```
W = Polyhedron('lb',[-0.1,-0.2],'ub',[0.1,0.2]);

Uw = U -W
```

```
PolyUnion in the dimension 2 with 5 polyhedra.
Properties of the union:
Overlaps: 0
Bounded: 1
Functions : none
```

Plot the new union `Uw`
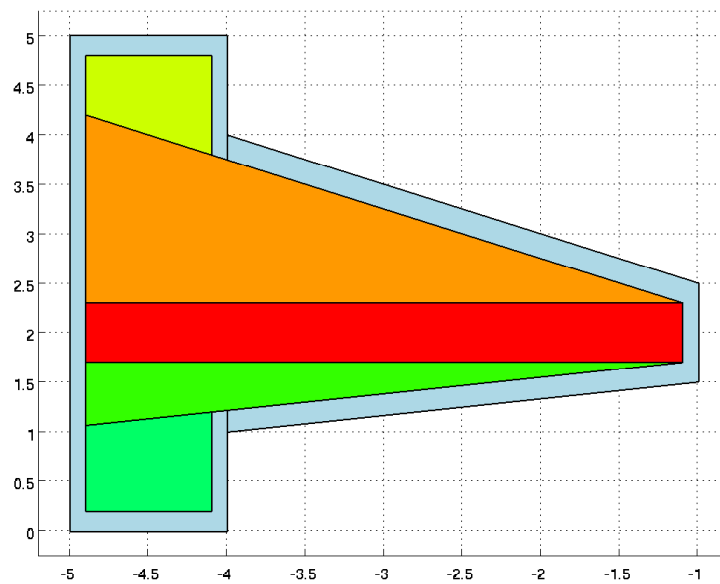
```
U.plot('color','lightblue'),hold on,Uw.plot
```


Compute Minkowski addition for low-dimensional and bounded polyhedron S.

```
S = Polyhedron('lb',[-0.1,0],'ub',[0.1,0]);

Us = U -S
```

```
PolyUnion in the dimension 2 with 5 polyhedra.
Properties of the union:
Overlaps: 0
Bounded: 1
FullDim: 1
Functions : none
```

Plot the new union `Us`

`U.plot('color','lightgreen'),hold on,Us.plot`
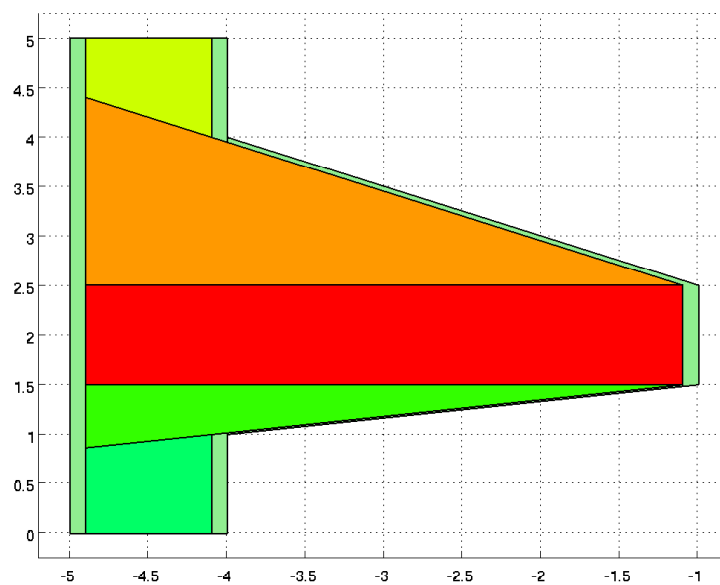
SEE ALSO

convexHull, plus

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
  mailto:herceg@control.ee.ethz.ch

© **2005** Mario Vasak: FER Zagreb
  mailto:mario.vasak@fer.hr

© **2003-2013** Michal Kvasnica: STU Bratislava
  mailto:michal.kvasnica@stuba.sk

© **2003-2005** Pascal Grieder: ETH Zurich
  mailto:grieder@control.ee.ethz.ch

© **2003** Mato Baotic: ETH Zurich
  mailto:baotic@control.ee.ethz.ch

REMOVEFUNCTION

Remove function from all Polyhedra in the union based on the function name.

SYNTAX

```
U = removeFunction(U,name)
```

```
U.removeFunction(name)
```

DESCRIPTION

Removes `Function` object from the union of polyhedra identified by the string `name`. The functions stored with the union can be retrieved using `listFunctions` method. Index or string must match with the corresponding arrays.

INPUT

U   Object of the `PolyUnion` class that holds polyhedra in
        the same dimension.
        Class: `PolyUnion`

name   Name of the function to remove from the union.
        String must match one of the names as retrieved by
        `listFunctions` method.For multiple names, provide
        `name` as a cell array of strings.
        Class: `char`

OUTPUT

U   Modified object of the `PolyUnion` class without the
    function handles that has been removed.
    Class: `PolyUnion`

EXAMPLE(s)

Example 1

Create an union of polyhedra that holds two quadratic functions.
Define Polyhedra

```
P(1) = Polyhedron('lb',-2,'ub',0);
```

```
P(2) = Polyhedron('lb',-1,'ub',1);
```

Define the functions

```
f1 = QuadFunction(2,-1,1);
```

```
f2 = QuadFunction(3,-2,0);
```

Add these functions to the polyhedron array

```
P.addFunction(f1,'f1');
```

```
P.addFunction(f2,'f2');
```

Create the `PolyUnion` object with some properties.

```
U = PolyUnion('Set',P,'Overlaps',true)
```

```
PolyUnion in the dimension 1 with 2 polyhedra.
Properties of the union:
Overlaps: 1
Functions : 2 attached "f1","f2"
```

Remove the function "f2" from the set

```
U.removeFunction('f2')
```

```
PolyUnion in the dimension 1 with 2 polyhedra.
Properties of the union:
Overlaps: 1
Functions : 1 attached "f1"
```

Union of polyhedra now contains only 'f1' function

```
U.listFunctions
```

```
ans =

'f1'
```

SEE ALSO

addFunction, removeAllFunctions

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

DISP

Overload display for `PolyUnion` class.

SYNTAX

`display(U)`

`U.display`

DESCRIPTION

Default display for `PolyUnion` class.

INPUT

  U `PolyUnion` object.
      Class: `PolyUnion`

SEE ALSO

Union

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

OUTERAPPROX

Computes outer bounding box for the union of polyhedra

SYNTAX

```
B = outerApprox(U)

B = U.outerApprox
```

DESCRIPTION

Compute the smallest axis-aligned hypercube that contains all polyhedra in this union. The lower and upper bounds of the hypercube are stored under `Internal` property, i.e. `Internal.lb` for lower bound and `Internal.ub` for upper bound.

INPUT

U   Union of polyhedra in the same dimension
    Class: `PolyUnion`

OUTPUT

B   Bounding box B described as `Polyhedron` in H-representation.
    Class: `Polyhedron`

EXAMPLE(s)

Example 1

We have union of two polyhedra.

```
P(1) = Polyhedron('A',randn(9,2),'b',2*ones(9,1));

P(2) = Polyhedron('V',randn(9,2));
```

Create the union without specifying the properties

```
U = PolyUnion(P);
```

Compute the bounding box

```
B = U.outerApprox;
```

Plot the sets such that the outer approximation is wired.

```
U.plot; hold on; B.plot('wire',true,'LineWidth',2)
```
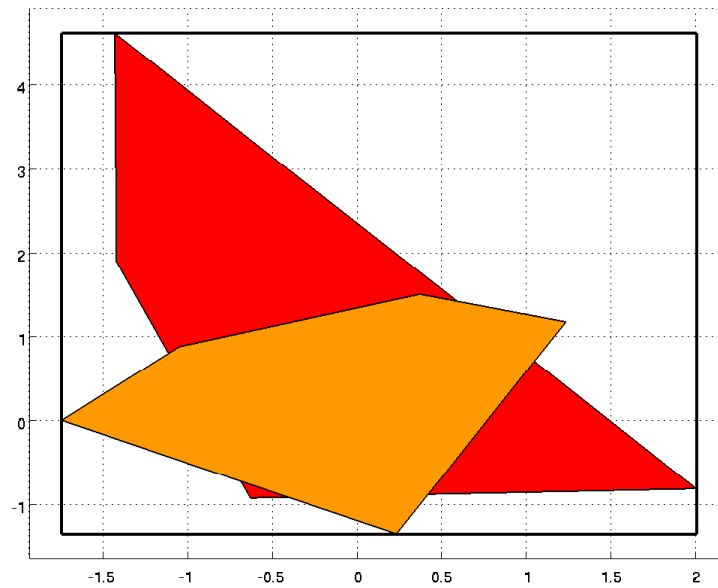
SEE ALSO

convexHull

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

FPLOT

Plot function over the union of polyhedra.

SYNTAX

```
h = Set.fplot()

h = Set.fplot('name', 'Prop1', value1, 'Prop2', value2)

h = fplot(Set, 'name', 'Prop1', value1, 'Prop2', value2)
```

DESCRIPTION

Plot function over the union of polyhedra. If there are more functions, then the string **name** indicates the function to be plotted. If the function is vector valued, i.e. its range is greater than 1, than the first element of the function is plotted by default. For vector valued functions the **position** property indicates which element of the function value to plot.

Figure properties, such as color, line width, etc, can be specified with "Property" - "Value" pairs.

INPUT

U	**PolyUnion** object that contains polyhedra in the same dimension with associated functions. Class: **PolyUnion**
name	If there are more functions over one set, the string **name** identifies the function to be plotted. This argument can be omitted if only one function is attached to a set. Class: **char**
Prop1	Specification of figure properties. Class: **char**

Allowed values: `position`   For vector valued functions, the `position` indicates which element of the function value to plot.

`Grid`   With how many gridpoints to grid the circle/-sphere. Default is 20.

`Color`   The color of the plot specified by real RGB vector or a string name of the color (e.g. 'gray');

`Wire`   Highlight the edges of the set. Default is false.

`LineStyle`   Specify the type of the line to plot edges of the set. Accepted values are '-',':','-.','–', and'none'.

`LineWidth`   Specify the width of the line. Default is 1.

`Alpha`   Transparency of the color. The value must be inside [0,1] interval. Default is 1.

`Contour`   Add contour graph. Default is false.

`ContourGrid`   With how many grid points to plot the contour graph. Default is 30.

`show_set`   Plot the domain of the function. Default is false.

`showgrid`   Show the grid inside the set. Default is false.

`colormap`   Color map to use given as a string or a function handle. Default is 'mpt'.

`colororder`   Either 'fixed' or 'random'. Default is 'fixed'.

`value1`   Assigns value to `Prop1`.

OUTPUT

`h`   Handle related to graphics object.
Class: `handle`

EXAMPLE(s)

Example 1

We have piecewise affine function defined over a union of convex sets.
Construct the set of four polyhedra in 1D.

```
P(1) = Polyhedron('lb',-5,'ub',-1);

P(2) = Polyhedron('lb',-1,'ub',1);

P(3) = Polyhedron('lb',1,'ub',4);

P(4) = Polyhedron('lb',4,'ub',5);
```

Add affine functions to these sets

```
P(1).addFunction(AffFunction(-2,1),'a');

P(2).addFunction(AffFunction(0.1,3.1),'a');
```

```
P(3).addFunction(AffFunction(-0.2,3.4),'a');
```

```
P(4).addFunction(AffFunction(2,-5.4),'a');
```

Construct the union saying that it is convex, non-overlapping, bounded, and full-dimensional.

```
U = PolyUnion('Set',P,'convex',true,'overlaps',false,'bounded',true,'fulldim',true)
```

```
PolyUnion in the dimension 1 with 4 polyhedra.
Properties of the union:
Convex: 1
Overlaps: 0
Connected: 1
Bounded: 1
FullDim: 1
Functions : 1 attached "a"
```

Plot the functions over the union with the line width equal 3 and by showing the polyhedra as well.

```
U.fplot('a','LineWidth',3,'show_set',true);
```
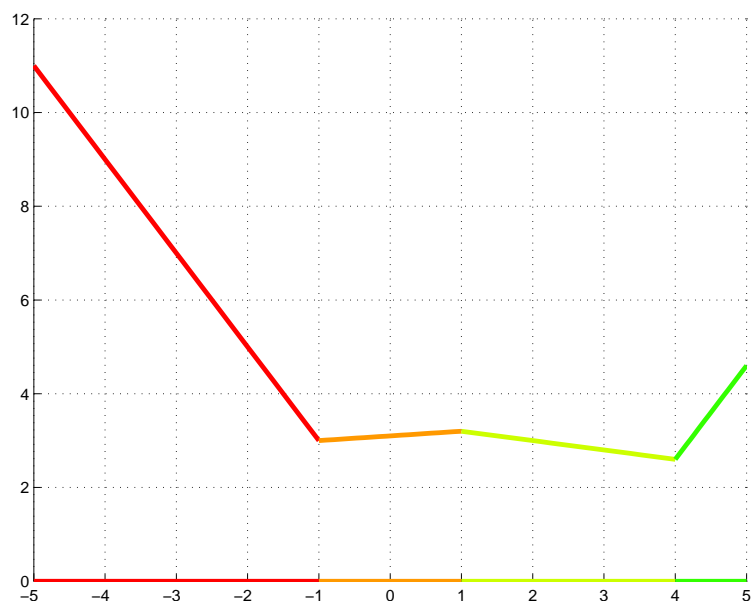

Example 2

Plot piecewise-quadratic function
Generate a polygon.

```
v = [-1,1.7; 1.1,0.5; 0,2.3; 1,-1.7; 0,-2.3; -1.1 -0.5];
```

```
Q = Polyhedron(v);
```

Triangulate the polygon and create an union of polyhedra.

```
T = PolyUnion('Set',Q.triangulate,'Convex',true,'overlaps',false)
```

```
PolyUnion in the dimension 2 with 4 polyhedra.
Properties of the union:
Convex: 1
Overlaps: 0
Connected: 1
Functions : none
```

Attach a quadratic function over this union and name it "power".

```
T.addFunction(QuadFunction(diag([1,2.3]),[-3,0.5],5),'power')
```

```
PolyUnion in the dimension 2 with 4 polyhedra.
Properties of the union:
Convex: 1
Overlaps: 0
Connected: 1
Functions : 1 attached "power"
```

Plot the function over the union with random color, transparency equal 0.4, and show grid.

```
T.fplot('colororder','random','show_set',true,'alpha',0.4,'showgrid',true)
```
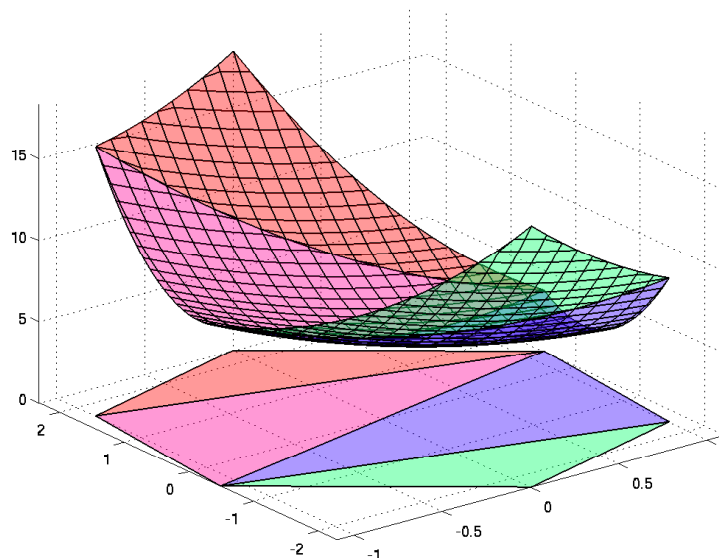

SEE ALSO

plot

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
      mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
      mailto:michal.kvasnica@stuba.sk

ISCONNECTED

Test if the union of polyhedra form a connected union.

SYNTAX

```
ts = U.isConnected
```

```
ts = isConnected(U)
```

DESCRIPTION

Return true if the union `U` of polyhedra is connected and false otherwise. Once this method has been called, the information about the connectivity can be retrieved from `U.Internal.Connected` property. Note tha if the union `U` is convex, it implies that the union is connected.
Note that this function is computationally demanding and is suitable for unions with small number of polyhedra.

INPUT

U   Union of polyhedra in the same dimension.
Class: `PolyUnion`

OUTPUT

ts   True if union of polyhedra is connected and false otherwise.
Class: `logical`
Allowed values: `true`
`false`

EXAMPLE(s)

Example 1

Create three polyhedra that are not connected.

```
P(1) = Polyhedron('lb',[-1;0],'ub',[-1;1]);
```

Shift the polyhedron `P(1)`

```
P(2) = P(1) + [1;0];
```

```
P(3) = P(1) + [2;0];
```

Plot the polyhedra to see that they are not connected.

```
P.plot('LineWidth',3); axis([-2 2 -1 2]);
```
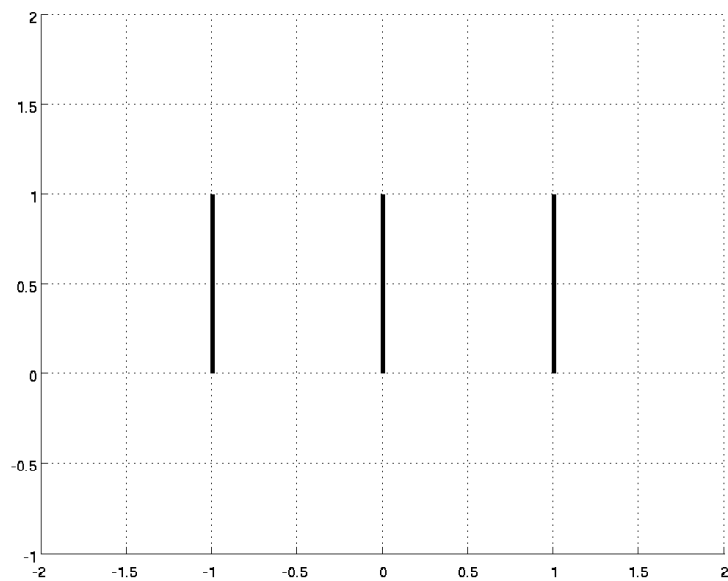
Create union out of these polyhedra without specifying the properties

```
U = PolyUnion(P)
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Functions : none
```

Check if the union is connected

```
U.isConnected
```

```
ans =
```

```
0
```

The information about the connectivity can be accessed in

```
U.Internal.Connected
```

```
ans =
```

```
0
```

SEE ALSO

isConvex, isOverlapping, isFullDim, isBounded

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

PLUS

Minkowski addition for union of polyhedra

SYNTAX

```
U + W
```

```
U.plus(W)
```

DESCRIPTION

Computation of Minkowski addition for the union of polyhedra in the same dimension. The algorithm proceeds in the following way:

1. Compute the Minkowski summation for each of the polyhedron contained in the union to get.

2. Compute the convex hull of the union.

3. Compute the set difference between the convex hull and the union.

4. Compute the set difference between the Minkowski sum for each polyhedron and set obtained in the previous result.

The result is a non-overlapping union of the polyhedra.

INPUT

U   Union of polyhedra in the same dimension.
     Class: `PolyUnion`

W   Polyhedron to be summed with the union that is in the
     same dimension as the union.
     Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create two polyhedra that form unbounded and non-overlapping union.

```
P(1) = Polyhedron('V',[1 0; 0 0],'R',[0 -1;0.3 -1;-0.3 -1]);

P(2) = Polyhedron('V',[1 0; 0 0; 0 1; 1 1]);

U = PolyUnion('Set',P,'overlaps',false,'bounded',false)

PolyUnion in the dimension 2 with 2 polyhedra.
Properties of the union:
Overlaps: 0
Bounded: 0
Functions : none
```

Compute Minkowski addition for full-dimensional and bounded polyhedron W.

```
W = Polyhedron('lb',[-0.1,-0.2],'ub',[0.1,0.2]);
```

```
Uw = U + W
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Properties of the union:
Overlaps: 0
Bounded: 0
Functions : none
```

Plot the new union `Uw`

```
U.plot('color','lightblue'),hold on,Uw.plot
```
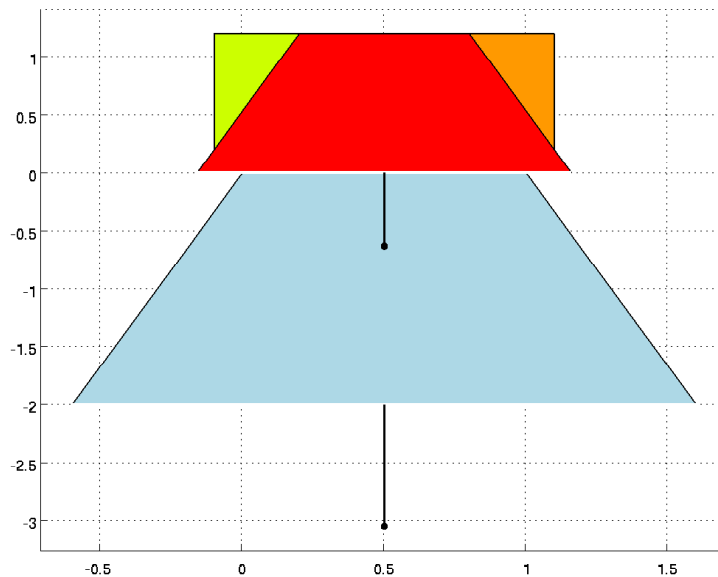

Compute Minkowski addition for low-dimensional and bounded polyhedron S.

```
S = Polyhedron('lb',[0,-0.2],'ub',[0,0.2]);
```

```
Us = U + S
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Properties of the union:
Overlaps: 0
Bounded: 0
Functions : none
```

Plot the new union `Us`

`U.plot('color','lightgreen'),hold on,Us.plot`
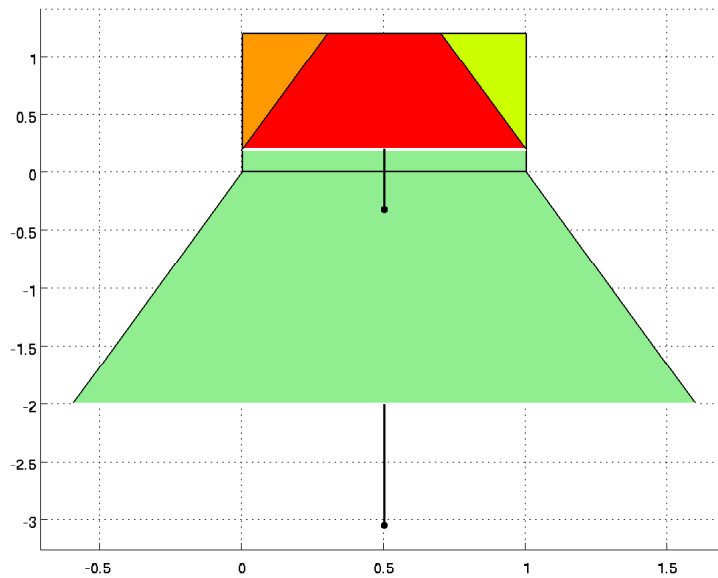
SEE ALSO

convexHull, minus

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

CONVEXHULL

Computes the convex hull for union of polyhedra

SYNTAX

```
H = U.convexHull
```

```
H = convexHull(U)
```

DESCRIPTION

The convex hull of the union of polyhedra is defined as the minimal convex set that contains all polyhedra.
Note that computation of convex hull is an expensive operation, therefore the result is stored internally under `Internal.convexHull` which can be accessed.

INPUT

U   Union of polyhedra in the same dimension.
Class: `PolyUnion`

OUTPUT

H   Convex hull of the polyhedra contained in the union
Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create 2 random V-polyhedra.

```
P(1) = 5*ExamplePoly.randVrep;
```

```
P(2) = 5*ExamplePoly.randVrep;
```

Create the union of polyhedra without specifying any properties.

```
U = PolyUnion('Set',P)
```

```
PolyUnion in the dimension 2 with 2 polyhedra.
Functions : none
```

Compute the convex hull

```
H = U.convexHull
```

```
Polyhedron in R^2 with representations:
H-rep (irredundant) : Inequalities   6 | Equalities   0
V-rep (irredundant) : Vertices   6 | Rays   0
Functions : none
```

Plot the union and the convex hull

```
U.plot; hold on; H.plot('wire',true,'linewidth',3,'linestyle','--')
```
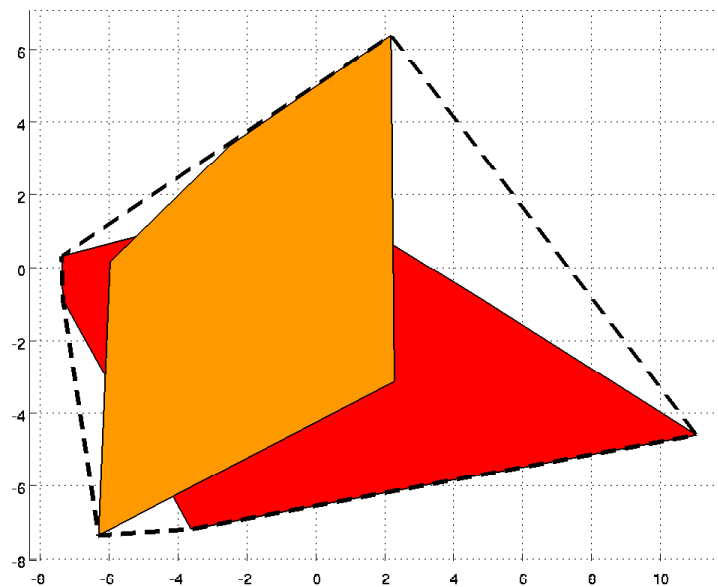

SEE ALSO

isConvex, merge, reduce

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
     mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
     mailto:herceg@control.ee.ethz.ch

PLOT

Plot the union of polyhedra.

SYNTAX

```
h = plot(U, 'Prop1', value1, 'Prop2', value2)

h = U.plot('Prop1', value1, 'Prop2', value2)

h = plot(U1, 'Prop1', value1, ..., U2, 'Prop2', value2, ...)
```

DESCRIPTION

Plot the union of polyhedra up to dimension three.
Figure properties, such as color, line width, etc, can be specified with "Property" - "Value" pairs.

INPUT

U	`PolyUnion` object that contains polyhedra in the same dimension. Class: `PolyUnion`
Prop1	Specification of figure properties. Class: `char` Allowed values:

                            Color The color of the plot specified by real RGB vector or a string name of the color (e.g. 'gray').

                            Wire Highlight or not the edges of the set. Default is false.

                            LineStyle Specify the type of the line to plot edges of the set. Accepted values are '-',':','-.','–', and'none'.

                            LineWidth Specify the width of the line. Default is 1.

                            Alpha Transparency of the color. The value must be inside [0,1] interval. Default is 1.

                            Marker Type of markings to use. Allowed values are ".", "o", "x", "+", "*", "s", "d", "v", "^", "¡", "¿", "p", "h" or "none". Default is "none".

                            MarkerSize The size of the marker. Default is 6.

                            ColorMap Color map given either as a M-by-3 matrix, or as a string. Default is 'mpt'. Other available options are 'hsv', 'hot', 'gray', 'lightgray', 'bone', 'copper', 'pink', 'white', 'flag', 'lines', 'colorcube', 'vga', 'jet', 'prism', 'cool', 'autumn', 'spring', 'winter', 'summer'.

                            ColorOrder Either 'fixed' for fixed ordering of colors, or 'random' for a random order. Default is 'fixed'.

                            ShowIndex This option is valid only for bounded polyhedra in 2D. If true, display an index of the plotted element. The default choice is false.

value1	Corresponding value to `Prop1`.

OUTPUT

   h   Handle related to graphics object.
       Class: `handle`

EXAMPLE(s)

Example 1

Plot union of three polyhedra in 2D.
Define the polyhedra

```
P(1) = Polyhedron('V',[-2,0;-1,1;-1,-1]);
```

```
P(2) = Polyhedron('lb',[-1,-1],'ub',[1,1]);
```

```
P(3) = Polyhedron('V',[2,0;1,1;1,-1]);
```

Create non-overlapping union, full-dimensional and bounded

```
U = PolyUnion('Set',P,'Overlaps',false,'FullDim',true,'Bounded',true)
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Properties of the union:
Overlaps: 0
Bounded: 1
FullDim: 1
Functions : none
```

Plot the union with 0.4 transparency and the lines in width 2.

```
U.plot('Alpha',0.4,'LineWidth',2,'LineStyle','-.');
```
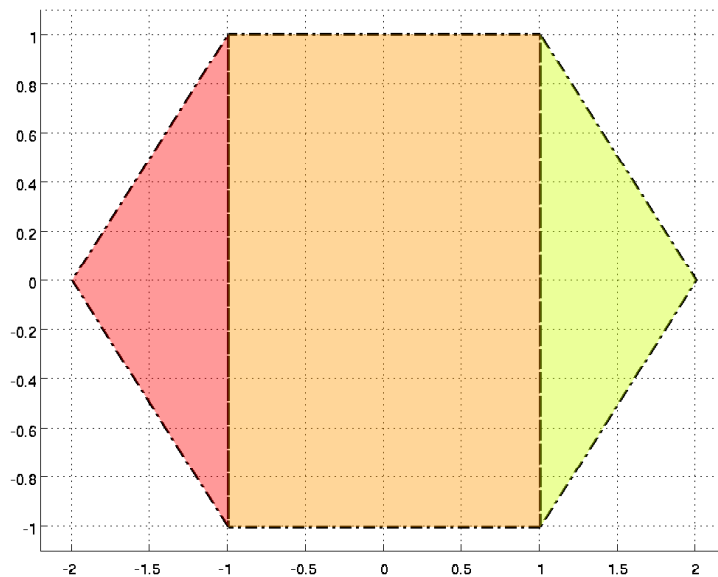
SEE ALSO

fplot

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

FEVAL

Evaluates a given function defined over a union of polyhedra.

SYNTAX

```
fval = U.feval(x)

fval = U.feval(x, function_name)

[fval, feasible, idx, tb_value] = U.feval(x, function_name)

[fval, feasible, idx, tb_value] = feval(U, x, function_name)
```

DESCRIPTION

Evaluates function for a given value of the point `x` over the union of polyhedra `U` characterized by the name `function_name`. If the string `function_name` is omitted, it is assumed that only one function is attached to the union. The dimension of the vector `x` must be the same as the dimension of the `PolyUnion`. If the point lies outside of the union, the result is `NaN`. Notes:

- `U` must be a single union. Arrays of unions are not accepted. Use `array.forEach(@(e) e.feval(...))` to evaluate arrays of unions.

- `function_name` must refer to a single function. If omitted, `U.feval(x)` only works if the union has a single function.

Outputs

1. `x` is not contained in any polyhedron of the union:

   `fval` = $(m \times 1)$ vector of `NaNs`, where m is the range of the function, `feasible = false`, `idx = []`, `tb_value = []`.

2. `x` is in a single polyhedron:

   `fval` = $(m \times 1)$ vector of function values, `feasible = true`, `idx` = index of the set which contains x, `tb_value = []`.

3. `x` is contained in multiple polyhedra (either at the boundary or in strict interior if there are overlaps), no tie-breaking (default):

   `fval` = $(m \times j)$ matrix of function values (j denotes the number of polyhedra which contain x), each column contains the value of `function_name` in the corresponding set, `feasible = true`, `idx` = $(1 \times j)$ vector of indices of polyhedra which contain x, `tb_value = []`.

4. `x` is contained in multiple polyhedra (either at the boundary or in strict interior if there are overlaps), tie-breaking enabled (see below):

   `fval` = $(m \times 1)$ vector containing the function value in the polyhedra in which value of the tie-breaking function is smallest (if there are multiple polyhedra with the same tie-breaking value, the first such set is considered), `feasible = true`, `idx` = index of

the polyhedron which contains x and, simultaneously, has the **smallest** value of the tie-breaking function, tb_value = scalar value of the tie-breaking function in a polyhedron indexed by idx.

Tie-breaking

The purpose of tie-breaking is to automatically resolve situations where the evaluation point x is contained in multiple polyhedra. With tie-breaking enabled PolyUnion/feval() evaluates the tie-breaking function to decide which polyhedron containing x should be used for evaluation of the original function.

The tie-breaking function can be specified by U.feval(x, 'tiebreak', tb_fun), where tb_fun can be either a string or a function handle. A string value must refer to an another function which exists in the union U.

A typical case where tie-breaking is useful is evaluation of discontinuous MPC feedback laws:
uopt = U.feval(x, 'primal', 'tiebreak', 'obj')
Here, if x is contained in multiple polyhedra, then the function primal is only evaluated in the polyhedron which contain x and simultaneously has the **smallest** value of the tie-breaking function obj.

A special case of tie-breaking is the "first-set" rule where we are only interested in evaluating a function in the first polyhedron which contains x (despite the fact there can be multiple such sets). This is achieved by
fval = U.feval(x, 'function_name', 'tiebreak', @(x) 0)
Notes:

- Tie-breaking functions must be scalar-valued.

- No tie-breaking is done by default.

Evaluation in particular polyhedra

fval = U.feval(x, 'myfun', 'regions', indices) evaluates function myfun over all polyhedra indexed by indices. The output fval is always an $(m \times j)$ matrix, where j is the cardinality of indices.

Note that once the regions option is enabled, PolyUnion/feval() will not perform point location. Instead, it will evaluate the function in all polyhedra indexed by indices, regardless of whether they contain x or not.

The regions option allows to quickly evaluate multiple functions as follows:
[first_value, idx] = U.feval(x, 'first_function')
second_value = U.feval(x, 'second_function', 'regions', idx)
In the second call, PolyUnion/feval will only evaluate second_function in polyhedra specified by the indices option, hence skipping expensive point location.

INPUT

U	Union of polyhedra in the same dimension.
	Class: PolyUnion

x	A point at which the function should be evaluated. The point must be given as a column real vector with the same dimension as the PolyUnion.

Class: `double`

function_name   Name of the function to evaluate. The string must match one of the stored function names. If there is only one function attached, this argument can be omitted.
Class: `char`

OUTPUT

fval   Function value at the point `x` over the `PolyUnion U`.
Class: `double`

feasible   Logical value indicating if the point `x` is contained in the union or not.
Class: `logical`
Allowed values: 1
                  0

idx   Vector of indices of polyhedra that contain the point `x`.
Class: `double`

tb_value   Value of the tie-breaking function if the point belongs to multiple polyhedra.
Class: `double`

EXAMPLE(s)

Example 1

PWA function over polyhedral complex.
Define one bounded polyhedron in dimension 3.

```
P = ExamplePoly.randVrep('d',3);
```

Triangulate the polyhedron

```
T = P.triangulate;
```

Add linear function to each polyhedron T in the array under the name "a".

```
T.addFunction(AffFunction([1 -2 0]),'a');
```

For each polyhedron in the array add second function 'b'.

```
for i=1:length(T),T(i).addFunction(AffFunction(rand(1,3),rand(1)),'b'); end
```

Create union out of polyhedra T with some properties.

```
U = PolyUnion('Set',T,'Convex',true,'Bounded',true,'Overlaps',false);
```

Evaluate the PWA function "a" for a point inside the polyhedron P.

```
x = P.interiorPoint.x;
```

```
y = U.feval(x,'a')
```

```
y =

-0.520345123015406
```

Evaluate function a with respect to a tie-breaking function 'b'

```
yn = U.feval(x,'a','tiebreak','b')
```

```
yn =

-0.520345123015406
```

SEE ALSO

fplot

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

TOC

Export of PWA/PWQ function to C-code

SYNTAX

```
controller.toC('function')
```

```
controller.toC('function','filename')
```

```
controller.toC('function','filename','tie_break_fcn')
```

DESCRIPTION

The function `toC()` exports given piecewise affine (PWA) or piecewise quadratic (PWQ) function to C-language including a sequential evaluation routine. The PWA/PWQ function must be attached to the `PolyUnion` object.

If the file name is not provided, the default output name is `mpt_getInput`.

The export routine generates two files on the output:

- `mpt_getInput.c` - which contains the PWA/PWQ function including the sequential search

- `mpt_getInput_mex.c` - mex interface for evaluation in Matlab

The file `mpt_getInput_mex` can be compiled inside Matlab and used for fast evaluation of PWA/PWQ function. The compilation is invoked by `mex` routine as follows:

```
mex mpt_getInput_mex
```

The PWA/PWQ function can be exported using the tie-break option if the function is multiple valued. The tie-breaking option determines which value of PWA/PWQ function will be evaluated based on the selecting the minimum in the tie-breaking function. In this case, the tie-breaking function must be attached to the `PolyUnion` object as well. If no tie-breaking function is provided, the first found value in the sequential search of PWA/PWQ function is evaluated.

The function `toC()` can export the floating point numbers to single or double precision. The default setting is `double` but this can be modified in global options

```
modules.geometry.unions.PolyUnion.toC.
```

INPUT

function	Name of the attached PWA/PWQ function to export. Class: `char`
filename	Base name of the file to be generated. Class: `char`

101

`tie_break_fcn`   Name of the attached scalar PWA/PWQ function to be used in tie-breaking case.
Class: `char`

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

LE

Test if a union of polyhedra is contained inside another union.

SYNTAX

`U1 <= U2`

DESCRIPTION

Check if the union of polyhedra `U1` is a non-strict subset of the union `U2`. The result it the logical statement if `U1 <= U2` and false otherwise.

INPUT

- `U1`   Union of polyhedra in the same dimension.
  Class: `PolyUnion`

- `U2`   Union of polyhedra in the same dimension.
  Class: `PolyUnion`

OUTPUT

- `tf`   True if `U1 <= U2` and false otherwise.
  Class: `logical`
  Allowed values: `true`
  `false`

EXAMPLE(s)

Example 1

Consider a rectangle in 2D.

`rectangle = Polyhedron('lb',[-2;-1],'ub',[4;5]);`

Cut the the rectangle into pieces by extracting an inner part.

`T = rectangle \ (0.2*rectangle);`

Create an union of these pieces.

`U1 = PolyUnion('Set',T,'FullDim',true,'Connected',true,'Convex',false,'Overlaps',false,'Bounded`

`U1.plot`
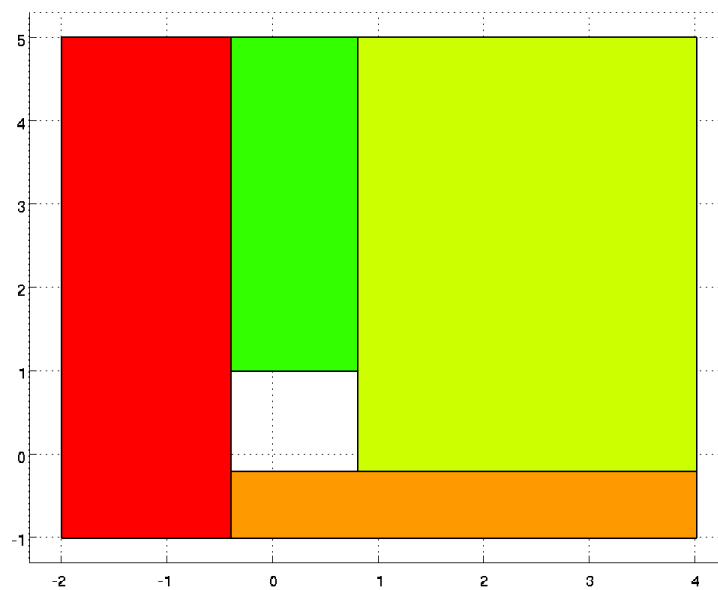
The union `U1` is contained inside the rectangle, so we verify that this statement is true.

```
U2 = PolyUnion(rectangle);

U1 <= U2
```

```
ans =

1
```

The inner rectangle is not contained inside the union.

```
U3 = PolyUnion(0.2*rectangle);

U1 <= U3
```

```
ans =

0
```

SEE ALSO

ge

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

GE

Test if a union of polyhedra contains another union.

SYNTAX

```
U1 >= U2
```

DESCRIPTION

Check if the union of polyhedra `U1` is a non-strict superset of another union `U2`. The result it the logical statement if `U1 >= U2` and false otherwise.

INPUT

U1   Union of polyhedra in the same dimension.
     Class: `PolyUnion`

U2   Union of polyhedra in the same dimension.
     Class: `PolyUnion`

OUTPUT

tf   True if `U1 >= U2` and false otherwise.
     Class: `logical`
     Allowed values: `true`
                       `false`

EXAMPLE(s)

Example 1

Consider an union of a rectangle and a triangle.

```
rectangle = Polyhedron('lb',[0;0],'ub',[2;1]);

triangle = Polyhedron([3,0;4,4; 6,2]);

U1 = PolyUnion([rectangle,triangle]);

U1.plot
```
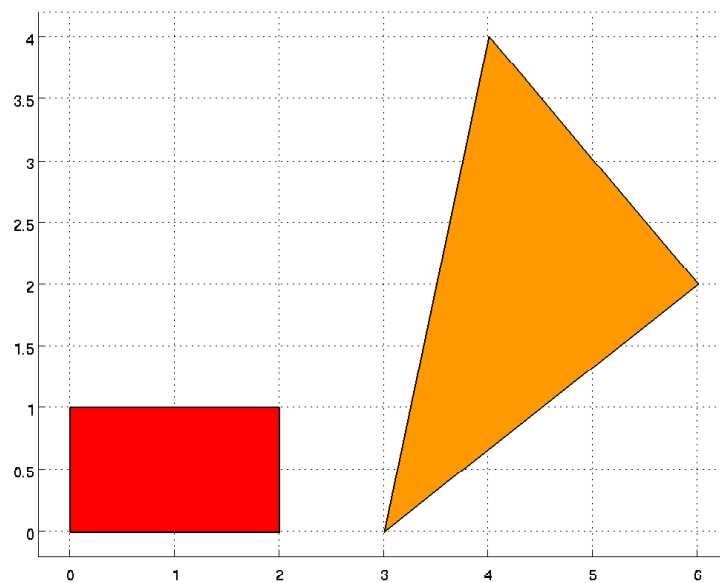
The union `U1` is located in the positive orthant, so we check if this is true.

```
U2 = PolyUnion(Polyhedron('lb',[0;0]));
```

```
U2 >= U1
```

```
ans =

1
```

SEE ALSO

le

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

REDUCE

Reduces the overlapping union to minimal number of sets.

SYNTAX

```
kept = U.reduce
```

```
kept = reduce(U)
```

DESCRIPTION

Simplifies the union of polyhedra by removing those regions who are completely covered by others. Note that this algorithm is valid only for overlapping unions.

INPUT

U   Union of polyhedra in the same dimension.
Class: `PolyUnion`

kept   Vector of logical indices indicating if the region is non-redundant.
Class: `logical`
Allowed values: `true`
                   `false`

EXAMPLE(s)

Example 1

Create a random H-polyhedron that contains the origin.

```
P = 5*ExamplePoly.randHrep;
```

Create another polyhedron that is contained within `P`

```
Q = 0.8*P;
```

Create another random polyhedron.

```
R = ExamplePoly.randVrep+[1;5];
```

Create the union of polyhedra without specifying any properties.

```
U = PolyUnion([P,Q,R])
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Functions : none
```

Plot the union

```
U.plot
```
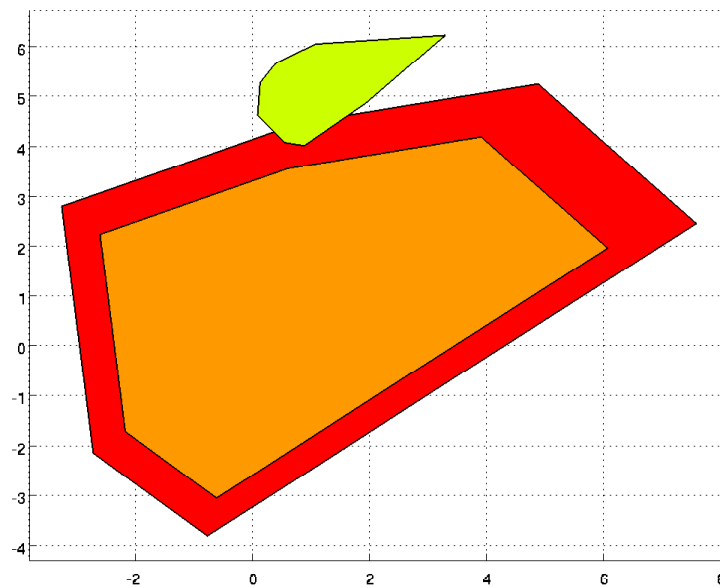
This union can be reduced to only regions P and R because Q is completely covered by P.

```
U.reduce
```

```
ans =

1     0     1
```

We can see that the union has now only 2 polyhedra. Plot the reduced union

```
U.plot
```
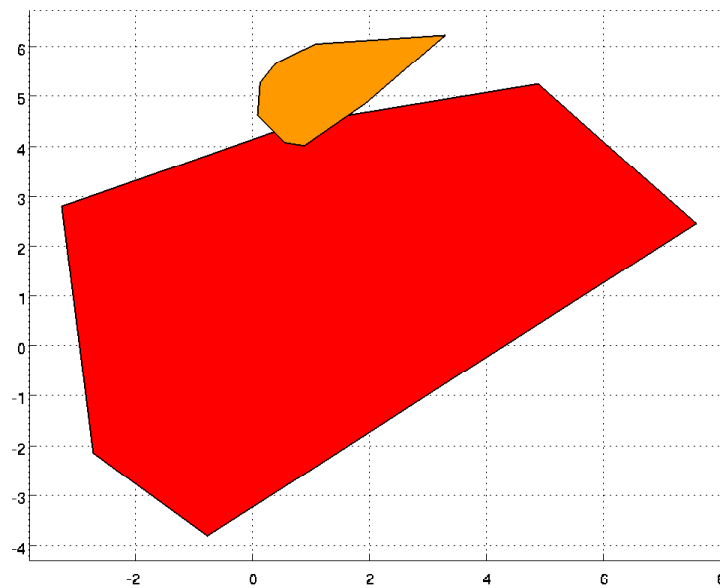
SEE ALSO

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

EQ

Returns true if the set covered by unions of polyhedra U_1 is the same as the set covered by union U_2 and false otherwise.

SYNTAX

```
ts = U1.eq(U2)

ts = U1 == U2
```

DESCRIPTION

Returns true if the union of polyhedra in the same dimension U1 covers the same space as the union U2 which is in the same dimension. The result is a logical statement if $U_1 = U_2$ holds true or not. The polyunion U1 can be an array, whereas U2 is restricted to be a single polyunion object.

INPUT

U1   Union of polyhedra in the same dimension.
Class: `PolyUnion`

U2   Union of polyhedra in the same dimension as U1.
Class: `PolyUnion`

OUTPUT

ts   True if U1 == U2 and false otherwise.
Class: `logical`
Allowed values: `true`
                  `false`

EXAMPLE(s)

Example 1

Create union of two polyhedra in the same dimension

```
P(1) = ExamplePoly.randHrep;

P(2) = ExamplePoly.randHrep('ne',1);

U1 = PolyUnion('Set',P,'Bounded',false)


PolyUnion in the dimension 2 with 2 polyhedra.
Properties of the union:
Bounded: 0
Functions : none
```

Create a copy of this polyunion

```
U2 = U1.copy
```

```
PolyUnion in the dimension 2 with 2 polyhedra.
Properties of the union:
Bounded: 0
Functions : none
```

Obviously, the polyunions cover the same space. We can verify that using equality test:

```
U1 == U2
```

```
ans =

1
```

SEE ALSO

copy, join

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

JOIN

Merges arrays of polyunions to one polyunion object.

SYNTAX

```
Un = U.join
```

```
Un = join(U)
```

DESCRIPTION

Merging of array of `PolyUnion` objecs in the same dimension to single `PolyUnion` object. If the union has `Bounded` and `FullDim` properties set as true, then all polyhedra must be bounded and full-dimensional.

INPUT

U   Array of `PolyUnion` objects in the same dimension.
    Class: `PolyUnion`

OUTPUT

Un   Single `PolyUnion` object.
     Class: `PolyUnion`

EXAMPLE(s)

Example 1

Create a partition by triangulation of polyhedron P.

```
P = 5*ExamplePoly.randVrep('d',3);
```

Triangulate the polyhedron

```
T = P.triangulate;
```

Create the array of unions of polyhedra and specifying some properties.

```
U(1) = PolyUnion('Set',T(1:4),'fulldim',true,'bounded',true);
```

```
U(2) = PolyUnion('Set',T(5:6));
```

```
U(3) = PolyUnion('Set',T(7:end),'bounded',true);
```

All unions are in the same dimension and all polyhedra are bounded and full-dimensional, therefore we can merge the unions.

```
Un = U.join
```

```
PolyUnion in the dimension 3 with 11 polyhedra.
Properties of the union:
Bounded: 1
FullDim: 1
Functions : none
```

SEE ALSO

merge, isFullDim, isBounded

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

COPY

Create a new copy of the PolyUnion object.

SYNTAX

```
Un = U.copy
```

```
Un = copy(U)
```

DESCRIPTION

Create a new copy of the `PolyUnion` object. Any change of data in the new object `Un` will not affect the data contained in `U` object.

INPUT

U   Union of polyhedra in the same dimension.
     Class: `PolyUnion`

EXAMPLE(s)

Example 1

Create union of two polyhedra P.

```
P(1) = 5*ExamplePoly.randVrep;
```

```
P(2) = -P(1);
```

Create the union of polyhedra by specifying some properties.

```
U = PolyUnion('Set',P,'convex',false,'overlaps',true);
```

Create the copy of the union `U`

```
Un = U.copy;
```

We can do some operations on the new object `Un`, without affecting the data stored in the old object`U`. For instance, querying for full-dimensionality:

```
Un.isFullDim
```

```
ans =

1
```

The old object was not affected by this query

```
U
```

```
PolyUnion in the dimension 2 with 2 polyhedra.
Properties of the union:
Convex: 0
Overlaps: 1
Functions : none
```

Whereas in the new object was the property "FullDim" has been changed.

```
Un
```

```
PolyUnion in the dimension 2 with 2 polyhedra.
Properties of the union:
Convex: 0
Overlaps: 1
FullDim: 1
Functions : none
```

SEE ALSO

PolyUnion

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ISFULLDIM

Test if the union is built from full-dimensional polyhedra.

SYNTAX

```
ts = U.isFullDim
```

```
ts = isFullDim(U)
```

DESCRIPTION

Return true if all polyhedra in the union `U` are full-dimensional and false otherwise. Once this method has been called, the information about the full-dimensionality can be retrieved from `U.Internal.FullDim` property.

INPUT

U   Union of polyhedra in the same dimension.
    Class: `PolyUnion`

OUTPUT

ts   True if all polyhedra in the union are full-dimensional
    and false otherwise.
    Class: `logical`
    Allowed values: `true`
                     `false`

EXAMPLE(s)

Example 1

Union of full-dimensional and low-dimensional polyhedron.

```
P(1) = Polyhedron('lb',[0;0],'ub',[1;1]);
```

```
P(2) = Polyhedron('lb',[0;0],'ub',[0;1]);
```

Create union

```
U = PolyUnion(P)
```

```
PolyUnion in the dimension 2 with 2 polyhedra.
Functions : none
```

Check if the union is full-dimensional

```
U.isFullDim
```

```
ans =

0
```

The full-dimensionality property can be retrieved from

`U.Internal.FullDim`

```
ans =

0
```

SEE ALSO

`isConvex`, `isOverlapping`, `isConnected`, `isBounded`

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ISCONVEX

Test if the union of polyhedra is convex.

SYNTAX

```
ts = U.isConvex
```

```
ts = isConvex(U)
```

DESCRIPTION

Return true if the union `U` of polyhedra is convex and false otherwise. Once this method has been called, the information about the convexity can be retrieved from `U.Internal.Convex` property. **Note that this function is very computationally demanding and is suitable for unions with small number of polyhedra.**

INPUT

   U   Union of polyhedra in the same dimension.
       Class: `PolyUnion`

OUTPUT

   ts   True if union of polyhedra is convex and false otherwise.
        Class: `logical`
        Allowed values: `true`
                          `false`

EXAMPLE(s)

Example 1

Create a random polyhedron in V-representation.

```
P = 10*ExamplePoly.randVrep;
```

Triangulate polyhedron to get a partition.

```
T = P.triangulate
```

```
Array of 4 polyhedra.
```

Create union out of these polyhedra without specifying the properties

```
U = PolyUnion(T)
```

```
PolyUnion in the dimension 2 with 4 polyhedra.
Functions : none
```

Check if the union is convex

`U.isConvex`

`ans =`

`1`

We can plot the union to see that it is truly convex.

`U.plot`
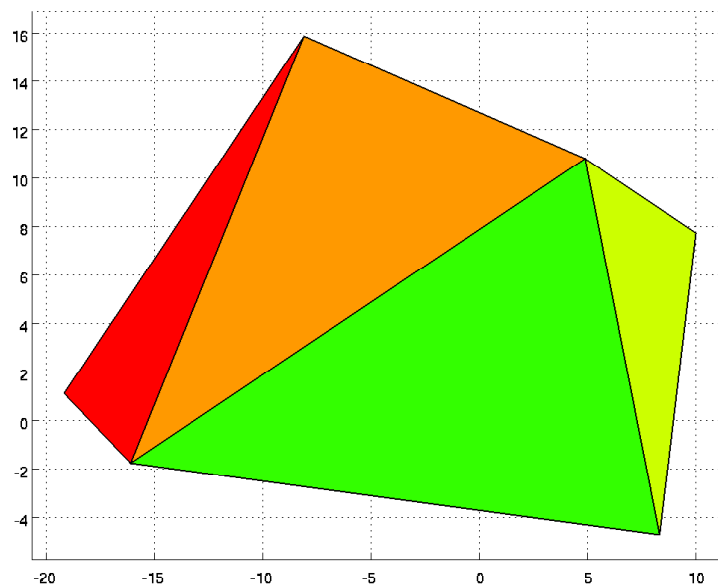
SEE ALSO

isConnected, isOverlapping, isFullDim, isBounded

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

CONTAINS

Test if a point is contained inside the union of polyhedra in the same dimension.

SYNTAX

```
[isin, inwhich, closest] = contains(U, x, fastbreak)

[isin, inwhich, closest] = U.contains(x)

[isin, inwhich, closest] = U.contains(x, fastbreak)
```

DESCRIPTION

Check if the point `x` is contained in any of the polyhedra stored in the union `U`. The result it the logical statement if $x \in U$ and false otherwise. If the point is contained inside the union, indices of the corresponding polyhedra in which the point lie are returned. If the point does not lie in the union, the index of the region with the least distance to the point x is returned.

INPUT

U	Single `PolyUnion` object that holds sets polyhedra in the same dimension. Class: `PolyUnion`
x	A point in the same dimension as the union. Class: `double`
fastbreak	Do a quick stop in the consecutive search when `x` is contained in the first polyhedron it founds. Class: `logical` Allowed values: `true` `false` Default: `false`

OUTPUT

isin	True if $x \in U$ and false otherwise. Class: `logical` Allowed values: `true` `false`
inwhich	Indices of sets that contain x. If the `fastbreak` option is turned on, a single index is returned. Class: `double`
closest	If the point is not contained inside the union, this output indicates the index of the set that is the closest to the point `x`. Note: since this computation is expensive, do not ask for the third output unless you really need it.

Class: `double`

EXAMPLE(s)

Example 1

Create five polyhedra in 3D.

`for i=1:5,P(i) = ExamplePoly.randVrep('d',3)+rand(3,1); end`

Create an union of these sets without specifying any properties.

`U = PolyUnion(P);`

Check if the point $x = (1, -1, 0)$ is contained in the union.

`x = [1; -1; 0];`

`[isin,inwhich,closest] = U.contains(x)`

`isin =`

`0`

`inwhich =`

`Empty matrix: 1-by-0`

`closest =`

`5`

SEE ALSO

`feval`

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

REMOVEALLFUNCTIONS

Remove all functions that are associated to this union of polyhedra.

SYNTAX

```
U = U.removeAllFunctions

U = removeAllFunctions(U)
```

DESCRIPTION

Removes all `Function` objects associated to this union of polyhedra.

INPUT

  U   Object of the `PolyUnion` class that holds polyhedra in
       the same dimension.
       Class: `PolyUnion`

OUTPUT

  U   Modified object of `PolyUnion` class without the function
       handles.
       Class: `PolyUnion`

EXAMPLE(s)

Example 1

Create three polyhedra and two functions to each one of them.

```
for i=1:3,P(i)=ExamplePoly.randHrep; end

P.addFunction(AffFunction([-1,5],-1/8),'affine_function');

P.addFunction(QuadFunction(eye(2),[-2,4],0.6),'quadratic_function');
```

Create the union of polyhedra

```
U = PolyUnion(P)

PolyUnion in the dimension 2 with 3 polyhedra.
Functions : 2 attached "affine_function","quadratic_function"
```

Remove all functions handles from the union

```
U.removeAllFunctions

PolyUnion in the dimension 2 with 3 polyhedra.
Functions : none
```

SEE ALSO

addFunction, removeFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ADD

Insert Polyhedron to PolyUnion object.

SYNTAX

```
U = add(U,P)
```

```
U.add(P)
```

DESCRIPTION

Insert the `Polyhedron` object inside the existing `PolyUnion` object. The polyhedron P can be given as an array. Each element of the array is stored under `PolyUnion.Set` property which can be accessed later for usage.

Any polyhedron P that is empty, it is not added to the union.

If the `PolyUnion` object has been created with some properties, such as: `Convex`, `Overlaps`, `Connected`, `Bounded`, and `FullDim`, then any polyhedron to be added is checked for this property. If the union created by merging of the existing polyhedra and the new polyhedra P does not satisfy any of these properties, an error message is thrown.

INPUT

U   The object of the `PolyUnion` class.
    Class: `PolyUnion`

P   A `Polyhedron` or an array of polyhedra to be added to
    the union.
    Class: `Polyhedron`

OUTPUT

U   Union of the sets.
    Class: `Union`

EXAMPLE(s)

Example 1

Construct `PolyUnion` object by triangulating a zonotope.

```
Z = ExamplePoly.randZono;
```

Triangulate the zonotope `Z`.

```
T = Z.triangulate
```

```
Array of 8 polyhedra.
```

We know that the polyhedra `T` are not overlapping, bounded and in full-dimension.
We can create a union of first four polyhedra

```
U = PolyUnion('Set',T(1:4),'Overlaps',false,'Bounded',true,'FullDim',true)
```

```
PolyUnion in the dimension 2 with 4 polyhedra.
Properties of the union:
Overlaps: 0
Bounded: 1
FullDim: 1
Functions : none
```

Plot the union

```
U.plot
```
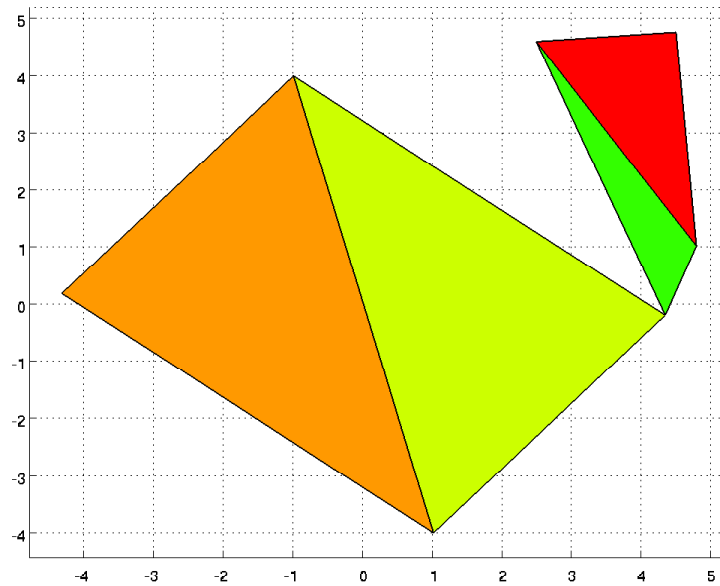

To add the remainining polyhedra is not a problem because the properties are fulfilled.

```
U.add(T(5:end))
```

```
PolyUnion in the dimension 2 with 8 polyhedra.
Properties of the union:
Overlaps: 0
Bounded: 1
FullDim: 1
Functions : none
```

We can plot the union

126

`U.plot`

SEE ALSO
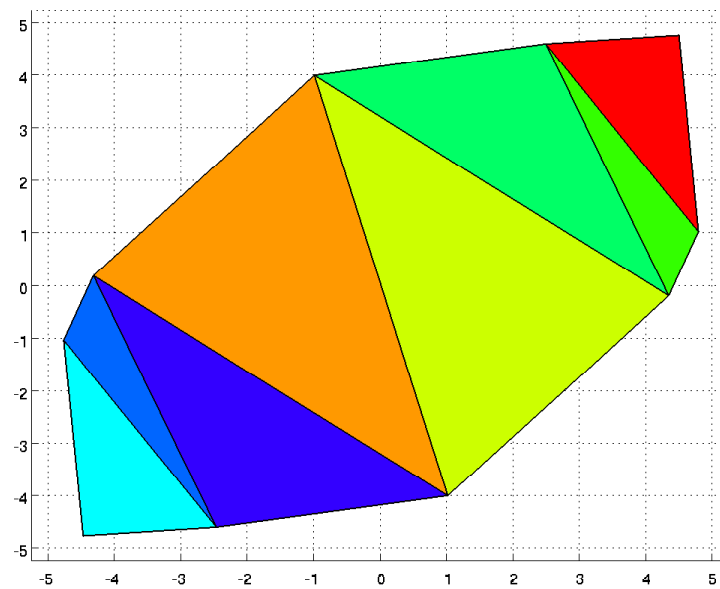
AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

LOCATEPOINT

Implementation of a graph search algorithm for a point location problem.

SYNTAX

```
[index, details] = locatePoint(U,x)
```

DESCRIPTION

Return an index of region in the polyunion U where the point x is contained. If the point lies outside of the union of polyhedra, the output is empty.

The polyunion U must be returned from PLCP solver with an appropriate adjacency list, otherwise the method is not applicable. The best performance is achieved if the region exploration in PLCP solver has been done using breadth-first search (BFS) where the regions have a leveled structure and the first region lies in the middle of the partition. In this case, the graph traversal algorithm proceeds through increasing levels outside from the first region until the desired region is found.

The set membership operation depends on the settings of absolute tolerance that can be changed in Options settings.

INPUT

U   Union of polyhedra returned from PLCP solver with an
included adjacency list.
Class: PolyUnion

x   A point in the same dimension as PolyUnion given as
real column vector.
Class: double

OUTPUT

index   Index of a region where x is contained, otherwise an
empty output [].
Class: double

details   A structure with statistical information numerical performance of the graph traversal algorithm.
Class: struct
Allowed values: niter Number of iterations.
noper Total number of arithmetic operations (multiplications+summations+comparisons).
multiplications Multiplications count.
summations Summation count.
comparisons Comparisons count.

EXAMPLE(s)

Example 1

Generate random polytope P.

```
P = ExamplePoly.randVrep('d',3);
```

Formulate a parametric optimization problem in 2D over the polytope P.

```
problem = Opt('f',[1,-0.4,0.4],'A',P.A,'b',P.b,'pB',randn(size(P.H,1),2));
```

Solve the problem to get a polyunion with attached adjacency list.

```
res = problem.solve

mpt_plcp: 20 regions

res =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Get the interior point located in the last region.

```
last = res.xopt.Num

last =

20
```

```
p = res.xopt.Set(last).interiorPoint

p =

x: [2x1 double]
isStrict: 1
r: 916.361188799218
```

Verify using the graph traversal algorithm that the point lies in the last region.

```
index = locatePoint(res.xopt,p.x)

index =

20
```

129

SEE ALSO

contains, isInside

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ISBOUNDED

Test if a convex set is bounded.

SYNTAX

```
ts = S.isBounded
```

```
ts = isBounded(S)
```

DESCRIPTION

Return true if the convex set S is bounded and false otherwise. Empty set is considered as bounded.

INPUT

S   Any set derived from `ConvexSet` class, e.g. `YSet` or
    `Polyhedron`.
    Class: `ConvexSet`

OUTPUT

ts   True if the convex set P is bounded and false otherwise.
     Class: `logical`
     Allowed values: `true`
                     `false`

EXAMPLE(s)

Example 1

Describe positive orthant in 3D using YALMIP

```
x = sdpvar(3,1);
```

```
F = [x>=0];
```

```
S = YSet(x,F);
```

Check if the set is unbounded

```
S.isBounded
```

```
ans =
```

```
0
```

SEE ALSO

isEmptySet

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

HASFUNCTION

Returns true if the set contains given function name.

SYNTAX

```
ts = hasFunction(Set, FuncName)

ts = Set.hasFunction(FuncName)
```

DESCRIPTION

Returns true or false if the function stored under `FuncName` is attached to a convex `Set`.

INPUT

Set	Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `ConvexSet`
FuncName	Name of the function to test. Class: `char`

OUTPUT

ts	Logical statement if the set contains the function `FuncName` or not. Class: `logical`

EXAMPLE(s)

Example 1

The polyhedron contains two functions "a" and "b".

```
P = ExamplePoly.randHrep;

P.addFunction(AffFunction([1,-2]),'a');

P.addFunction(QuadFunction([0 0.4;-0.1 0.2]),'b');
```

Does the set P contain the function "b" ?

```
P.hasFunction('b')

ans =

1
```

SEE ALSO

listFunctions, addFunction, removeFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

DISTANCE

Computes the closest distance between the convex set and given point.

SYNTAX

```
ret = distance(Set,x)

ret = Set.distance(x)
```

DESCRIPTION

Compute the closest distance between the convex `Set` and given point `x` The approach is based on solving the optimization problem

$$\min_y \|x - y\|_2$$
$$\text{s.t.} \quad y \in \texttt{Set}$$

where x is the given point in the same dimension as the set and y is the point inside the `Set`. If the optimization terminated successfully, the output contains the distance `ret.d` and the points `ret.y`, `ret.x`. Otherwise, the output is empty. If the `Set` is an array of convex sets, the distance and the point `y` are returned in a cell arrays.

INPUT

Set	Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `ConvexSet`
x	A point given as a real vector with the same dimension as the convex set. Class: `double`

OUTPUT

ret	Structure that contains the information about the computed distance and the points x, y. Class: `struct`
ret.exitflag	Termination status from the related optimization problem. Class: `double`
ret.dist	Distance between the point `x` and the convex `Set`. Class: `double`
ret.y	The point that is contained in the convex `Set` and is closest to `x`.

Class: `double`

ret.x	The point `x` that was provided.
	Class: `double`

EXAMPLE(s)

Example 1

Construct a convex set by intersecting a circle and random linear inequalities.
Define variable first

```
z = sdpvar(2,1);
```

Define a set `S` using `YSet` class

```
options = sdpsettings('solver','sedumi','verbose',0);
```

```
S = YSet(z,[norm(z)<=1; randn(2)*z<=0.5*ones(2,1)],options);
```

Compute the distance to a point [-5;8]

```
x = [-5; 8];
```

```
d = S.distance(x)
```

```
d =

exitflag: 1
dist: 8.43398113093422
x: [2x1 double]
y: [2x1 double]
```

We can plot the set and the points `x`, `y`

```
S.plot('alpha',0.5,'color','green');
hold on;  text(x(1),x(2),'x');
axis([-3 2 -2 3]); text(d.y(1),d.y(2),'x');
```

```
Plotting...
27 of 40
```

136
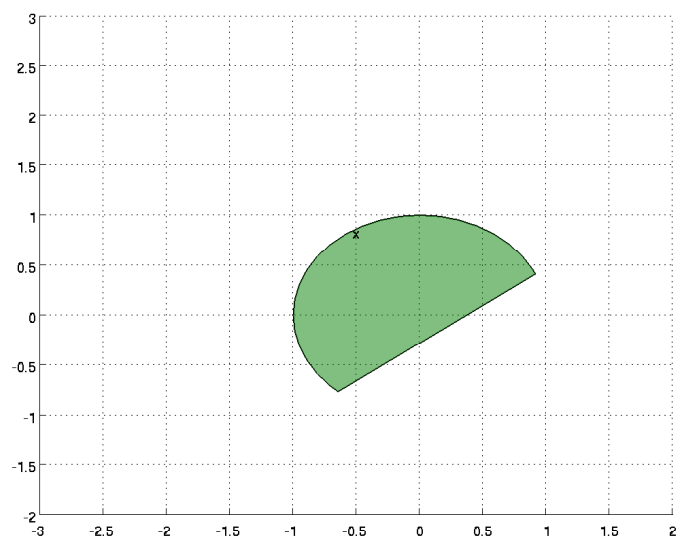
SEE ALSO

outerApprox, support, separate

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

GETFUNCTION

Extract function handle from a convex set.

SYNTAX

```
F = getFunction ( Set , FuncName )

F = Set . getFunction ( FuncName )
```

DESCRIPTION

Extract `Function` object that is attached to a convex `Set` under the name `FuncName`.

INPUT

Set	Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `ConvexSet`
FuncName	Name of the function to extract. Class: `char`

OUTPUT

F	Function object stored under the `FuncName` string. Class: `Function`

EXAMPLE(s)

Example 1

The polyhedron contains two functions "a" and "b".

```
P = ExamplePoly.randHrep;

P.addFunction(AffFunction([1,-2]),'a');

P.addFunction(QuadFunction([0 0.4;-0.1 0.2]),'b');
```

Extract the function "b" from the set

```
F = P.getFunction('b')
```

```
Quadratic Function: R^2 ->R^1
```

SEE ALSO

addFunction, removeFunction

AUTHOR(s)

VERTCAT

Vertical concatenation for convex set objecs.

SYNTAX

```
S = [S1; S2]

S = vertcat(S1,S2)
```

DESCRIPTION

Overloaded method for vertical concatenation of convex sets. It is not possible to concatenate objects of different type to the same array (e.g. `Polyhedron` and `YSet`). Similarly, it is not possible to concatenate into matrices, only vectors are allowed.

INPUT

S1   Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ...
     Class: `ConvexSet`

S2   Any object derived from the `ConvexSet` class that is of the same type as `S1`.
     Class: `ConvexSet`

OUTPUT

S   The array of the convex sets.
     Class: `ConvexSet`

EXAMPLE(s)

Example 1

We have two sets described in Yalmip.

```
x = sdpvar(2,1);

S1 = YSet(x,[x(1)-x(2)<=1]);

S2 = YSet(x,[-2*x(1)+3*x(2)>=2]);
```

Vertical concanation gives an array with 2 elements

```
S = [S1; S2]
```

```
Array of 2 YSets.
```

SEE ALSO

horzcat

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

REMOVEFUNCTION

Remove function associated to a convex set based on the function name.

SYNTAX

```
Set = removeFunction(Set,name)
```

```
Set.removeFunction(name)
```

DESCRIPTION

Removes the `Function` object `F` from the convex `Set` identified by the string `name`. The functions stored with the set can be found under `Func` property. The function name must match one of function names stored under this set that can be retrieved using `listFunctions` method.

INPUT

Set	Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `ConvexSet`
name	Name of the function to remove from the array. String must match one of the stored function names. For multiple names, specify `name` as a cell array of strings. Class: `double or char`

OUTPUT

Set	Modified object of `ConvexSet` class without the function that has been removed. Class: `ConvexSet`

EXAMPLE(s)

Example 1

Add two functions $f_1(x) = -x+1$ and $f_2(x) = x+2$ to a polyhedron $P = \{x \mid -4 <= x <= 5\}$
Define Polyhedron

```
P = Polyhedron('lb',-4,'ub',5);
```

Define the functions

```
f1 = AffFunction(-1,1); f2=AffFunction(1,2);
```

Add these functions to the polyhedron

```
P.addFunction(f1,'f1');
```

```
P.addFunction(f2,'f2');
```

Remove the function "f1" from the set

```
P.removeFunction('f1')
```

```
Polyhedron in R^1 with representations:
H-rep (redundant)   : Inequalities   2 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : 1 attached "f2"
```

Polyhedron no contains only 'f2' function

```
P.Func
```

```
ans =

[1x1 AffFunction]
```

SEE ALSO

ConvexSet, Function, QuadFunction, AffFunction, addFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

OUTERAPPROX

Computes outer bounding box of the convex set.

SYNTAX

```
B = outerApprox(S)
```

```
B = S.outerApprox
```

DESCRIPTION

Compute the smallest axis-aligned hypercube that contains this set. The lower and upper bounds of the hypercube are stored under `Internal` property, i.e. `Internal.lb` for lower bound and `Internal.ub` for upper bound.

INPUT

S   Any set derived from `ConvexSet` class, e.g. `YSet` or `Polyhedron`.
    Class: `ConvexSet`

OUTPUT

B   Bounding box B described as `Polyhedron` in H-representation.
    Class: `Polyhedron`

EXAMPLE(s)

Example 1

Describe circle in 2D using YALMIP

```
x = sdpvar(2,1);
```

```
F = [x'*x<=1];
```

```
S = YSet(x,F);
```

The bounding box for the circle is a cube with diameter 1.

```
B = S.outerApprox
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities  4 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : none
```

Plot the sets

```
S.plot; hold on; B.plot
```
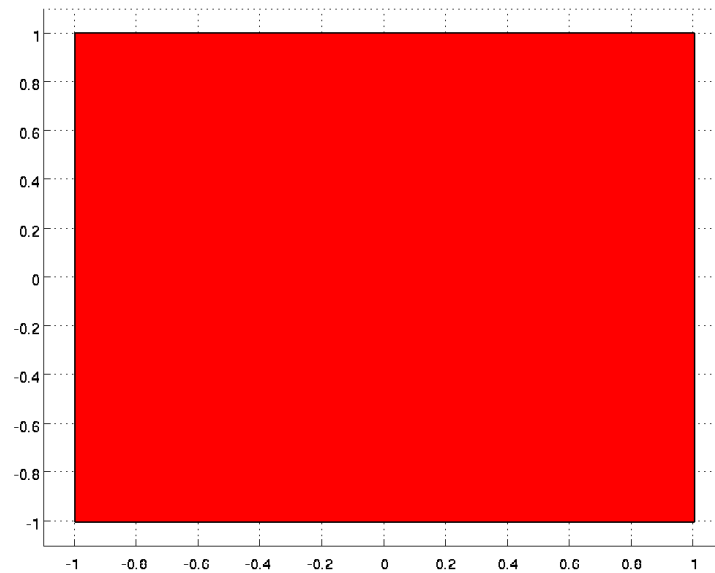

Example 2

We have a lower dimensional polyhedron in 3D.

```
P = Polyhedron('A',randn(9,3),'b',ones(9,1),'Ae',randn(1,3),'be',0.5);
```

Compute the bounding box

```
B = P.outerApprox;
```

Plot the sets such that the outer approximation is wired.

```
P.plot; hold on; B.plot('wire',true,'LineWidth',3)
```

SEE ALSO

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch
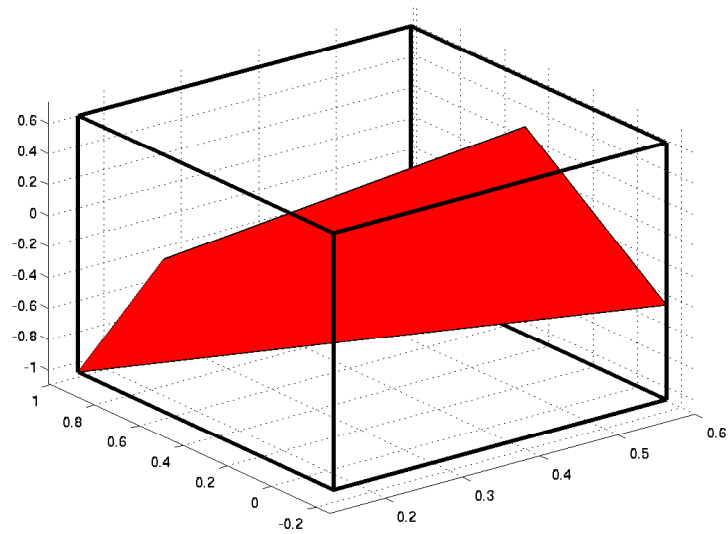
© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

FPLOT

Plot a single function over a convex set or over an array of convex sets.

SYNTAX

```
h = Set.fplot()

h = Set.fplot('name', 'Prop1', value1, 'Prop2', value2)

h = fplot(Set, 'name', 'Prop1', value1, 'Prop2', value2)
```

DESCRIPTION

Plot single function over a convex set. If there are more functions attached to a set, then the string **name** identifies the function to be plotted. If the function is vector valued, i.e. its range is greater than 1, than the first element of the function is plotted by default. For vector valued functions, use the **position** property to indicate that you want a different element of the function value to plot.

Figure properties, such as color, line width, etc, can be specified with "Property" - "Value" pairs.

INPUT

Set	Any object derived from the **ConvexSet** class, e.g. **Polyhedron**, **YSet**, ... Class: **ConvexSet**
name	If there are more functions attached to the set, this string indicates the **name** of the function to plot. Class: **char**
Prop1	Specification of figure properties. Class: **char**

Allowed values: `position` For vector valued functions, the `position` indicates which element of the function value to plot.

`Grid` With how many gridpoints to grid the circle/-sphere. Default is 20.

`Color` The color of the plot specified by real RGB vector or a string name of the color (e.g. 'gray');

`Wire` Highlight the edges of the set. Default is false.

`LineStyle` Specify the type of the line to plot edges of the set. Accepted values are '-',':','-.','–', and'none'.

`LineWidth` Specify the width of the line. Default is 1.

`Alpha` Transparency of the color. The value must be inside [0,1] interval. Default is 1.

`Contour` Add contour graph. Default is false.

`ContourGrid` With how many grid points to plot the contour graph. Default is 30.

`show_set` Plot the domain of the function. Default is false.

`showgrid` Show the grid inside the set. Default is false.

`ColorMap` Color map given either as a M-by-3 matrix, or as a string. Default is 'mpt'. Other available options are 'hsv', 'hot', 'gray', 'lightgray', 'bone', 'copper', 'pink', 'white', 'flag', 'lines', 'colorcube','vga', 'jet', 'prism', 'cool', 'autumn', 'spring', 'winter', 'summer'.

`ColorOrder` Either 'fixed' for fixed ordering of colors, or 'random' for a random order. Default is 'fixed'.

`value1` Assigns value to `Prop1`.

OUTPUT

`h` Handle related to graphics object.
Class: `handle`

EXAMPLE(s)

Example 1

Plot one function over one dimensional set.
The set is an interval [-1, 2]

```
P = Polyhedron('lb',-1,'ub',2);
```

Assign a quadratic function $f(x) = x^2 - x + 3$ under name "f" to the set `P`

```
Q = QuadFunction(1,-1,3);
```

```
P.addFunction(Q,'f');
```

Plot the function

`P.fplot`

Example 2

We have two linear functions "a", "b" over a convex set.
Construct the set first as the intersection of a circle and linear inequality

```
x = sdpvar(2,1);
```

```
F = [0.5*x'*x<=0.2; [-1,0.6]*x<=0.5];
```

```
S = YSet(x,F);
```

Construct quadratic and linear function.

```
Q = QuadFunction([1 -2;3 0.5],[-1,0],-1.4);
```

```
L = AffFunction([2.6,-0.2],0.7);
```

Add functions to the set with names "a" and "b".

```
S.addFunction(Q,'a');
```

```
S.addFunction(L,'b');
```

Plot the function "b" based on the function name

```
S.fplot('b');
```
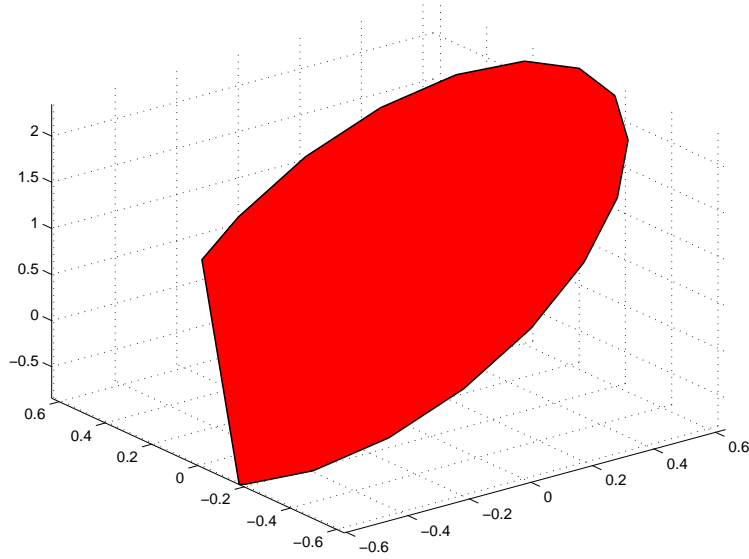
Example 3

We have three affine and vector valued functions over the set `S`.
Construct the set first as the intersection of a circle and linear inequality

```
x = sdpvar(2,1);
```

```
F = [0.5*x'*x<=0.2; [-3,0.4]*x<=0.5];
```

```
S = YSet(x,F);
```

Construct three affine functions and put them in an array.

```
A1 = AffFunction([-0.4,0.9; -1,0.3],[0.4; 0.7]);
```

```
A2 = AffFunction([0.7,-0.1; -2,0.7],[0.3; -0.1]);
```

```
A3 = AffFunction([0.7,-0.1; -0.6 5],[0.3; 0.2]);
```

add these functions to the set `S` with some names

```
S.addFunction(A1,'mon');
```

```
S.addFunction(A2,'wed');
```

```
S.addFunction(A3,'fri');
```

Plot the second element of the function "fri" in gray color.

```
S.fplot('fri','position',2,'Color','gray')
```

150
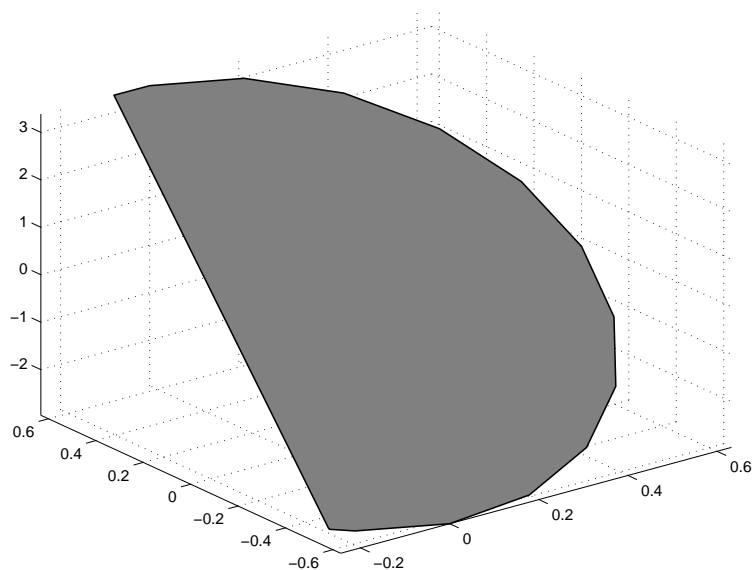
Plot the second element of the function "mon" and the underlying domain with some transparency.

`S.fplot('mon','position',2,'show_set',true,'alpha',0.6,'colororder','random')`
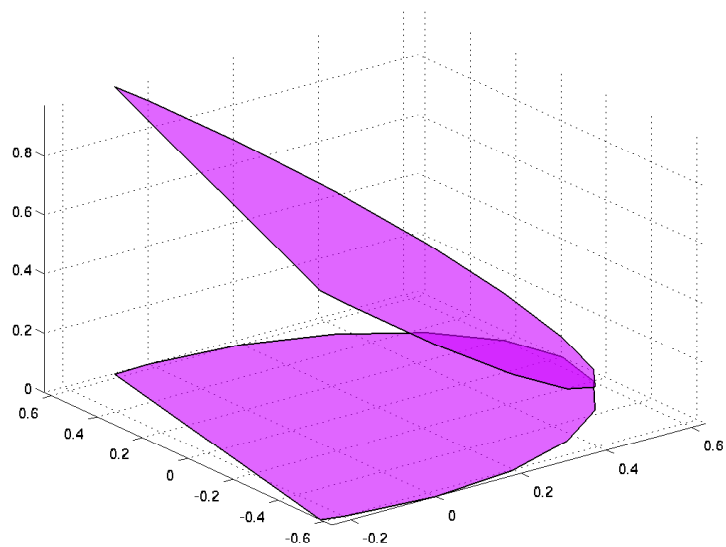
SEE ALSO

plot

151

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

ISEMPTYSET

Test if a convex set is empty.

SYNTAX

```
ts = S.isEmptySet
```

```
ts = isEmptySet(S)
```

DESCRIPTION

Return true if the convex set S is empty and false otherwise.

INPUT

   S   Any set derived from `ConvexSet` class, e.g. `YSet` or
       `Polyhedron`.
       Class: `ConvexSet`

OUTPUT

   ts   True if the convex set P is empty and false otherwise.
       Class: `logical`
       Allowed values: `true`
                     `false`

EXAMPLE(s)

Example 1

Describe circle in 2D using YALMIP

```
x = sdpvar(2,1);
```

```
F = [x'*x<=1];
```

```
S = YSet(x,F);
```

Check if the set is not empty

```
S.isEmptySet
```

```
ans =

0
```

Example 2

We have non-consistent constraints in 2D.

```
x = sdpvar(2,1);

F = [2*x(1)-3*x(2) <= 2; 2*x(1)-3*x(2) >= 3];

S = YSet(x,F);
```

The set must be empty

```
S.isEmptySet
```

```
ans =

1
```

SEE ALSO

isBounded

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
  mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
  mailto:herceg@control.ee.ethz.ch

FLIPLR

Revert the order of convex sets in the array.

SYNTAX

```
T = fliplr(S)
```

```
T = S.fliplr
```

DESCRIPTION

Overloaded method for reverting the order of sets in the array of convex sets.

INPUT

S An array of `ConvexSet` objects `Polyhedron`, `YSet`, ...
in the order of `S = [S1, S2, S3, ...]`.
Class: `ConvexSet`

OUTPUT

T The array of the convex sets in the reverted order `T = [..., S3, S2, S1]`.
Class: `ConvexSet`

EXAMPLE(s)

Example 1

We have three polyhedra stored in the array.

```
P1 = Polyhedron('lb',-1);
```

```
P2 = Polyhedron('lb',-2);
```

```
P3 = Polyhedron('lb',-3);
```

```
P = [P1,P2,P3]
```

```
Array of 3 polyhedra.
```

To reverse the order of the polyhedra, call the `fliplr` method

```
R = P.fliplr
```

```
Array of 3 polyhedra.
```

SEE ALSO

horzcat, vertcat

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

PLOT

Plot the convex set.

SYNTAX

```
h = plot(Set, 'Prop1', value1, 'Prop2', value2)

h = Set.plot('Prop1', value1, 'Prop2', value2)
```

DESCRIPTION

Plot the general convex set up to dimension three.
Figure properties, such as color, line width, etc, can be specified with "Property" - "Value" pairs.

INPUT

Set	Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `ConvexSet`
Prop1	Specification of figure properties. Class: `char` Allowed values:

Grid With how many gridpoints to grid the circle/sphere. Default is 40.

Color The color of the plot specified by real RGB vector or a string name of the color (e.g. 'gray');

Wire Highlight or not the edges of the set. Default is false.

LineStyle Specify the type of the line to plot edges of the set. Accepted values are '-',':','-.','–', and'none'.

LineWidth Specify the width of the line. Default is 1.

Alpha Transparency of the color. The value must be inside [0,1] interval. Default is 1.

Marker Type of markings to use. Allowed values are ".", "o", "x", "+", "*", "s", "d", "v", "^", "¡", "¿", "p", "h" or "none". Default is "none".

MarkerSize The size of the marker. Default is 6.

ColorMap Color map given either as a M-by-3 matrix, or as a string. Default is 'mpt'. Other available options are 'hsv', 'hot', 'gray', 'lightgray', 'bone', 'copper', 'pink', 'white', 'flag', 'lines', 'colorcube', 'vga', 'jet', 'prism', 'cool', 'autumn', 'spring', 'winter', 'summer'.

ColorOrder Either 'fixed' for fixed ordering of colors, or 'random' for a random order. Default is 'fixed'.

ShowIndex This option is valid only for bounded polyhedra in 2D. If true, display an index of the plotted element. The default choice is false.

value1	Corresponding value to `Prop1`.

OUTPUT

h   Handle related to graphics object.
     Class: `handle`

EXAMPLE(s)

Example 1

Plot two sets in 1D.
The first set is an interval [0, 3]

`x = sdpvar(1);`

`F1 = [ 0<= x <=3 ];`

`Y1 = YSet(x,F1);`

The second set is an intersection of two sets.

`F2 = [x <= 1; 0.3*x^2 <= 0.5];`

`Y2 = YSet(x,F2);`

Put the sets to an array.

`Y = [Y1; Y2];`

Plot the sets with the dash-dotted line and the size 3.

`Y.plot('LineStyle','-.','LineWidth',3);`
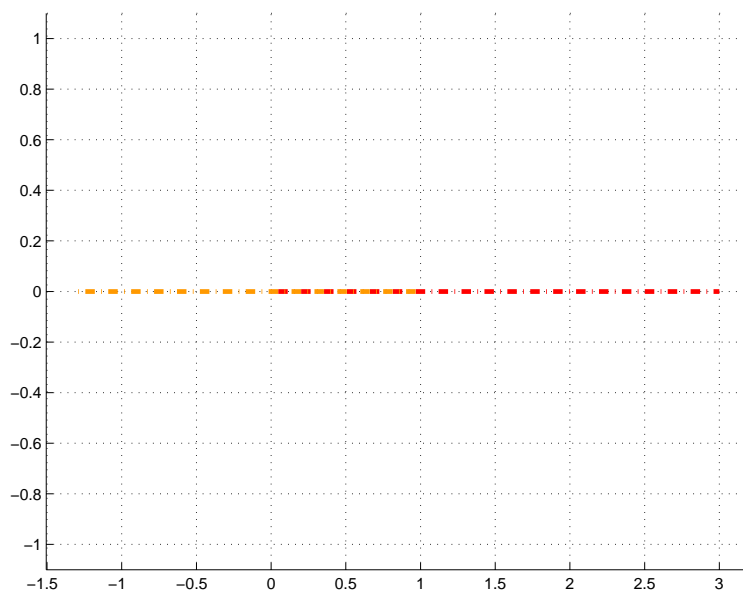
Example 2

We have a half circle describe the the following inequalities

```
x = sdpvar(2,1);
```

```
F = [x(1) <=1; x(2)>=1; 0.3*x'*x <= 0.5];
```

Construct the set

```
S = YSet(x,F);
```

Plot the set with with the olive color, half-transparent with circular markings.

```
S.plot('Color','olive','Alpha',0.5,'Marker','o');
```

```
Plotting...
27 of 40
```
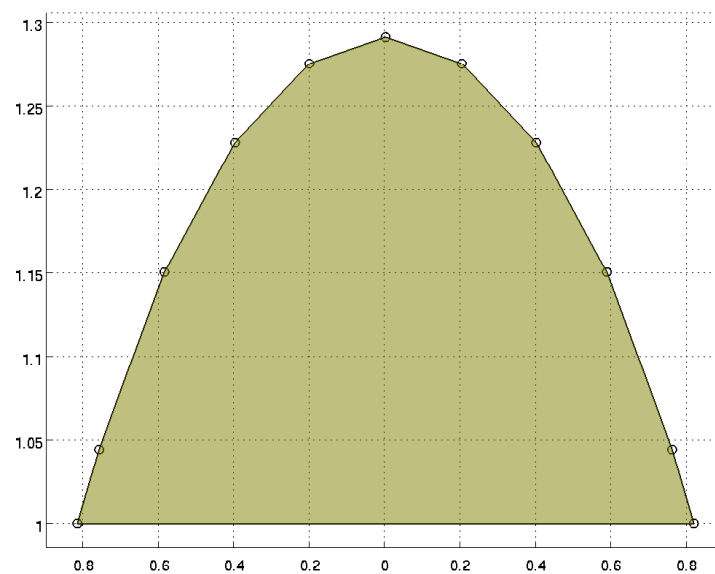

SEE ALSO

fplot

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

SEPARATE

Computes separating hyperplane between the set and given point.

SYNTAX

```
h = separate(S,x)

h = S.separate(x)
```

DESCRIPTION

Compute a separating hyperplane between the convex `Set` and the given point `x`. If `Set` is an array, the hyperplanes are returned as a cell array.

INPUT

S Any set derived from `ConvexSet` class, e.g. `YSet` or `Polyhedron`.
   Class: `ConvexSet`

x The point given as vector in the same dimension as `ConvexSet`.
   Class: `double`

OUTPUT

h Separating hyperplane defined as $\left\{ x \mid h\begin{pmatrix} x \\ -1 \end{pmatrix} = 0 \right\}$.
   Class: `double`

EXAMPLE(s)

Example 1

Describe a convex using YALMIP

```
x = sdpvar(2,1);

F = set(0.3*x'*x -0.7*x(1)<=1) + set(-x(1)+2.3*x(2)<=0.5);

S = YSet(x,F);
```

Compute the separating hyperplane from the point `v=[-2,2]`

```
v = [-3;2];

h = S.separate(v)
```

```
h =

2.01047891957449        -2.21283525231897        -5.98777459338193
```

Construct a new polyhedron out of the hyperplane and plot the sets

```
P = Polyhedron('He',h);

S.plot; hold on; axis([-4 4 -3 3]); text(v(1),v(2),'x'); P.plot;
```

```
Plotting...
28 of 40
```
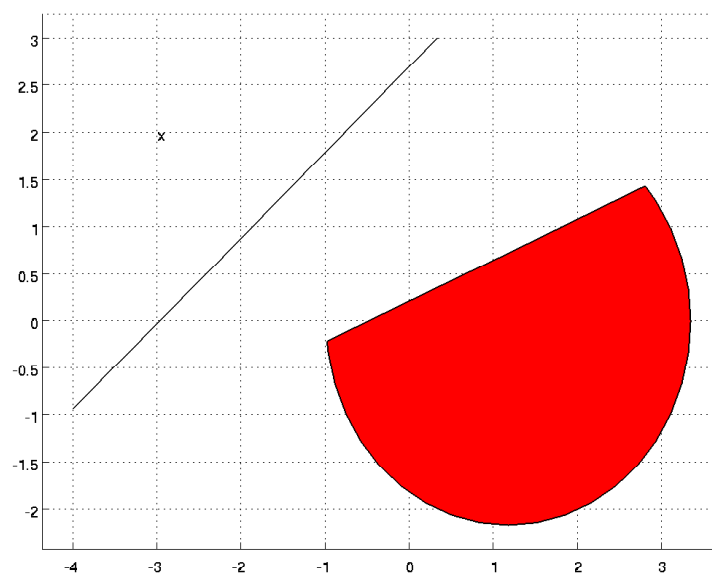

SEE ALSO

distance, support

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

AFFINEHULL

Computes affine hull of a convex set.

SYNTAX

```
He = affineHull(Set)

He = Set.affineHull
```

DESCRIPTION

Compute an implicitly-defined affine hull of the convex `Set`. The output is a real matrix `He` that defines the affine set

$$\left\{ x \ \middle| \ \mathtt{He} \begin{pmatrix} x \\ 1 \end{pmatrix} = 0 \right\}$$

If `He` is empty, then the affine hull is empty. The affine hull function for general convex sets will only function for bounded sets. If you want the affine hull of an unbounded set, then intersect your set with a large full-dimensional box.

INPUT

Set	Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `ConvexSet`

OUTPUT

H	The real matrix that defines the affine hull. Class: `double`

EXAMPLE(s)

Example 1

Construct a set by intersecting a circle, linear equality and inequality constraints using YALMIP.
Define variable first

```
x = sdpvar(2,1);
```

Define a set `S` using `YSet` class

```
opts = sdpsettings('solver','sedumi','verbose',0);
```

```
S = YSet(x,[norm(x)<=1; x(1)-x(2)==0.2; [1 -0.5; 0.3,0.8]*x<=[0.5;0.6]],opts);
```

We can plot the set

```
S.plot
```
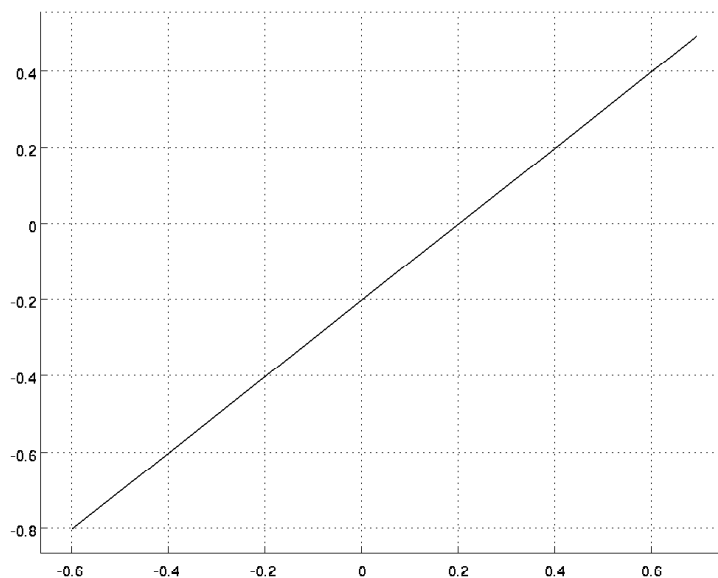
Compute the affine hull

```
S.affineHull
```

```
ans =

-0.70710678116385        0.707106781209245        -0.141421356213361
```

SEE ALSO

`innerApprox`, `outerApprox`, `isEmptySet`

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
   mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

FEVAL

Evaluates a function defined over a convex set or an array thereof.

SYNTAX

```
[fval, feasible] = Set.feval(x)

[fval, feasible] = Set.feval(x,function_name)

[fval, feasible] = feval(Set,x,function_name)
```

DESCRIPTION

Evaluates function for given value of the point x over the convex Set characterized by the the string function_name. The dimension of the vector x must be the same as the dimension of the Set. If the Set is an array of convex sets, the function values are returned in an array. For a point that lies outside of the Set, the output is NaN.

INPUT

Set	Convex set or an array thereof, i.e. any object derived from the ConvexSet class, e.g. Polyhedron, YSet, ... Class: ConvexSet
x	A point at which the function should be evaluated. The point must be given as column and must be in the same dimension as the set. Class: double
function_name	String name of the function to evaluate. It must refer to a single function. If omitted, S.feval(x) only works if the set has a single function. Class: char

OUTPUT

fval	An $(n \times m)$ matrix of function values at x, where m is the number of sets. If x is not contained in the j-th set, then the j-th column of fval is a vector of NaN. Class: double
feasible	An $(1 \times m)$ vector of logicals. feasible(j)=true if the j-th element of the array contains x. Class: double

EXAMPLE(s)

Example 1

Evaluate a function over a simple polytope.
Construct the polytope in H-representation

```
P = Polyhedron('lb',[-1;-2],'ub',[1;2]);
```

Assign a quadratic function to it

```
Q = QuadFunction([1 2;-2 3],[0 -5],0.6)
```

Quadratic Function: R^2 ->R^1


```
P.addFunction(Q,'q');
```

Obtain the value of the function for the point [-1;-1.5]

```
x = [-1; -1.5];
```

```
y = P.feval(x)
```

y =

15.85

We can plot the function over the set with the point [x,y]

```
P.fplot; hold on;  text(x(1),x(2),y,'\bf{x}','FontSize',14);
```


165
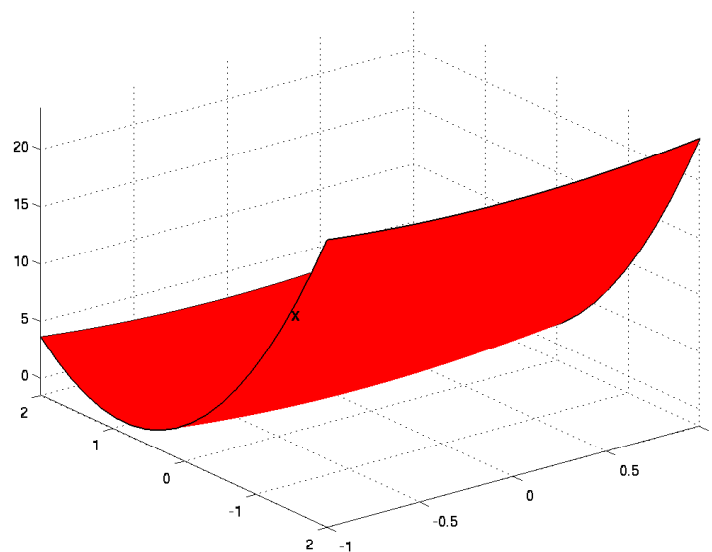
Example 2

We have three linear functions "a", "b", and "c" over unbounded polyhedron P.

```
P = Polyhedron([1 2;-0.6 8; 0 0.4; 5 -1],[1;2;0.7;0.01])
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities   4 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

```
F1 = AffFunction([1 2])
```

```
Affine Function: R^2 ->R^1
```

```
F2 = AffFunction([3 4; -1 0])
```

```
Affine Function: R^2 ->R^2
```

```
F3 = AffFunction([5 6; 7 8; 9 -1])
```

```
Affine Function: R^2 ->R^3
```

Add functions to a polyhedron with corresponding names in a cell array.

```
P.addFunction(F1,'a')
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities   4 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : 1 attached "a"
```

```
P.addFunction(F2,'b')
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities   4 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : 2 attached "a","b"
```

```
P.addFunction(F3,'c')
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities  4 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : 3 attached "a","b","c"
```

Evaluate function "a" for the point [-1;-1]

```
P.feval([-1;-1],'a')
```

```
ans =

-3
```

Evaluate functions "b" and "c" for the point [-1;-1]

```
y1 = P.feval([-1;-1],'b')
```

```
y1 =

-7
1
```

```
y2 = P.feval([-1;-1],'c')
```

```
y2 =

-11
-15
-8
```

SEE ALSO

fplot, Function

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
      mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
      mailto:michal.kvasnica@stuba.sk

HORZCAT

Horizontal concatenation for convex set objecs.

SYNTAX

```
S = [S1, S2]

S = horzcat(S1,S2)
```

DESCRIPTION

Overloaded method for horizontal concatenation of convex sets. It is not possible to concatenate objects of different type to the same array (e.g. `Polyhedron` and `YSet`). Similarly, it is not possible to concatenate into matrices, only vectors are allowed.

INPUT

S1   Any object derived from the `ConvexSet` class, e.g.
     `Polyhedron`, `YSet`, ...
     Class: `ConvexSet`

S2   Any object derived from the `ConvexSet` class that is of
     the same type as `S1`.
     Class: `ConvexSet`

OUTPUT

S   The array of the convex sets.
     Class: `ConvexSet`

EXAMPLE(s)

Example 1

We have two sets described in Yalmip.

```
x = sdpvar(1);

S1 = YSet(x,[x<=1]);

S2 = YSet(x,[x>=2]);
```

Horizontal concanation gives an array with 2 elements

```
S = [S1,S2]
```

```
Array of 2 YSets.
```

Example 2

It is not possible to create array with the mixed sets.
Define Yalmip set first

```
x = sdpvar(2,1);
```

```
S = YSet(x,[0.5*x(1)-4*x(2)<=1]);
```

Define Polyhedron object

```
P = Polyhedron([1 0; -1 -1]);
```

Concatenation of `YSet` and `Polyhedron` is not allowed

```
[S,P]
```

```
Only the same sets can be concatenated.
```

SEE ALSO

vertcat

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

SUPPORT

Compute the support of the set in the specified direction.

SYNTAX

```
s = support(S,x)
```

```
s = S.support(x)
```

DESCRIPTION

Compute the support of the set in the direction given by the point `x`. The underlying optimization problem to be solved is as follows

$$\max_y x'y$$

$$\text{s.t.} \quad y \in \texttt{Set}$$

where x is the point with the desired direction and y is the point lying inside the convex `Set`. The support is returned as the optimal value of the objective function $x'y$. The dimension of `x` must be the same as the `Set`. If an error occurs during by solving the above optimization problem, the support is empty.

INPUT

S   Any set derived from `ConvexSet` class, e.g. `YSet` or `Polyhedron`.
Class: `ConvexSet`

x   The point given as real vector in the same dimension as the `ConvexSet`.
Class: `double`

OUTPUT

s   The support is returned as the optimal value of the cost function.
Class: `double`

EXAMPLE(s)

Example 1

Describe a convex using YALMIP

```
x = sdpvar(2,1);
```

```
F = set(0.3*x'*x -0.7*x(1)<=1) + set(-x(1)+2.3*x(2)<=0.5);
```

```
S = YSet(x,F);
```

Compute the support in the direction of the point `v=[1,1]`

```
v = [1;1];
```

```
s = S.support(v)
```

```
s =

4.22461388383033
```

Check if the support was computed properly by computing the point y

```
r = S.extreme(v)
```

```
r =

exitflag: 1
how: 'Successfully solved (SeDuMi-1.3)'
x: [2x1 double]
supp: 4.22461388383033
```

Check the objective value

```
r.x'*v
```

```
ans =

4.22461388383033
```

Plot the set with the points `x`, `y`

```
S.plot; hold on; text(v(1),v(2),'\bf x'); text(r.x(1),r.x(2),'\bf y');
```

```
Plotting...
26 of 40
```

171
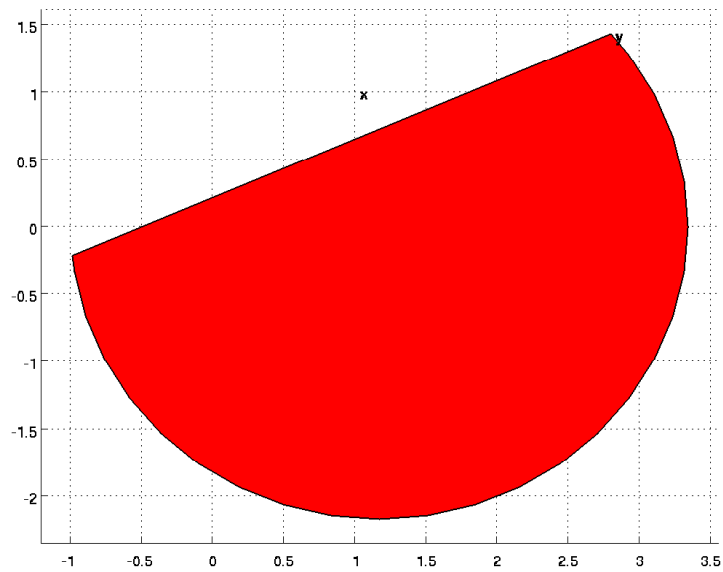
SEE ALSO

separate, distance

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

COPY

Create a copy of an object derived from the `ConvexSet` class.

SYNTAX

```
Snew = S.copy
```

```
Snew = copy(S)
```

DESCRIPTION

Create a copy of a set `S` and return a new object `Snew` with same properties as the original set. Changing one object does not affect the second one because the objects are not connected by reference.

Note that this method is different versus the equality assignment `Snew = S` which points to the same object. See the example for explanation.

INPUT

S   Any object derived from the `ConvexSet` class, e.g.
     `Polyhedron`, `YSet`, ...
     Class: `ConvexSet`

OUTPUT

Snew   New object which is an exact copy of the set `S`.
         Class: `ConvexSet`

EXAMPLE(s)

Example 1

Create a random polyhedron in V-representation.

```
P=ExamplePoly.randVrep
```

```
Polyhedron in R^2 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices  10 | Rays   0
Functions : none
```

Create a new object `Pnew` that is a copy of the set `P`.

```
Pnew = P.copy
```

```
Polyhedron in R^2 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices  10 | Rays   0
Functions : none
```

If we do any change on the new polyhedron `Pnew`, this will not affect the original polyhedron. For instance, we can convert it to H-representation:

```
Pnew.H;
```

```
Pnew.hasHRep
```

```
ans =

1
```

The original polyhedron `P` remains in V-representation

```
P.hasHRep
```

```
ans =

0
```

The different story is when a new object is a reference of the original set. This is achieved by equality assignment:

```
Q = P;
```

When the polyhedron `Q` is changed, this also affects the original polyhedron

```
Q.H;
```

```
Q.hasHRep
```

```
ans =

1
```

```
P.hasHRep
```

```
ans =

1
```

SEE ALSO

YSet, Polyhedron

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

LISTFUNCTIONS

Shows functions attached to a convex set.

SYNTAX

```
l = listFunctions(Set)
```

```
l = Set.listFunctions
```

DESCRIPTION

Shows all `Function` objects that are attached to a convex `Set`.

INPUT

Set   Any object derived from the `ConvexSet` class, e.g.
      `Polyhedron`, `YSet`, ...
      Class: `ConvexSet`

OUTPUT

l   List of function names stored with the set.
    Class: `cell`

EXAMPLE(s)

Example 1

The polyhedron contains two functions "a" and "b".

```
P = ExamplePoly.randHrep;
```

```
P.addFunction(AffFunction([1,-2]),'a');
```

```
P.addFunction(QuadFunction([0 0.4;-0.1 0.2]),'b');
```

List all functions in the set

```
l = P.listFunctions
```

```
l =

'a'   'b'
```

SEE ALSO

addFunction, getFunction, removeFunction

REMOVEALLFUNCTIONS

Remove all functions that are associated to this set.

SYNTAX

```
Set.removeAllFunctions
```

```
removeAllFunctions(Set)
```

DESCRIPTION

Removes all `Function` objects associated to this set.

INPUT

Set   Any object derived from the `ConvexSet` class, e.g.
      `Polyhedron`, `YSet`, ...
      Class: `ConvexSet`

OUTPUT

Set   Modified object of `ConvexSet` class without the function
      handles.
      Class: `ConvexSet`

EXAMPLE(s)

Example 1

Create a polyhedron and add two functions to the set.

```
P=ExamplePoly.randHrep;
```

```
P.addFunction(AffFunction([1,0],-1),'a')
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities  10 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : 1 attached "a"
```

```
P.addFunction(AffFunction([-3,5],2),'b')
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities  10 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : 2 attached "a","b"
```

178

The set contains now two functions. We can remove this functions at once with the following command

```
P.removeAllFunctions
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities  10 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : none
```

SEE ALSO

addFunction, removeFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

CONVEXSET

Represets a convex set in MPT

SYNTAX

DESCRIPTION

Represents a general convex set in $\mathbb{R}^n$. The dimension of the set, i.e. n is stored under `Dim` property. The `ConvexSet` cannot be instantiated, you can only create objects derived from this class, see related functions.

You can associate functions to `ConvexSet` class via `addFunction` methods. Function handles are stored inside `Func` property.

The `ConvexSet` class handles various methods that operate over convex sets and functions over convex sets. For a list of available methods type "methods('ConvexSet')".

SEE ALSO

YSet, Polyhedron, Union, PolyUnion

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

GRID

Grid the convex set.

SYNTAX

```
x = grid(Set, N)

x = Set.grid(N)
```

DESCRIPTION

Gridding of the convex `Set` with respect to `N` linearly scaled gridding points. The output `x` consist of points sorted vertically that belonging to the `Set`.
The principle of the algorithm is as follows:

1. Compute outer bounding hypercube.

2. Grid the hypercube.

3. Test each point for inclusion in the set, discarding those outside.

Before running the algorithm, consider the number `N` of gridding points. If this number is very large then it takes algorithm longer to grid the space because an exhaustive search is done at the last step of the algorithm.

INPUT

Set   Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ...
Class: `ConvexSet`

N   The number of gridding point. It specifies with how many elements to scale the interval equidistantly.

OUTPUT

x   An array of points sorted vertically. The number of columns specifies the dimension.
Class: `double`

EXAMPLE(s)

Example 1

The set is given as intersection of two sets.

```
x = sdpvar(2,1);

F = [x(1)-x(2)<=1; 0.3*x'*x <= 0.5];

Y = YSet(x,F);
```

Grid the set with 40 gridding points (can take some time).

```
x = Y.grid(30);
```

Plot each point

```
plot(x(:,1),x(:,2),'x','MarkerSize',7)
```
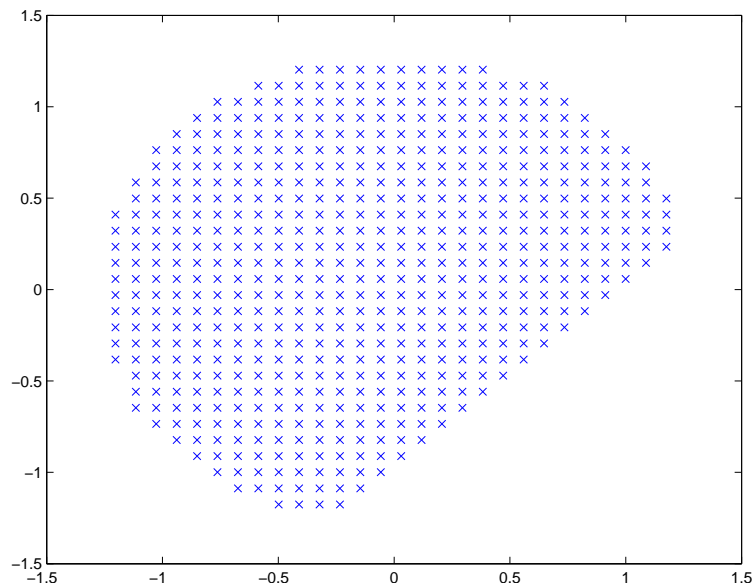

Example 2

We have two polyhedra that intersect

```
P(1) = Polyhedron('A',[1 -0.3; 0.5 0.7],'b',[2.3;3.1],'lb',[1;0.4]);
```

```
P(2) = Polyhedron('A',[1.6 -0.6; 0.2 0.4],'b',[2.1;2.8],'lb',[1.5;0.9]);
```

Plot the sets

```
P.plot
```
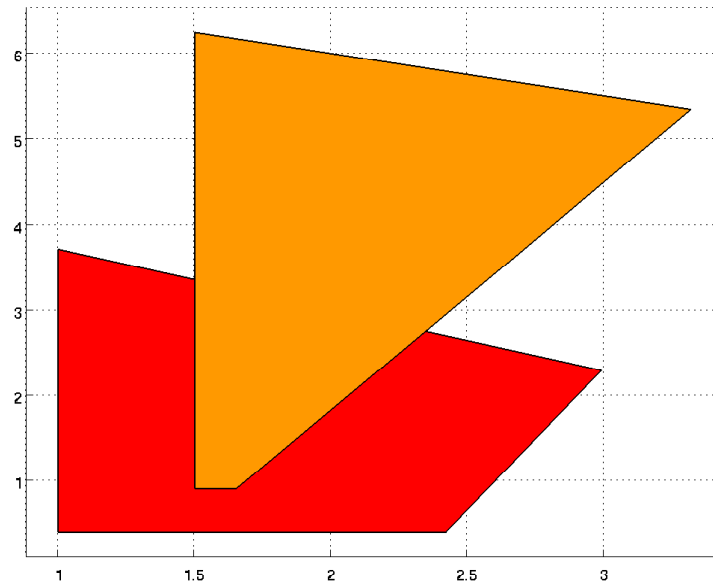
Find the intersection and grid it

```
Q = intersect(P(1),P(2));
```

```
x = Q.grid(40);
```

Plot the grid points of the intersection

```
plot(x(:,1),x(:,2),'o','Color','r');axis([1 3.5 0 7]);
```
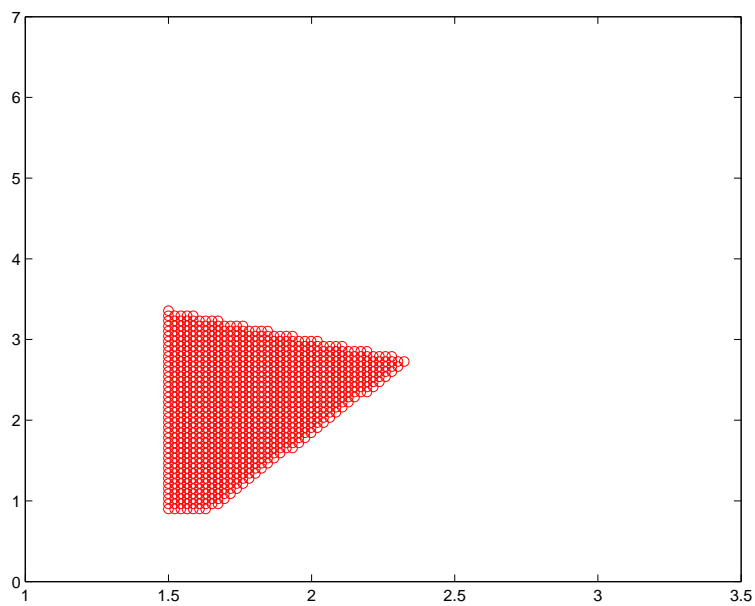
SEE ALSO

plot, fplot, contains

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

ADDFUNCTION

Attach function to a convex set.

SYNTAX

```
Set = addFunction(Set,F,FuncName)
```

```
Set.addFunction(Set,F,FuncName)
```

DESCRIPTION

Assign a `Function` object `F` to a convex set `Set` under the name `FuncName`. The set represent the domain of the function. The string `FuncName` is obligatory and is used to identify the function from the list.

Multiple functions can be assigned to a set, their names must be different.

INPUT

Set	Any object derived from the `ConvexSet` class, e.g. `Polyhedron`, `YSet`, ... Class: `ConvexSet`
F	Function or an array of functions to be assigned, given as objects derived form `Function` class. Class: `Function`
FuncName	A string or an cell array of strings that represent the name of the assigned function. Every function must have an unique name. Class: `struct`

OUTPUT

Set	Modified object of `ConvexSet` class that has a function stored under the `Func` property. Class: `ConvexSet`

EXAMPLE(s)

Example 1

Construct polyhedron $x <= 1$

```
P = Polyhedron(1,1)
```

```
Polyhedron in R^1 with representations:
H-rep (redundant)  : Inequalities  1 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
```

Functions : none

Define a function over $f(x) = 3x^2 + 1$ this set as a `QuadFunction` object.

`F = QuadFunction(3,0,1)`

Quadratic Function: R^1 ->R^1

Assign a function `F` to this set `P` under name "f".

`P.addFunction(F,'f')`

```
Polyhedron in R^1 with representations:
H-rep (redundant)   : Inequalities   1 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : 1 attached "f"
```

Example 2

Construct 2 polyhedra, $P_1 = \{x \mid x <= 1\}$, $P_2 = \{x \mid x >= 1\}$

`P1 = Polyhedron(1,1); P2 = Polyhedron(-1,-1);`

Put the polyhedra to one array.

`P = [P1,P2];`

Define function $f(x) = x/(x-1)$ that we want to associate to both functions.

`F = Function(@(x)x/(x-1))`

Function: @(x)x/(x-1)

Add the function to each of the polyhedron in the array `P` and give it the name "f".

`P.addFunction(F,'f')`

Array of 2 polyhedra.

Example 3

Add two functions $f_1(x) = \sqrt{(x_1^2 + x_2^2 + 1)}$ and $f_2(x) = (x_1 - 2)/(x_2 + 3)$ to a circle $||x||_2 <= 1$
Define the set using YALMIP

```
x = sdpvar(2,1); Set = YSet(x,[norm(x)<=1]);
```

Define the functions

```
f1 = Function(@(x) sqrt(x(1)^2+x(2)^2+1)); f2 = Function(@(x) (x(1)-2)/(x(2)+3));
```

Add these functions to the circle `Set`

```
Set.addFunction(f1,'polynomial')
```

```
YALMIP set in dimension 2.
Functions : 1 attached "polynomial"
```

```
Set.addFunction(f2,'rational')
```

```
YALMIP set in dimension 2.
Functions : 2 attached "polynomial","rational"
```

Plot the functions above the set

```
Set.fplot('polynomial')
```
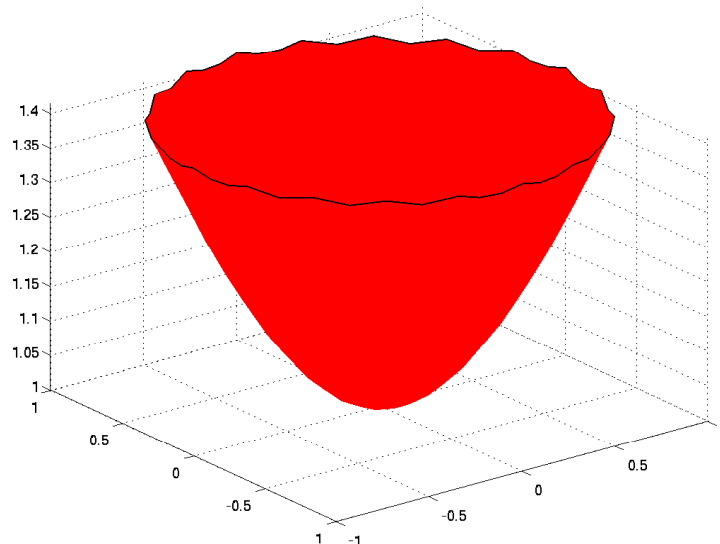


```
Set.fplot('rational')
```
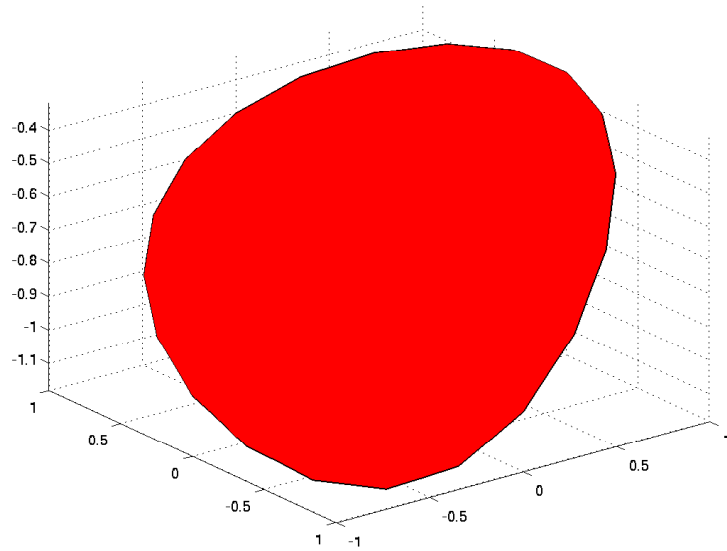
SEE ALSO

ConvexSet, Function, QuadFunction, AffFunction, removeFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

EXTREME

Compute an extreme point of this set in the given direction.

SYNTAX

```
s = S.extreme(x)

s = extreme(S, x)
```

DESCRIPTION

Compute an extreme point of this set in the direction given by the point x.

INPUT

S   A convex set described as YSet object.
     Class: YSet

x   A point given as vector. Note that for YSet with sym-
     metric matrix variable, the point x must be given as
     vector with symmetric terms.
     Class: double

OUTPUT

s   The output structure with the information about the
            extreme point and the exit status from the optimization.
            Class: struct

s.exitflag Exit status from the optimization, i.e. an integer value
            that informs if the result was feasible (1), or otherwise
            (different from 1).
            Class: double

s.how   A string that informs if the result was feasible ('ok'), or
            if any problem appeared through optimization.
            Class: char

s.x   Computed extreme point that lies on the boundary of
            the set S.
            Class: double

s.supp   The support of this set in the direction x which repre-
            sents the optimal value of the objective function in the
            optimization problem $\max_{y} x^T y$,   s.t. $y \in$ Set.

            Class: double

EXAMPLE(s)

Example 1

Create a set in 2D as intersection of the following inequalities.

```
x = sdpvar(2,1);

F = [ [-3 0.3;0.1 -1;-0.1 2]*x<=[0.8;2.1;1.5] ];

F = [F; 0.3*x'*x-4*x(1)+2*x(2)<=0.1];

S = YSet(x,F);
```

Compute the extreme point in the direction of the point v=[0;2].

```
v = [6;0];

s = S.extreme(v);
```

The computed extreme point is lying the edge of the set.
We can plot the set and the point `s.x`.

```
S.plot; hold on; text(s.x(1),s.x(2),'\bf x');
```

Plotting...
21 of 40
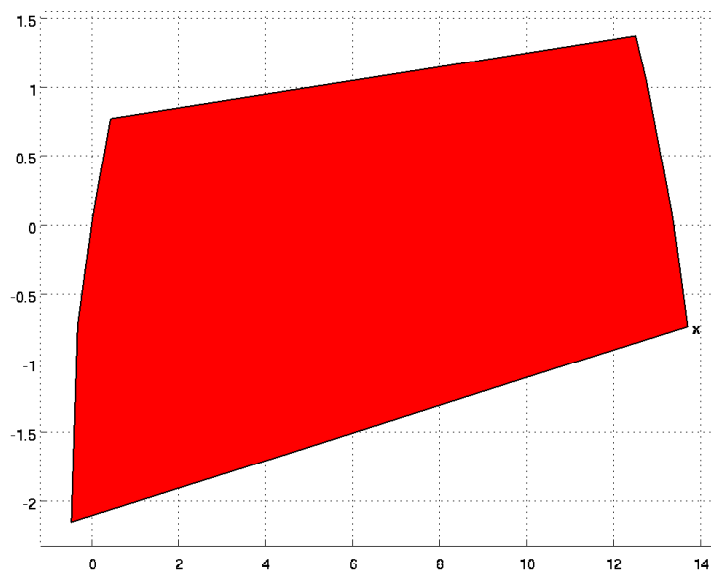
SEE ALSO

contains, project, shoot

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

DISP

Overload display for `YSet` class.

SYNTAX

```
display(S)
```

```
S.display
```

DESCRIPTION

Default display for `YSet` class.

INPUT

S   `YSet` object.
    Class: `YSet`

SEE ALSO

YSet

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

PROJECT

Compute the projection of the point onto this set.

SYNTAX

```
s = S.project(x)

s = project(S, x)
```

DESCRIPTION

Compute the projection of the point x onto this set. The projection problem can be also explained as finding the closest point y from the set S to the given point x.

INPUT

S   A convex set described as YSet object.
    Class: YSet

x   A point given as vector. Note that for YSet with symmetric matrix variable, the point x must be given as vector with symmetric terms.
    Class: double

OUTPUT

s   The output structure with the information about the projected point and the exit status from the optimization.
           Class: struct

s.exitflag Exit status from the optimization, i.e. an integer value that informs if the result was feasible (1), or otherwise (different from 1).
           Class: double

s.how   A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization.
           Class: char

s.x   Projected point that lies inside the set S.
           Class: double

s.dist   The distance between the projected point and the given point x.
           Class: double

EXAMPLE(s)

Example 1

Create a set `S` in 2D as follows.

```
x = sdpvar(2,1);
```

```
F = [x'*x-2*x<=1];
```

```
S = YSet(x,F);
```

Compute the projection of the point v=[-0.5; 1.5].

```
v = [-0.5; 1.5];
```

```
s = S.project(v);
```

The computed extreme point is lying the edge of the set and it is obviously the point [0;1]
We can plot the set and the points `s.x`, `v`.

```
S.plot; hold on; text(s.x(1),s.x(2),'\bf x'); axis([-1 1.5 -0.5 2]); text(v(1),v(2),'\bf
x');
```

```
Plotting...
26 of 40
```
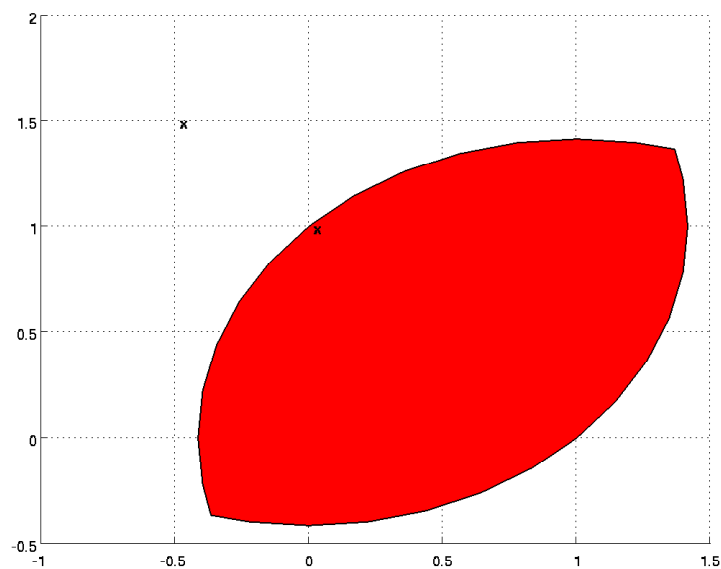

The computed distance must be

```
norm(s.x-v)
```

```
ans =
```

```
0.707106781189978
```

which equals the computed distance

```
s.dist
```

```
ans =
```

```
0.707106781189978
```

SEE ALSO

contains, extreme, shoot

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

SHOOT

Compute the maximal value of a multiplier in the desired direction.

SYNTAX

```
alpha = S.shoot(x)

alpha = shoot(S, x)
```

DESCRIPTION

Compute the maximal value of the multiplier α in the direction given by the point x by solving an optimization problem

$$\max \alpha$$

$$\text{s.t.} \quad \alpha x \in \texttt{Set}$$

This problem is usually referred as shooting towards the point $\texttt{x}$ because the point αx should lie on the edge of the set.

INPUT

S   A convex set described as `YSet` object.
Class: `YSet`

x   A point given as vector. Note that for `YSet` with symmetric matrix variable, the point `x` must be given as vector with symmetric terms.
Class: `double`

OUTPUT

alpha   The optimal value of the multiplier `alpha` such that `alpha*x` lies inside the `Set`. If the set is empty, not-a-number is returned. For unbounded direction, an inf-value is returned.
Class: `double`

EXAMPLE(s)

Example 1

Create a set `S` in 2D as follows.

```
x = sdpvar(2,1);

F = [x'*x-2*x<=1; x(2)+x(1)<=1];
```

```
S = YSet(x,F);
```

Compute the multiplier in the direction of the point v=[0.7; 0.7].

```
v = [0.7; 0.7];
```

```
alpha = S.shoot(v)
```

```
alpha =

0.714285714284946
```

The computed extreme point is lying the edge
We can plot the set and see the points v, `alpha*v`.

```
S.plot; hold on; text(v(1),v(2),'\bf x'); text(alpha*v(1),alpha*v(2),'\bf x');
```

```
Plotting...
25 of 40
```


SEE ALSO

contains, extreme, project

197

AUTHOR(s)

CONTAINS

Test if the point is contained inside convex set `YSet`.

SYNTAX

```
ts = S.contains(x)

ts = contains(S, x)
```

DESCRIPTION

Returns true if $x \in S$ and false otherwise.

INPUT

S   A convex set described as `YSet` object.
    Class: `YSet`

x   A point given as vector. Note that for `YSet` with sym-
    metric matrix variable, the point `x` must be given as
    vector with symmetric terms.
    Class: `double`

OUTPUT

ts True if the point `x` is contained inside `YSet`
    Class: `logical`
    Allowed values: `true`
                     `false`

EXAMPLE(s)

Example 1

Create two sets: polytope `P` and circle `C` in 2D.

```
x = sdpvar(2,1);

F1 = [ [1 -2;-0.4 1;0.6 -5]*x<=[1;1.2;1.7] ];

F2 = [ 0.3*x'*x-4*x(1)+2*x(2)<=0.1 ];

P = YSet(x,F1);

C = YSet(x,F2);
```

It is obvious that the point `x=[0;0]` must lie inside both sets.
Define the point `v` and the array `S`.

```
v = [0;0];
```

```
S = [P;C];
```

Check if the point is contained in both sets.

```
S.contains(v)
```

```
ans =

1
1
```

We can plot the sets and the point x.

```
S.plot; hold on; text(v(1),v(2),'\bf x');
```

```
Plotting...
23 of 40
```
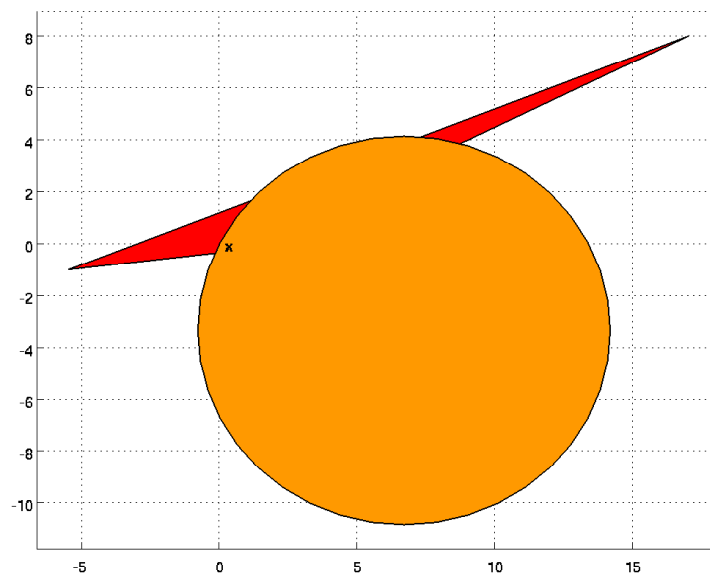

For instance, the point z=[5;-5] lies only in the set C.

```
z = [5;-5];
```

```
S.contains(z)
```

```
ans =

0
1
```

SEE ALSO

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

YSET

Representation of a convex set using YALMIP constraints.

SYNTAX

```
S = YSet(vars, constraints)

S = YSet(vars, constraints, options)
```

DESCRIPTION

The class `YSet` represents convex sets described by YALMIP constraints. Because YALMIP offers very broad specification of convex sets, the class `YSet` is useful when applying methods of the `ConvexSet` class that are not available in YALMIP. However, it is not intended with this class to replace basic functionalities for YALMIP objects. For reference how to use YALMIP objects, refer to YALMIP help. Only convex sets are accepted. Convexity is checked internally by YALMIP.

INPUT

vars	Symbolic variables used in the description of the constraints. The dimension of the variables must much the dimension used in the constraint set. Vector and matrix variables are accepted, multidimensional matrices are not allowed. Class: `sdpvar`
constraints	Constraint set given as `lmi` object. The constraints must build a convex set, otherwise the argument is not accepted. The convexity is checked internally by YALMIP. Class: `lmi`
options	YALMIP options defined by `sdpsettings`. You can specify the solver here, verbosity, the tolerances, etc. By default, these options are idependent of MPT settings. YALMIP chooses the solver based on its internal preferences and depending on the type of the constraint set. For more details, type `help sdpsettings`. Class: `struct`

OUTPUT

S	`YSet` object representing a convex set. Class: `YSet`

EXAMPLE(s)

Example 1

Create a set that looks like a *pizza cut*.
Define 2D vector x.

```
x = sdpvar(2,1);
```

Create constraints in YALMIP.

```
F = [-x(1)+x(2)<=0;  -x(1)-x(2)>=0; x'*x<=1];
```

Construct the set

```
S = YSet(x,F)
```

```
YALMIP set in dimension 2.
Functions : none
```

Plot the set

```
S.plot
```

```
Plotting...
27 of 40
```
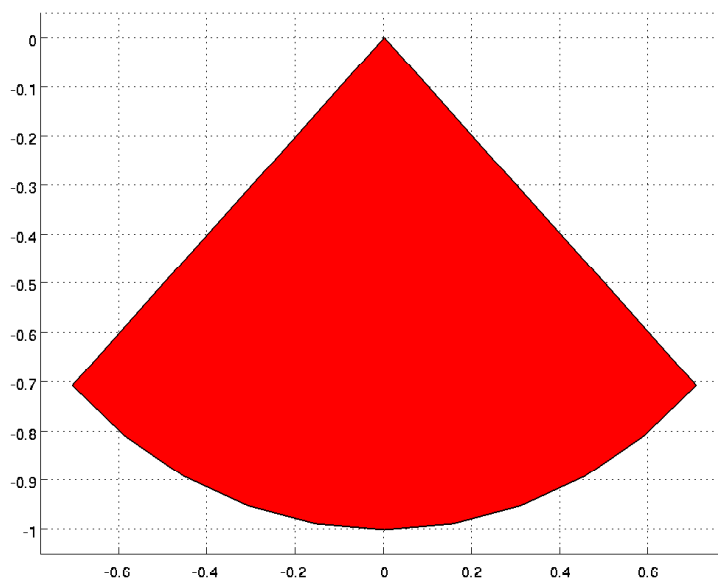
Example 2

Create a set described by linear matrix inequalities $P >= 0$, $A'P + PA <= I$.
Firstly, define the unknow symmetric matrix `P` and a matrix `A`.

```
P = sdpvar(2),A = randn(2);
```

```
Linear matrix variable 2x2 (symmetric,real,3 variables)
```

Secondly, set the inequalities in YALMIP.

```
constraints = [P>=0; A'*P + P*A <= eye(2)]
```

```
+++++++++++++++++++++++++++++++++++++++++++++++++++
|  ID|       Constraint|                       Type|
+++++++++++++++++++++++++++++++++++++++++++++++++++
|  #1|    Numeric value|    Matrix inequality 2x2|
|  #2|    Numeric value|    Matrix inequality 2x2|
+++++++++++++++++++++++++++++++++++++++++++++++++++
```

Construct the set `S` out of this constraint description.

```
S = YSet(P(:),constraints)
```

```
YALMIP set in dimension 4.
Functions : none
```

SEE ALSO

ConvexSet, Polyhedron

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

ISPOINTED

Test if a polyhedron is pointed or not

SYNTAX

```
ts = P.isPointed
```

```
ts = isPointed(P)
```

DESCRIPTION

A polyhedron P in H-representation $P = \{ x \,|\, Ax \le b, \; A_e x = b_e \}$ is pointed if and only if its lineality space `null([A; A_e])` is empty. If the polyhedron is in V-representation, automatic conversion to H-representation will be performed.

INPUT

  P   Polyhedron in any format
       Class: `Polyhedron`

OUTPUT

  ts   True if the polyhedron P is pointed and false otherwise.
       Class: `logical`
       Allowed values: `true`
                    `false`

EXAMPLE(s)

Example 1

The unpointed polyhedron does not contain any vertex. For instance the following low-dimensional polyhedron is un-pointed:

```
A = [-1.2 1.63; 0.7 0.5]; b = [1; 0.7]; Ae = [0.4 -1.4]; be = 0.8;
```

```
P = Polyhedron('A',A,'b',b,'Ae',Ae,'be',be);
```

The polyhedron `P` is pointed

```
P.isPointed
```

```
ans =

1
```

SEE ALSO

isEmptySet, isBounded, isFullDim

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

COMPUTEVREP

Compute V-representation of a polyhedron.

SYNTAX

```
P = P.computeVRep
```

DESCRIPTION

Computes (possibly redundant) V-representation of the polyhedron:

$$P = V'lam|lam >= 0, sum(lam) = 1 + R'gam|gam >= 0$$

This method implements vertex enumeration using CDD solver and `nlrs` solver. Please note that this is computationally demanding problem and the CDD solver may become irresponsive for large input data.

INPUT

P   Polyhedron in H-representation
    Class: `Polyhedron`

OUTPUT

P   Polyhedron in V-representation.
    Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create facet representation of a polyhedron:

```
P = Polyhedron(randn(15,2),ones(15,1))

Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities  15 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

Compute its V-representation:

```
P.computeVRep

Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities  15 | Equalities   0
V-rep (redundant)   : Vertices   6 | Rays   0
Functions : none
```

Plot the result

```
plot(P);
```
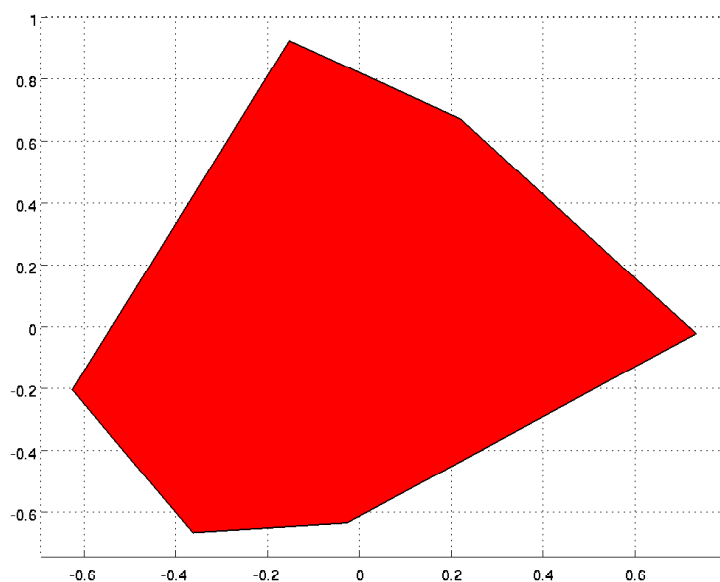

SEE ALSO

computeHRep, minVRep, minHRep

LITERATURE

Fukuda: PolyFaq

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

ISBOUNDED

Test if a polyhedron is bounded.

SYNTAX

```
tf = P.isBounded
```

```
tf = isBounded(P)
```

DESCRIPTION

Return true if the polyhedron P is bounded and false otherwise.

INPUT

P   Polyhedron in any format
     Class: `Polyhedron`

OUTPUT

tf   true if the polyhedron P is bounded and false otherwise.
      Class: `logical`
      Allowed values: `true`
                       `false`

EXAMPLE(s)

Example 1

Bounded polyhedron:

```
Polyhedron('V',randn(30,5)).isBounded
```

ans =

1

Unbounded polyhedra:

```
Polyhedron('V',randn(30,5),'R',randn(2,5)).isBounded
```

ans =

0

```
Polyhedron('H',[rand(30,5) ones(30,1)],'He',[randn(1,5) 0]).isBounded
```

```
ans =

0
```

SEE ALSO

isEmptySet, isFullDim, isBounded

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

COMPUTEHREP

Compute H-representation of a polyhedron.

SYNTAX

```
P = P.computeHRep
```

DESCRIPTION

Computes (possibly redundant) H-representation of the polyhedron:

$$P = \{x \mid Ax \le b\} \cap \{x \mid A_e x = b_e\}$$

This method implements facet enumeration using CDD solver and `nlrs` solver. Please note that this is computationally demanding problem and the CDD solver may become irresponsive for large input data.

INPUT

P   Polyhedron in V-representation
Class: `Polyhedron`

OUTPUT

P   Polyhedron in H-representation.
Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create vertex representation of a polyhedron:

```
P = Polyhedron('V',randn(15,3))

Polyhedron in R^3 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices  15 | Rays   0
Functions : none
```

Compute its H-representation:

```
P.computeHRep

Polyhedron in R^3 with representations:
H-rep (redundant)  : Inequalities  16 | Equalities   0
V-rep (irredundant) : Vertices  10 | Rays   0
Functions : none
```

Plot the result

```
plot(P);
```
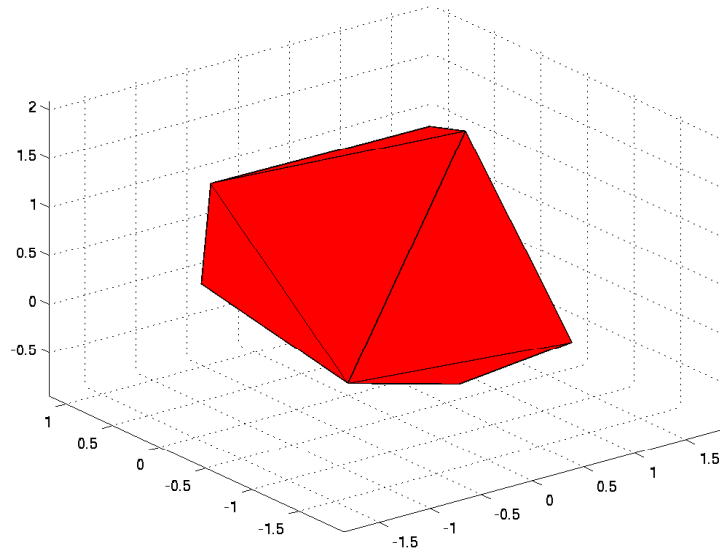

SEE ALSO

computeVRep, minVRep, minHRep

LITERATURE

Fukuda: PolyFaq

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

DISTANCE

Compute the distance between the given point/polyhedron and this polyhedron.

SYNTAX

```
dist = P.distance(x)

ret = P.distance(S)

dist = distance(P, x)

ret = distance(P, S)
```

DESCRIPTION

Compute the distance between the polyhedron P and the point x or the polyhedron S.

1. By providing real vector x, the distance between x and P is computed by solving the optimization problem

$$\min\{\|x - y\|_2^2 \mid y \in P\}$$

and the distance is returned as real number.

2. If polyhedron S is specified as the argument, the distance between S and P is computed by solving the following optimization problem

$$\min\{\|x - y\|_2^2 \mid y \in P, \ x \in S\}$$

where the results of the optimization are returned in a struct format.

INPUT

P   Polyhedron in any format
    Class: `Polyhedron`

x   Vector of size P.Dim
    Class: `double vector`

S   Polyhedron with the same dimension as P.
    Class: `Polyhedron`

OUTPUT

dist   Distance between the point x and the Polyhedron P
            Class: `double`

ret   Optimal solution or [] if P is empty.

Class: `struct`

`ret.exitflag`	Integer value informing about the termination status of the optimization. Class: `double`
`ret.dist`	Distance from S to the set P Class: `double`
`ret.x`	Point x in S closest to P. Class: `double`
`ret.y`	Point y in P closest to S. Class: `double`

EXAMPLE(s)

Example 1

Create a polytope:

```
P = ExamplePoly.poly3d_sin;
```

Choose a point and compute the distance:

```
y = [2;2];
```

```
d = P.distance(y);
```

Plot the result

```
P.plot; hold on; pplot(y','or');
axis square; th=linspace(0,2*pi,100)';
pplot(d.dist*[sin(th) cos(th)]+repmat(y',100,1),'k');
```
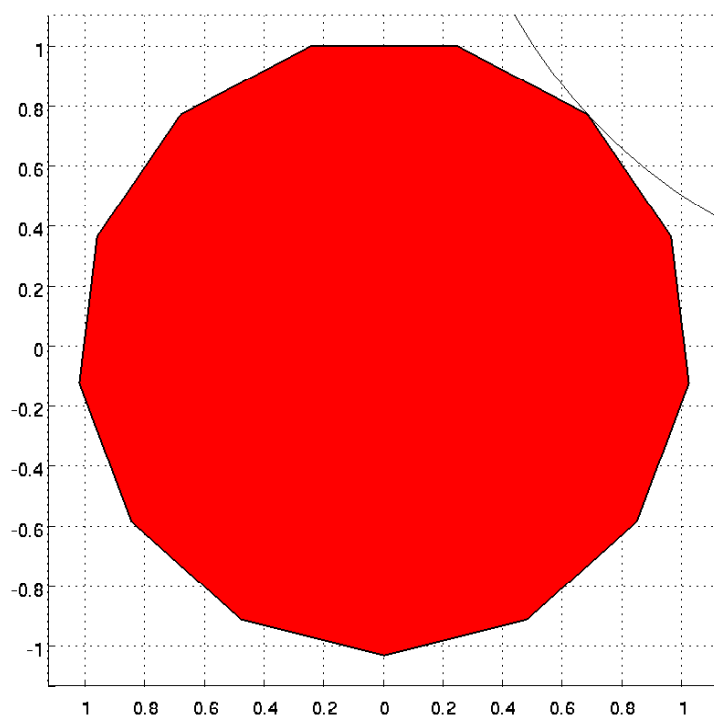
Example 2

Create two polyhedra:

```
P = ExamplePoly.poly3d_sin('d',3);

[U,s]=svd(randn(3));

S = ExamplePoly.poly3d_sin.affineMap(U(:,1:2)) + [2;2;2];
```

Compute the distance:

```
ret = distance(P,S);
```

Plot the result

```
plot([P S]); hold on; pplot([ret.x';ret.y'],'ok-'); axis square;
```

SEE ALSO
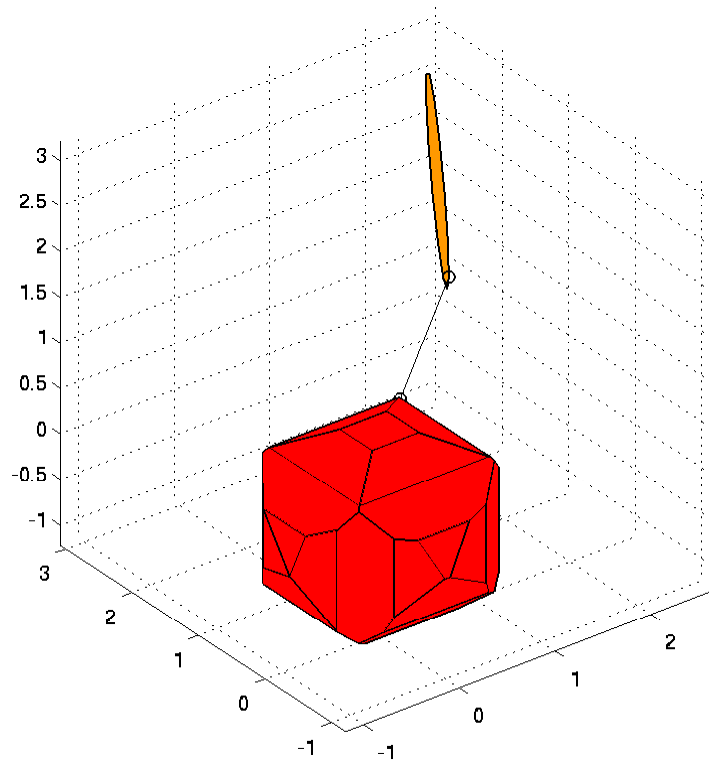
minVRep

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

GETFACET

Extract facet of the polyhedron specified by the inequality index.

SYNTAX

```
Q = P.getFacet()

Q = P.getFacet(index)
```

DESCRIPTION

Extract the facet of the polyhedron P specified by the inequality `index`. The returned polyhedron Q is formed by rewriting this ineqality as equality constraint. The polyhedron P is given as

$$P = \left\{ x \mid a_i^T x \le b_i,\ i = 1, \ldots, m,\ a_{e,j}^T x = b_{e,j}, j = 1, \ldots, m_e \right\}$$

Given the `index`, the polyhedron Q is built as

$$Q = \left\{ x \mid a_i^T x \le b_i, \forall i \ne \text{index},\ a_i^T x = b_i, i \in \text{index},\ a_{e,j}^T x = b_{e,j}, j = 1, \ldots, m_e, \right\}$$

The polyhedron P must be given in its minimal representation (irredundant) H-representation, otherwise an error is thrown.

INPUT

P   Polyhedron in H-representation
    Class: `Polyhedron`

index   Index of a facet from polyhedron P which is less or equal
        than the number of hyperplanes defining P. If omited,
        all facets will be returned as a polyhedron array.
        Class: `double`

OUTPUT

Q   Polyhedron Q that represents lower-dimensional facet
    of the Polyhedron P.
    Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create random polyhedron P and make it irredundant.

```
P = ExamplePoly.randHrep;

P.minHRep();
```

Extract the third facet of the polyhedron.

```
Q = P.getFacet(3);
```

Plot the polyhedra.

```
P.plot; hold on; Q.plot('LineWidth',3,'LineStyle','-.');
```


SEE ALSO

isAdjacent
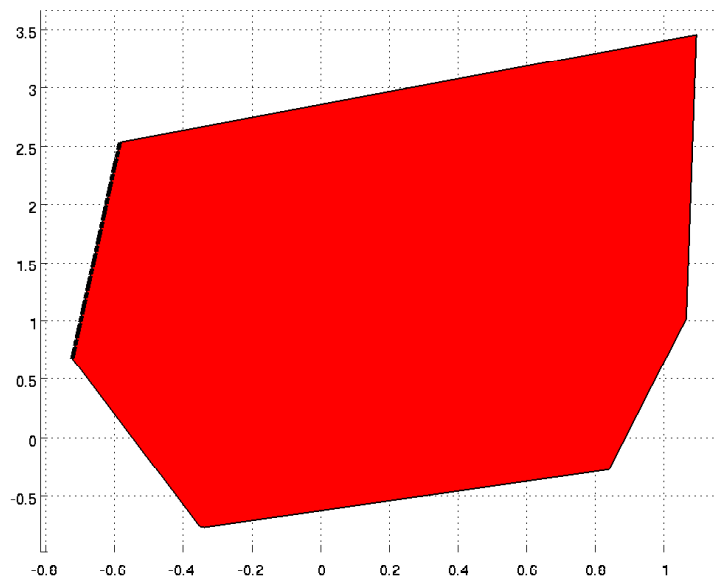
AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

HOMOGENIZE

Compute the homogenization of the given Polyhedron.

SYNTAX

```
H = P.homogenize(type)

H = homogenize(P, type)
```

DESCRIPTION

Compute the homogenization of the given Polyhedron. Parametrize the right hand side of the inequalities/equalities that describe the polyhedron to get homogenized system of equations $Ax \leq 0$ and $A_e x = 0$.
Given Polyhedron

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b, \ A_e x = b_e\}$$

the homogenization is

$$H = \{x \in \mathbb{R}^n, t \in \mathbb{R}^1 \mid Ax - bt \leq 0, \ A_e x - b_e t = 0\}$$

where the t is the lifting parameter. The dimension of the polyhedron H is by one greater than the dimension of P.
If `type = 'Hrep'` or `type = 'Vrep'` is specified, then the homogenization is returned in this form, otherwise the returned type is equal to the type of P.

INPUT

P   Polyhedron in any format
         Class: `Polyhedron`

type   Desired type of the returned polyhedron
         Class: `char`
         Allowed values: `Hrep` Hyperplane representation.
                          `Vrep` Vertex representation.

OUTPUT

H   Homogenization of `P` polyhedron
     Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create random polyhedron and homogenize:

```
P = ExamplePoly.randHrep;

H = P.homogenize;
```

Plot the result:

```
plot([P,H]);
```


AUTHOR(s)
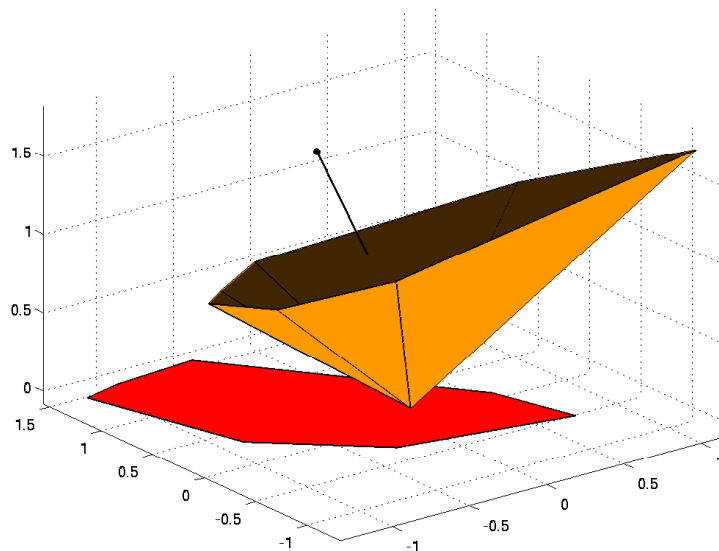
© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

EXTREME

Compute extremal point of a polyhedron in a given direction.

SYNTAX

```
sol = P.extreme(y)
```

DESCRIPTION

`P.extreme(y)` solves the following problem:

$$J(y) = \max_x \ \{y'x \,|\, x \in P\}$$

and returns the optimizer / extreme point x.

INPUT

P   Polyhedron in any format.
      Class: `Polyhedron`

y   Direction to compute the extreme point.
      Class: `double`

OUTPUT

sol   Support of y in P.
            Class: `struct`

sol.exitflag   Integer value informing about the termination status of
            the optimization.
            Class: `double`

sol.x   An optimizer of $\max_x \ \{y'x \,|\, x \in P\}$
            Class: `double`

sol.supp   Optimal value of $\max_x \ \{y'x \,|\, x \in P\}$
            Class: `double`

EXAMPLE(s)

Example 1

Create random polytope:

```
P = Polyhedron('H',[randn(20,2) ones(20,1)])

Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities  20 | Equalities   0
```

Compute extreme point in random direction

`y = randn(2,1); sol = P.extreme(y);`

Plot the point `sol.x` and the supporting hyperplane

```
plot(P,'alpha',0.2);  hold on;
plot(sol.x(1),sol.x(2),'ro','markersize',10);
Polyhedron('He',[y'sol.supp],'lb',-[1;1],'ub',[1;1]).plot;
```
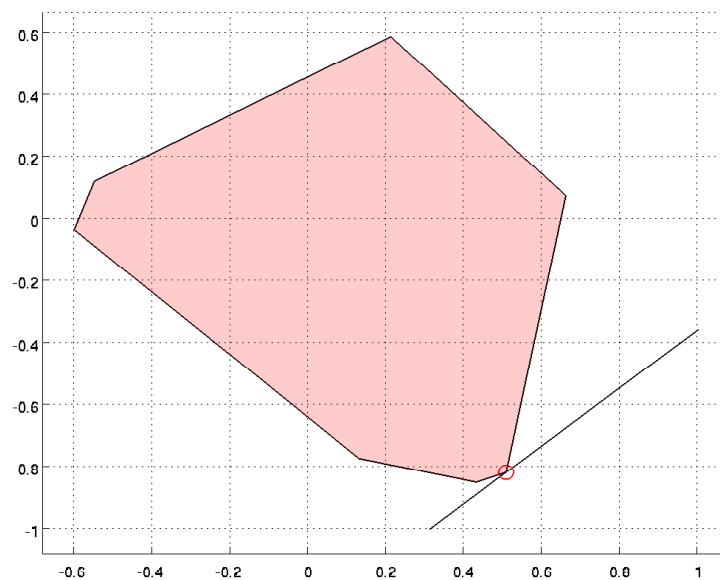

LITERATURE

Fukuda: PolyFaq

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

ISNEIGHBOR

Test if a polyhedron touches another polyhedron along a given facet.

SYNTAX

```
ts = P.isNeighbor(Q)

ts = isNeighbor(P,Q)

[ts, iP, iQ] = isNeighbor(P,Q,fP,fQ)
```

DESCRIPTION

Return true if the polyhedron P shares with the polyhedron Q a part of a common facet. Both polyhedrons must be in H-representation. If they are not, the irredundant H-representation will be computed.

The function tests if the intersection of two polyhedra P and S in dimension d is nonempty and is of dimension $d - 1$. If this holds, then the two polyhedra are touching themself along one facet. For closer explanation, see the example below.

INPUT

P   Polyhedron in H-representation
     Class: `Polyhedron`

Q   Polyhedron in H-representation
     Class: `Polyhedron`

fP   Index of a facet to test from polyhedron P.
     Class: `double`

fQ   Index of a facet to test from polyhedron Q.
     Class: `double`

OUTPUT

ts   Logical statement if the polyhedron P touches Q along
     the facet.
     Class: `logical`
     Allowed values: `true`
                     `false`

iP   Index of a facet from polyhedron P that touches the
     polyhedron Q.
     Class: `double`

iQ   Index of a facet from polyhedron Q that touches the
     polyhedron P.
     Class: `double`

EXAMPLE(s)

Example 1

Create three neighboring polyhedra `R1`, `R2`, and `R3`.

`R1 = Polyhedron([0 0; 2 0; 0 1; 2 1]);`

`R2 = Polyhedron([1 0; 2 0; 1 -1; 2 -1]);`

`R3 = Polyhedron([2 0; 3 0; 2 -1; 3 -1]);`

Plot the polyhedrons to see that they are touching themself.

`plot([R1 R2 R3],'showindex',true)`
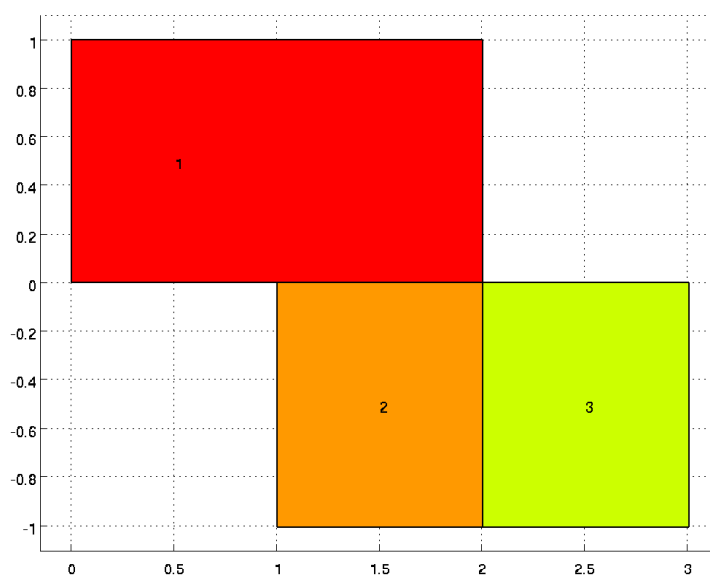
Check that polyhedron R1 touches polyhedron R2.

`R1.isNeighbor(R2)`

`ans =`

`1`

Check that the fourth facet of the polyhedron R2 touches the second facet of polyhedron R3.

`[ts,iR2,iR3] = R2.isNeighbor(R3)`

```
ts =

1


iR2 =

2


iR3 =

4
```

Actually, the facets of `R2` and `R3` are the same, so these polyhedra have face to face property that can be checked using `isAdjacent` method.

```
R2.isAdjacent(R3)


ans =

1
```

SEE ALSO

isAdjacent

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ISINSIDE

Test if a point is contained inside polyhedron in H-representation.

SYNTAX

```
[isin, inwhich, closest] = isInside(Pn, x0, Options)

[isin, inwhich, closest] = Pn.isInside(x0, Options)
```

DESCRIPTION

Check if the point `x0` is contained inside the polyhedron array `Pn`. The result it the logical statement if $x0 \in P$ and false otherwise.
`P` must be given in H-representation, otherwise an error is thrown.
Note that this operation depends on the settings of absolute tolerance that can be changed in `Options` settings.

INPUT

P	Polyhedron in H-representation. Class: `Polyhedron`
x0	A point in the same dimension as polyhedron and given as real column vector. Class: `double`
Options	A structure with the option settings for point location problem. Class: `struct`
Options.abs_tol	Absolute tolerance for checking the satisfaction of inequalities and equalities. Class: `double` Default: `mptopt.abs_tol`
Options.fastbreak	If `Pn` is the polyhedron array, then do a quick stop in the consecutive search when `x0` is contained in any of the polyhedrons. Class: `logical` Allowed values: `true` `false` Default: `false`

OUTPUT

isin	True if $x0 \in P$ and false otherwise. Class: `logical` Allowed values: `true` `false`

inwhich	If `Pn` is an array of polyhedra in the same dimension, than `isin` indicates which polyhedra from this array contain the point `x0`. Class: `double`
closest	If `Pn` is an array of polyhedra in the same dimension and none of polyhedra contains `x0`, then the field `closest` indicates which polyhedra has the closest distance for `x0` to lie in it's interior. Class: `double`

EXAMPLE(s)

Example 1

Create two polytopes in 2D.

```
P(1) = ExamplePoly.randHrep;
```

```
P(2) = ExamplePoly.randHrep+[1;1];
```

Check if the point x0=[1.1;0.9] is contained in any of them.

```
x0 = [1.1; 0.9];
```

```
[isin,inwhich] = P.isInside(x0)
```

```
isin =

1


inwhich =

2
```

Plot the point and polyhedra

```
plot(P); hold on; text(x0(1),x0(2),'\bf x')
```
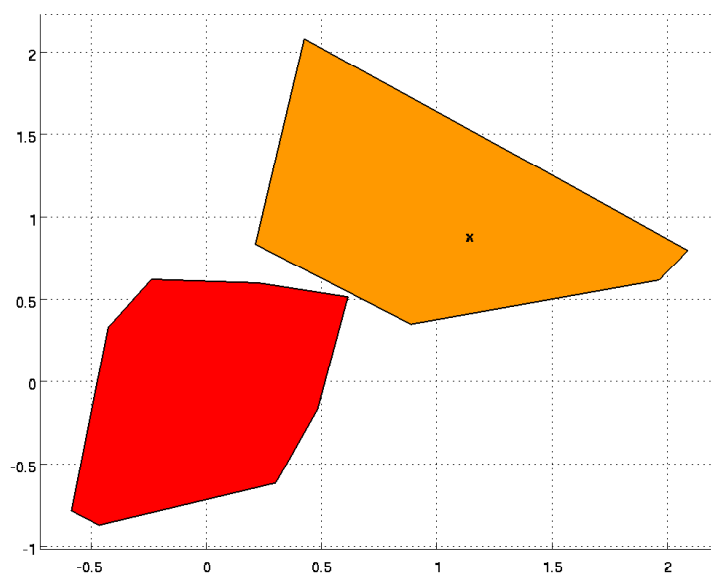
SEE ALSO

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

NEQ

Test if a polyhedron is not equal to another polyhedron.

SYNTAX

```
S ~= P
```

DESCRIPTION

Check if the polyhedron `S` is not equal to polyhedron `P`. The result it the logical statement if $P \neq S$ and false otherwise.

INPUT

P   Polyhedron in any format
    Class: `Polyhedron`

S   Polyhedron in any format with the same dimension as
    P.
    Class: `Polyhedron`

OUTPUT

tf   True if $P \neq S$ and false otherwise.
     Class: `logical`
     Allowed values: `true`
                       `false`

EXAMPLE(s)

Example 1

Compare two different polyhedra.
Polyhedron `P`.

```
P = ExamplePoly.randHrep;
```

Polyhedron `S`.

```
S = ExamplePoly.randVrep;
```

```
P =S
```

SEE ALSO

eq, contains, le, lt, gt, ge

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

INTERIORPOINT

Compute a point in the relative interior of the Polyhedron.

SYNTAX

```
sol = P.interiorPoint
```

```
sol = P.interiorPoint(facetIndex)
```

DESCRIPTION

Compute a point in the relative interior of the polyhedron.
If $\texttt{facetIndex} \in \{1, \ldots, \texttt{P.size(P.A, 1)}\}$ is specified, then a point in the relative interior of
$P \cap \{x \,|\, P.A(\texttt{facetIndex}, :)x \le P.b(\texttt{facetIndex})\}$ is returned.

INPUT

P	Polyhedron in any format Class: `Polyhedron`
facetIndex	Index of an inequality of P (row of `P.H`). Class: `integer`

OUTPUT

sol	Class: `struct`
sol.x	The interior point Class: `double vector`
sol.isStrict	The output is true if x is in the strict relative interior, false otherwise. Class: `logical`
sol.r	Radius of the largest ball centered at x that is still within P

$$y - x \in P, \forall \|y\| \le r$$

Note : r is empty if P is empty or only has a V-rep.
Class: `double`

EXAMPLE(s)

Example 1

Compute interior point in unbounded polyhedra

```
P = Polyhedron('V',randn(20,3),'R',-[1 0 0]);

sol = P.interiorPoint
```

```
sol =

x: [3x1 double]
isStrict: 1
r: []
```

Example 2

Compute a point in the relative interior of the fourth facet

```
P = Polyhedron('H',[sin([0:0.5:2*pi])'cos([0:0.5:2*pi])'ones(13,1)]);
```

Polyhedron must be in its minimal representation to compute facets. Perform redundancy elimination.

```
P.minHRep();
```

Compute the center of the fourth facet

```
sol = P.interiorPoint(4)
```

```
sol =

x: [2x1 double]
isStrict: 1
r: 0.122417438109627
```

```
plot(P); hold on;
pplot(sol.x,'ro','markerfacecolor','r','markersize',10);
```
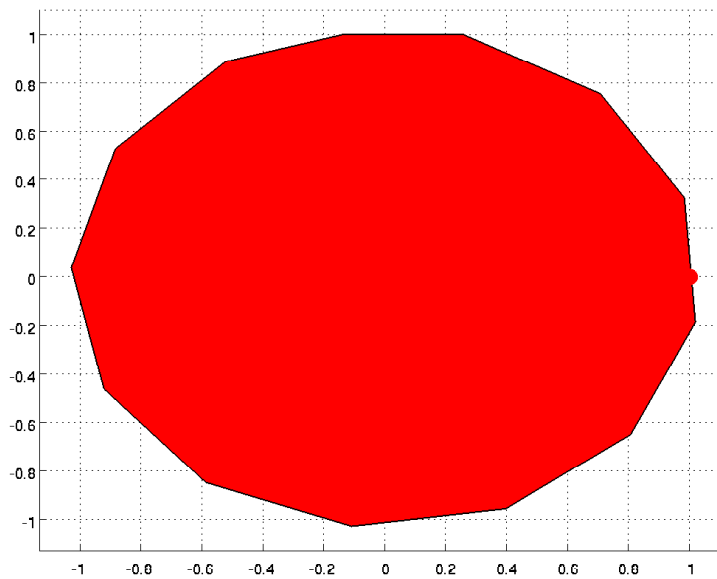
Example 3

Compute a point in the relative interior of a lower-dimensional polyhedron.

```
P = Polyhedron('H',[randn(20,3) ones(20,1)],'He',[0 0 1 0]);

sol = P.interiorPoint


sol =

x: [3x1 double]
isStrict: 0
r: 0.39490124253373



plot(P); hold on;
pplot(sol.x,'ro','markerfacecolor','r','markersize',10);
```
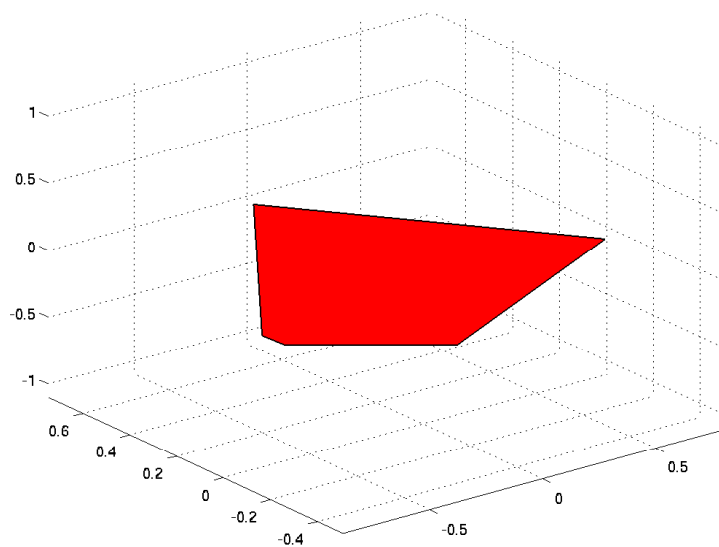
SEE ALSO

chebyCenter, facetInteriorPoints

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
     mailto:colin.jones@epfl.ch

MINUS

Subtract a Polyhedron or a vector from a Polyhedron.

SYNTAX

```
Q = P - S

Q = minus(P,S)

Q = P - x

Q = minus(P, x)
```

DESCRIPTION

Compute the Pontryagin difference of P and S, or P and x.

$$Q = P - S = \{x \in P \mid x + w \in P \ \forall w \in S\}$$

or

$$P - x = \{y - x \mid y \in P\}$$

INPUT

- P Polyhedron in any format
  Class: `Polyhedron`

- S Polyhedron in any format
  Class: `Polyhedron`

- x Column vector of length `P.Dim`
  Class: `Polyhedron`

OUTPUT

- Q Polyhedron `Q = P-S` or `Q = P-x`.
  Class: `Polyhedron`

EXAMPLE(s)

Example 1

Subtract one polytope from another

```
P = Polyhedron('H',[sin([0:0.5:2*pi])'cos([0:0.5:2*pi])'ones(13,1)]);

S = Polyhedron('lb',[-1;-1]*0.1,'ub',[1;1]*0.1);
```

```
Q = P-S;

P.plot('color','b'); hold on;
Q.plot('color','g');
S.plot('color','r');
```
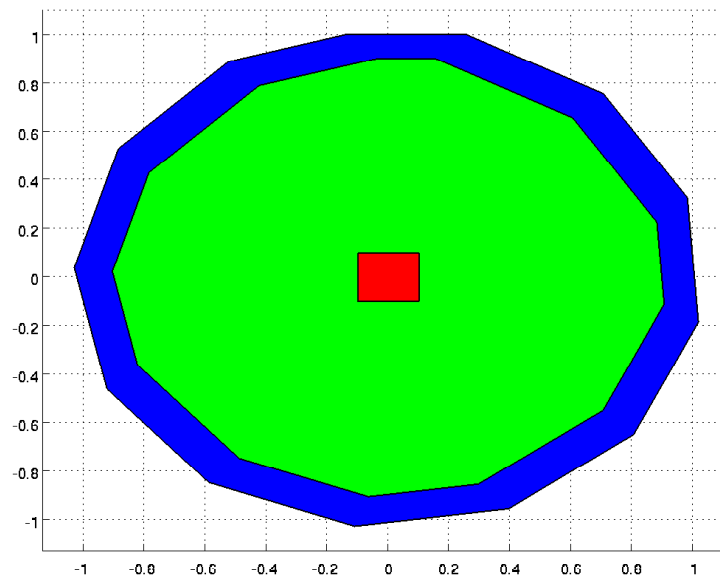

Example 2

Subtract a line segment from a polytope

```
P = Polyhedron('H',[sin([0:0.5:2*pi])'cos([0:0.5:2*pi])'ones(13,1)]);

S = Polyhedron('V',0.1*[-1 1;1 -1]);

Q = P-S;

P.plot('color','b'); hold on;
Q.plot('color','r');
S.plot('color','g','linewidth',2);
```
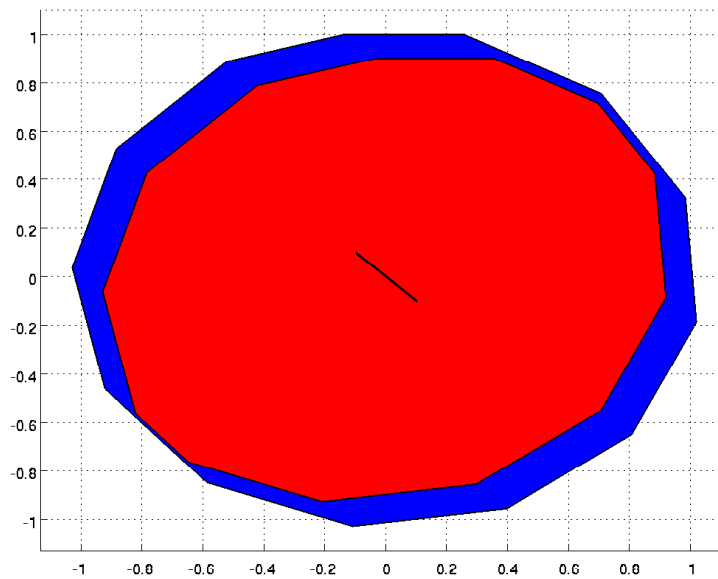
Example 3

Unbounded polyhedron minus lower-dimensional polytope

```
P = Polyhedron('lb',-[1;1;1]*0.5,'ub',[1;1;1]*0.5,'He',[randn(1,3) 0.1]);

S = Polyhedron('V',randn(10,3),'R',[1 1 1])+[1;1;1];

Q = S-P;

P.plot('color','g'); hold on;
S.plot('color','b');
Q.plot('color','r');
```
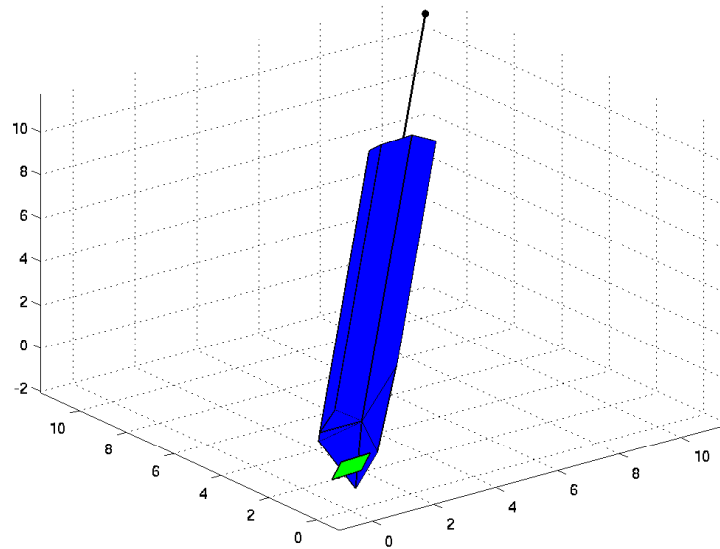
SEE ALSO

plus

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

ISADJACENT

Test if a polyhedron shares a facet with another polyhedron.

SYNTAX

```
ts = P.isAdjacent(Q)

ts = isAdjacent(P,Q)

[ts, iP, iQ] = isAdjacent(P,Q,fP,fQ)
```

DESCRIPTION

Return true if the polyhedron P has a facet to facet property with the polyhedron Q. Both polyhedrons must be in H-representation. If they are not, the irredundant H-representation will be computed.

Basically, the function tests if polyhedra P and S are adjacent by solving LP problem consecutively for each facet. The polyhedra are declared as adjacent if their intersection is of dimension $d - 1$ and if the facet of polyhedron P is also a facet for the polyhedron S.

If you want to test just specific facets, you can provide them in fP and fQ arguments.

INPUT

P   Polyhedron in H-representation
    Class: `Polyhedron`

Q   Polyhedron in H-representation
    Class: `Polyhedron`

fP Index of a facet to test from polyhedron P.
    Class: `double`

fQ Index of a facet to test from polyhedron Q.
    Class: `double`

OUTPUT

ts Logical statement if the polyhedron P is in a face to
    face property with Q.
    Class: `logical`
    Allowed values: `true`
                     `false`

iP Index of a facet from polyhedron P that is common with
    polyhedron Q.
    Class: `double`

iQ Index of a facet from polyhedron Q that is common with
    polyhedron P.

Class: `double`

EXAMPLE(s)

Example 1

Create two polyhedra P and Q.

```
H1 = [ 0.8905,0.23614,10;
-0.055625,0.030184,0;
-0.21887,-0.06688,0;
0,-1,10;
1, 0,10;
0, 1, 5;
0, 1,20];

H2 = [0.055625,-0.030184,0;
-0.053731,-0.010858,0;
0, 1, 10];

P = Polyhedron('H',H1);

Q = Polyhedron('H',H2);
```

Plot the polyhedrons.

```
plot([P,Q]);
```
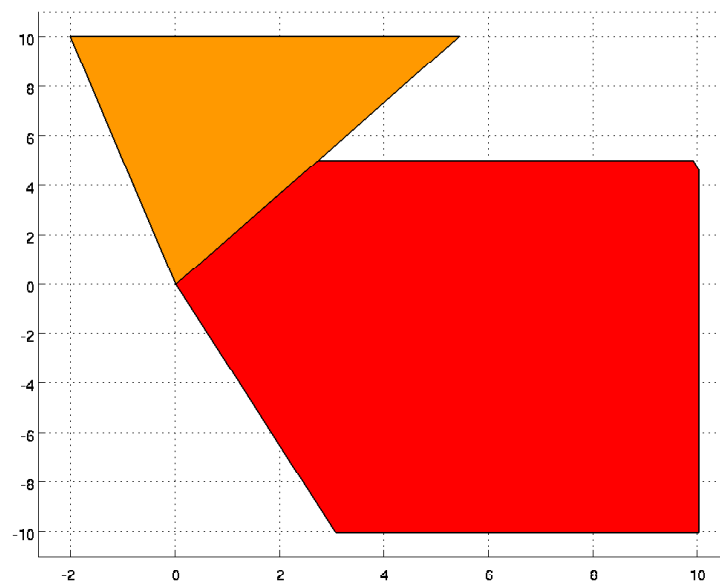

The polyhedrons are touching but are not adjacent because the common part is a facet of `P` but not a facet of `Q`.

```
P.isAdjacent(Q)
```

```
ans =
```

```
0
```

If the polyhedrons are touching on the facet, they are neighbors that can be checked by `isNeighbor` method.

```
P.isNeighbor(Q)
```

```
ans =
```

```
1
```

Example 2

Create two polyhedra P and Q.

```
H1 = [ 0.8905,0.23614,10;
-0.055625,0.030184,0;
-0.21887,-0.06688,0;
0,-1,10;
1, 0,10;
0, 1, 5;
0, 1,20];

H2 = [ 1.84286377 -1 0;
1.05619465 1 7.865634;
-4.9485172  -1 0];

P = Polyhedron('H',H1);

Q = Polyhedron('H',H2);
```

Plot the polyhedrons.

```
plot([P,Q]);
```
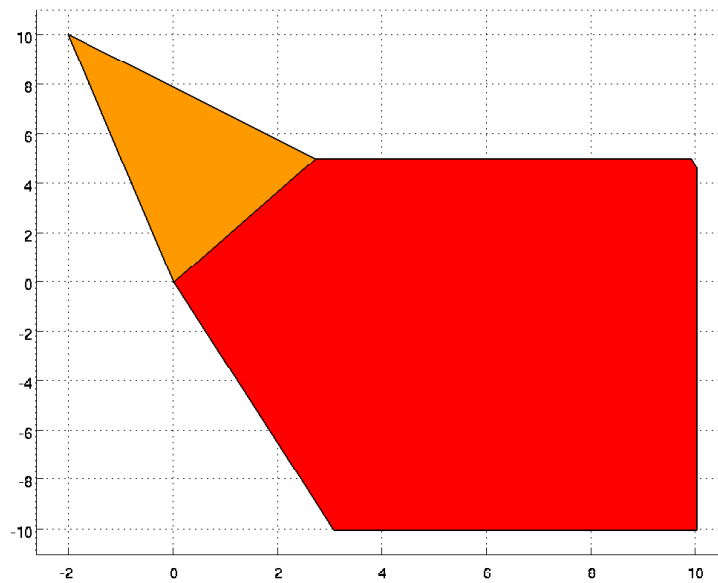
The polyhedra share the same facet so they are in face to face property and declared as adjacent. The indices of the common facet are returned in variables `iP` and `iQ`.

```
[ts,iP,iQ] = P.isAdjacent(Q)
```

```
ts =

1


iP =

2


iQ =

1
```

SEE ALSO

isNeighbor

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
      mailto:herceg@control.ee.ethz.ch

MLDIVIDE

Set difference between polyhedra

SYNTAX

```
R = P \ Q

R = mldivide(P,Q)
```

DESCRIPTION

Function computes the set difference between polyhedron P and Q which can be both single polyhedra or arrays of polyhedra in the same dimension
The set difference operation is defined as

$$R = P\backslash Q = \{x \mid x \in P\ ,x \notin Q\}$$

where the output R can comprise of multiple polyhedra.

INPUT

- P   Polyhedron in any format
     Class: `Polyhedron`

- Q   Polyhedron in any format
     Class: `Polyhedron`

OUTPUT

- R   Polyhedron (or array) `R = P`§ .
     Class: `Polyhedron`

EXAMPLE(s)

Example 1

Set difference of two full-dimensional polyhedra.

```
P = ExamplePoly.randHrep;

S = 0.2*P;
```

We can plot the polyhedra.

```
plot([P,S]);
```
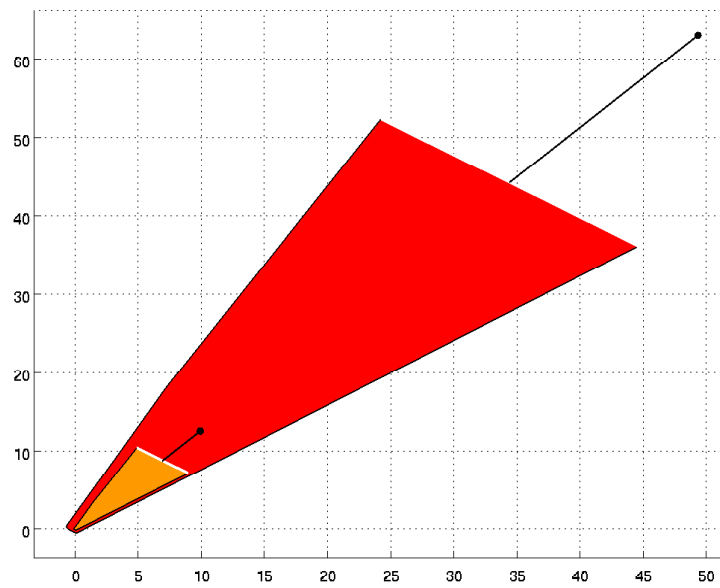
Compute the set difference.

```
Q = P \ S
```

```
Polyhedron in R^2 with representations:
H-rep (irredundant) : Inequalities   5 | Equalities   0
V-rep (irredundant) : Vertices   4 | Rays   2
Functions : none
```

Plot the polyhedra Q.

```
plot(Q,'color','limegreen');
```
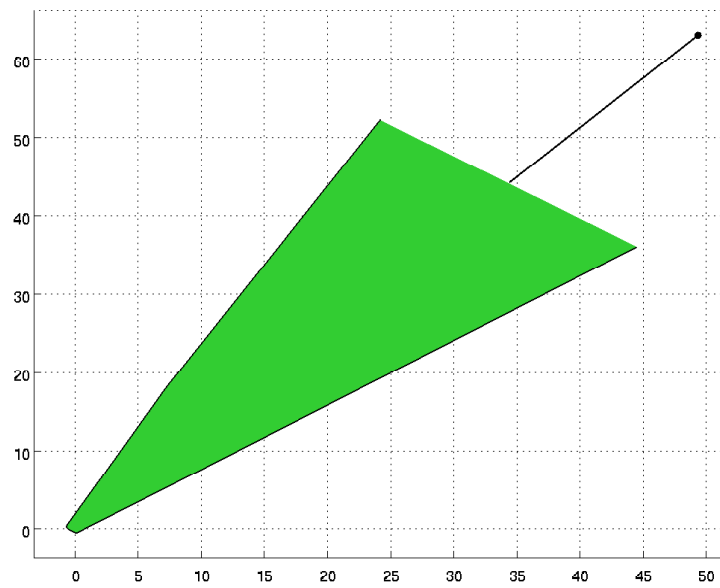
SEE ALSO

AUTHOR(s)

© **2003** Mato Baotic: ETH Zurich
    mailto:baotic@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

DISPLAY

Overload display for `Polyhedron` class.

SYNTAX

```
display(P)
```

```
P.display
```

DESCRIPTION

Default display for `Polyhedron` class.

INPUT

  P `Polyhedron` object.
     Class: `Polyhedron`

SEE ALSO

Polyhedron

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

OUTERAPPROX

Computes outer bounding hypercube of a polyhedron.

SYNTAX

```
B = P.outerApprox
```

```
P.outerApprox
```

DESCRIPTION

`B = P.outerApprox` computes the smallest axis-aligned hypercube `B` that contains the polyhedron `P`. The lower and upper bounds of the hypercube are also stored in `P.Internal.lb` and `P.Internal.ub`, respectively.

Use `P.outerApprox` if you only want the bounds to be written to `P.Internal`, but do not need the explicit bounding hypercube to be constructed.

INPUT

   P   Polyhedron.
       Class: `Polyhedron`

OUTPUT

   B   Bounding hypercube.
       Class: `Polyhedron`

EXAMPLE(s)

Example 1

We have a lower dimensional polyhedron in 3D.

```
P = Polyhedron('A',randn(9,3),'b',ones(9,1),'Ae',randn(1,3),'be',0.5);
```

Compute the bounding box

```
B = P.outerApprox;
```

Plot the sets such that the outer approximation is wired.

```
P.plot; hold on; B.plot('wire',true,'LineWidth',3)
```

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk
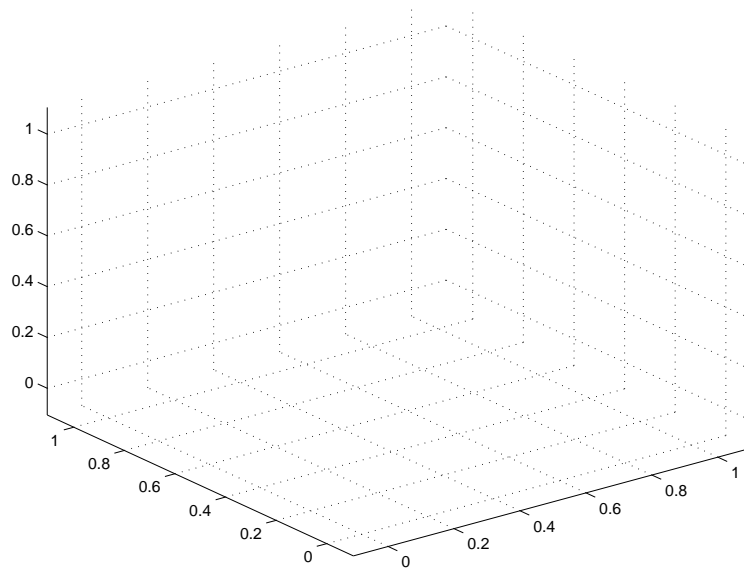
FPLOT

Plot single function over the polyhedron or array of polyhedra.

SYNTAX

```
h = P.fplot()

h = P.fplot('name', 'Prop1', value1, 'Prop2', value2)

h = fplot(P, 'name', 'Prop1', value1, 'Prop2', value2)
```

DESCRIPTION

Plot single function over a polyhedron or array of polyhedra. If there are more functions, then `name` indicates the string under which the function is attached to a set. If the function is vector valued, i.e. its range is greater than 1, than the first element of the function value is plotted by default. For vector valued functions use the `position` property if you want a different element of the function value to plot.

Figure properties, such as color, line width, etc, can be specified with "Property" - "Value" pairs.

INPUT

Polyhedron	Polyhedron in any format. Class: `Polyhedron`
name	If there are more functions associated to the polyhedron, the `name` indicates the function name. By default, only the first function is plotted. Functions can be accessed by referring to `Func` property. Class: `char` Default: `1`
dim	For vector valued functions, the `dim` indicates which element of the vector to plot. Class: `double` Default: `1`
Prop1	Specification of figure properties. Class: `char`

Allowed values: `position` For vector valued functions, the `position` indicates which element of the function value to plot.

`Color` The color of the plot specified by real RGB vector or a string name of the color (e.g. 'gray').

`Wire` Highlight the edges of the set. Default is false.

`LineStyle` Specify the type of the line to plot edges of the set. Accepted values are '-',':','-.','–', and'none'.

`LineWidth` Specify the width of the line. Default is 1.

`Alpha` Transparency of the color. The value must be inside [0,1] interval. Default is 1.

`Contour` Add contour graph. Default is false.

`ContourGrid` With how many grid point to do the contour graph. Default is 30.

`show_set` Plot the polyhedron with the function. Default is false.

`showgrid` Show the grid inside the set. Default is false.

`Grid` With how many gridpoints to grid the polyhedron. Default is 20.

`ColorMap` Color map given either as a M-by-3 matrix, or as a string. Default is 'mpt'. Other available options are 'hsv', 'hot', 'gray', 'lightgray', 'bone', 'copper', 'pink', 'white', 'flag', 'lines', 'colorcube','vga', 'jet', 'prism', 'cool', 'autumn', 'spring', 'winter', 'summer'.

`ColorOrder` Either 'fixed' for fixed ordering of colors, or 'random' for a random order. Default is 'fixed'.

value1   Assigns value to `Prop1`.

OUTPUT

h   Handle related to graphics object.
     Class: `handle`

EXAMPLE(s)

Example 1

Plot one function over one dimensional polyhedron.
The set is an interval [0, 1]

```
P = Polyhedron('lb',0,'ub',1);
```

Assign a quadratic function $f(x) = x^2 + 1$ under name "f" to it

```
Q = QuadFunction(1,0,1);
```

```
P.addFunction(Q,'f');
```

Plot the function "f" with the polyhedron in dark olive green color.

```
P.fplot('f','show_set',true,'Color','darkolivegreen','LineWidth',3)
```
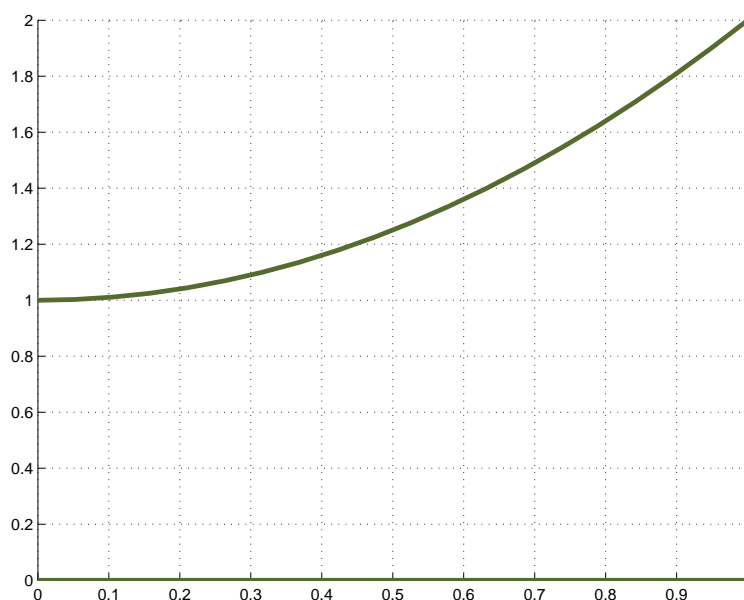

Example 2

We have two linear functions "friction", "gain" over one polyhedron.
Construct the functions. The first function maps from $R^2 \mapsto R^1$.

```
L1 = AffFunction([3 -1],2)
```

```
Affine Function: R^2 ->R^1
```

The second function is vector valued because it maps from $R^2 \mapsto R^2$.

```
L2 = AffFunction([1 0; -1 -2],[0.7;0.1])
```

```
Affine Function: R^2 ->R^2
```

Add functions to the set with names "friction" and "gain".

```
S = ExamplePoly.randVrep;
```

```
S.addFunction(L1,'friction');
```

```
S.addFunction(L2,'gain');
```

Plot first elements from both functions

```
S.fplot('friction','color','b'); hold on; S.fplot('gain','color','m'); hold off
```
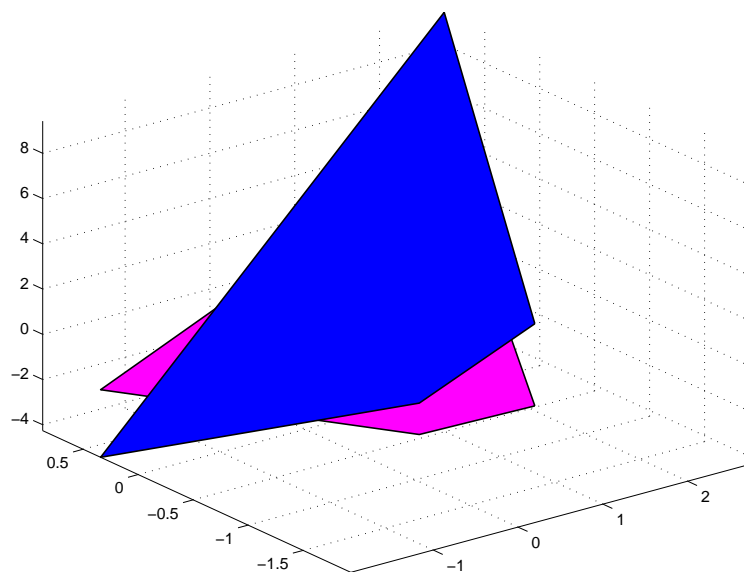
Plot the second element of the vector valued function "gain" over the wired polyhedron.

```
S.fplot('gain','position',2,'show_set',true,'wire',true);
```
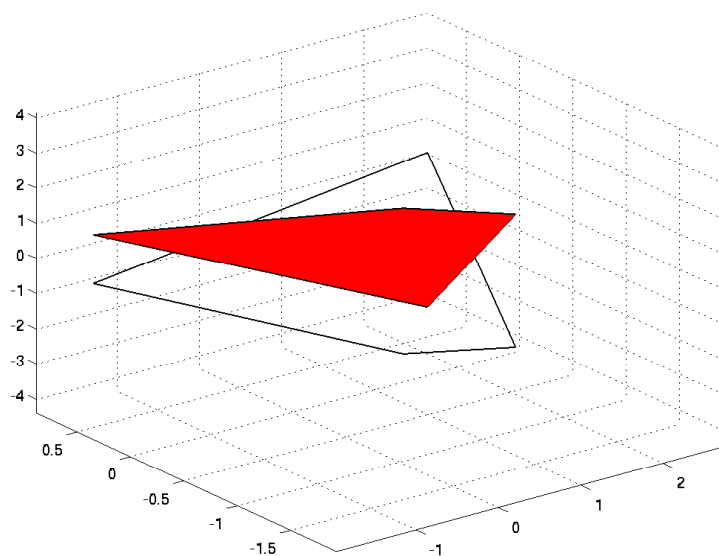

Example 3

Create a triangle from three points.

```
T = Polyhedron([0,0;4,1;1,3]);
```

Attach a quadratic function "energy" over the set T.

```
T.addFunction(QuadFunction(0.5*eye(2),[-1,0.6]),'energy');
```

Plot the function over the set and show grid

```
T.fplot('showgrid',true,'wire',true,'linewidth',3,'show_set',true,'color','lightsteelblue');
```
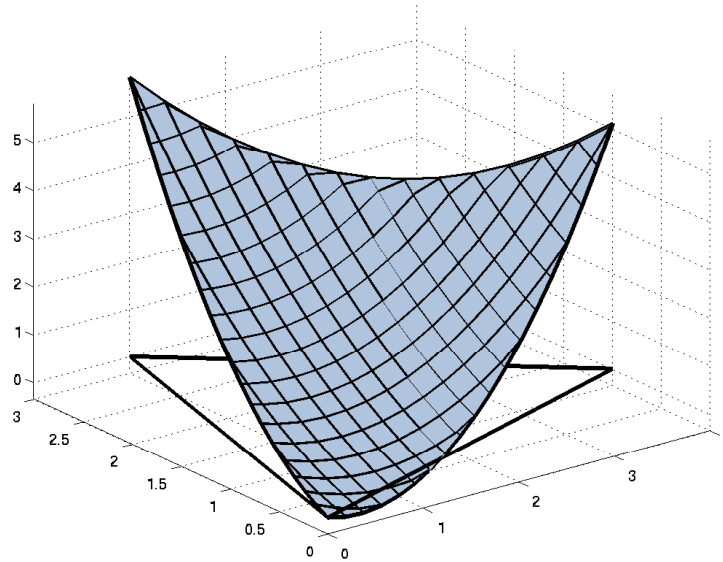

SEE ALSO

plot

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

AE

Get `Ae` matrix from equality description of the `Polyhedron`.

SYNTAX

`P.Ae`

DESCRIPTION

Get equality matrix A_e from $A_e x = b_e$ description of the polyhedron `P`.

INPUT

P   `Polyhedron` object.
    Class: `Polyhedron`

SEE ALSO

Polyhedron

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

ISEMPTYSET

Test if a polyhedron has a non-empty interior.

SYNTAX

```
tf = P.isEmptySet
```

```
tf = isEmptySet(P)
```

DESCRIPTION

Return true if the polyhedron P is empty or false otherwise.
A polyhedron is considered as empty if the diameter of the inscribed ball is less than `region_tol` or if there exist no feasible point inside the set.

INPUT

  P   Polyhedron in any format
      Class: `Polyhedron`

OUTPUT

  tf   True if the polyhedron P is empty and false otherwise.
      Class: `logical`
      Allowed values: `true`
                       `false`

EXAMPLE(s)

Example 1

Non-empty polyhedron

```
P = ExamplePoly.randHrep;
```

```
P.isEmptySet
```

```
ans =

0
```

Empty polyhedron is created at the time of construction.

```
Q = Polyhedron
```

```
Empty polyhedron in R^0
```

```
Q.isEmptySet
```

```
ans =
```

```
1
```

Array of 3 empty polyhedra

```
R(1,3) = Polyhedron
```

Array of 3 polyhedra.

```
R.isEmptySet
```

```
ans =
```

```
1    1    1
```

SEE ALSO

isEmptySet, isBounded, isFullDim

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

PLUS

Add a Polyhedron and a vector or Polyhedron.

SYNTAX

```
Q = P + S

Q = plus(P,S,method)

Q = P + x

Q = plus(P, x)
```

DESCRIPTION

Compute the Minkowski sum of P and S, or P and x.

$$P + S = \{x + y \mid x \in P, y \in S\}$$

or

$$P + y = \{x + y \mid x \in P\}$$

INPUT

P	Polyhedron in any format Class: `Polyhedron`
S	Polyhedron in any format or a point given as column real vector of the length `P.Dim` Class: `Polyhedron or double`
method	String selecting the projection method to be used. Can be 'vrep', 'fourier', or 'mplp'. Class: `string`

OUTPUT

Q	Polyhedron `Q = P+S`. Class: `Polyhedron`

EXAMPLE(s)

Example 1

Sum of Polyhedron and vector

```
P = Polyhedron('H',[sin([0:0.5:2*pi])'cos([0:0.5:2*pi])'ones(13,1)]);
```

```
Q = P + [2;2];

plot([P;Q],'alpha',0.1);
```
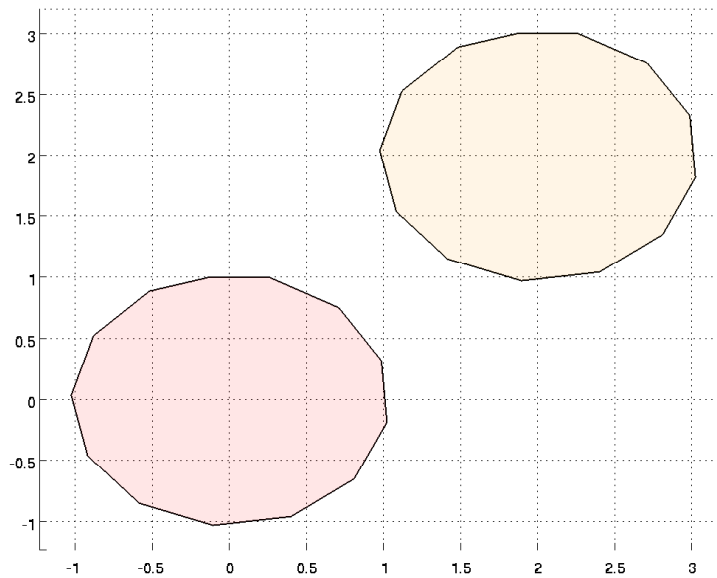

Example 2

Sum of two Polytopes

```
P = Polyhedron('H',[sin([0:0.5:2*pi])'cos([0:0.5:2*pi])'ones(13,1)]);

S = Polyhedron('lb',[-1;-1],'ub',[1;1]);

Q = P + S;

Q.plot('color','r','alpha',0.2); hold on;
S.plot('color','b','alpha',0.5);
P.plot('color','g','alpha',0.5);
```
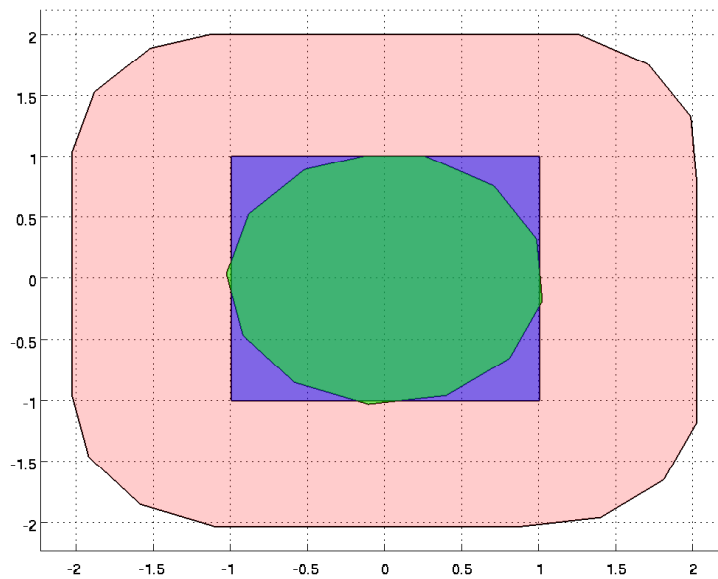
Example 3

Sum of a polytope and a cone

```
P = Polyhedron('H',[sin([0:0.5:2*pi])'cos([0:0.5:2*pi])'ones(13,1)]);

S = Polyhedron('R',[1 1;1 0.5]);

Q = P + S;

Q.plot('color','r','alpha',0.5); hold on;
S.minus([5;0]).plot('color','b','alpha',0.5);
P.minus([2.5;0]).plot('color','g','alpha',0.5);
```
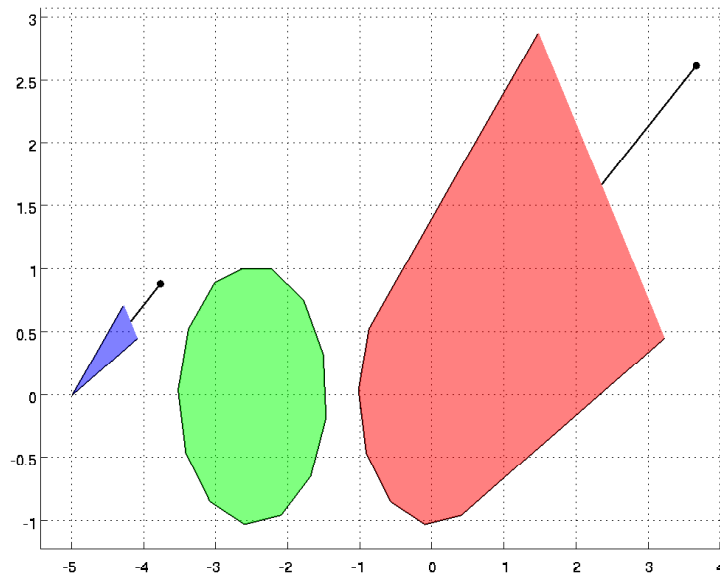
Example 4

Sum of lower dimensional polytopes

```
P = Polyhedron('lb',-ones(3,1),'ub',ones(3,1),'He',[1 1 1 0.2]);

S = Polyhedron('lb',-ones(3,1),'ub',ones(3,1),'He',[-1 1 1 0.4]);

Q = P + S;

P.plot('color','g','alpha',0.5); hold on;
S.plot('color','b','alpha',0.5);
Q.plot('color','r','alpha',0.2);
```
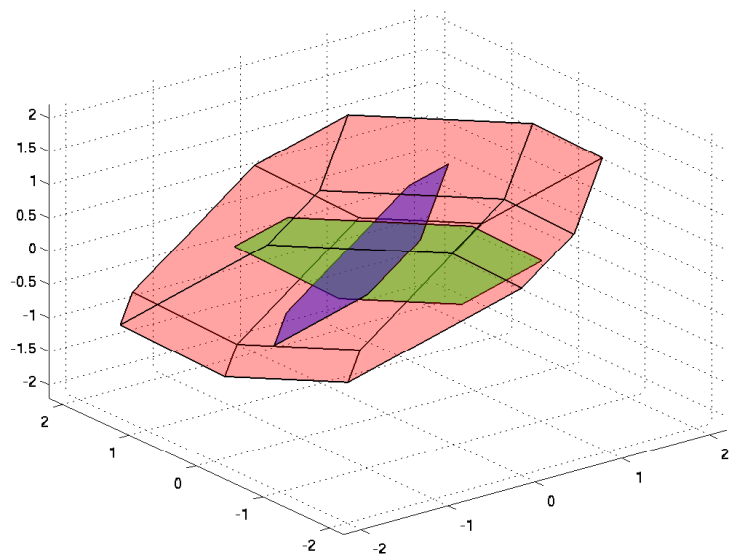
Example 5

Build a zonotope by adding 5 line segments.

```
Z = Polyhedron;

for (i=1:5) Z = Z + Polyhedron('V',[0 0 0;randn(1,3)]); end;

Z.plot('color','b','alpha',0.7);
```
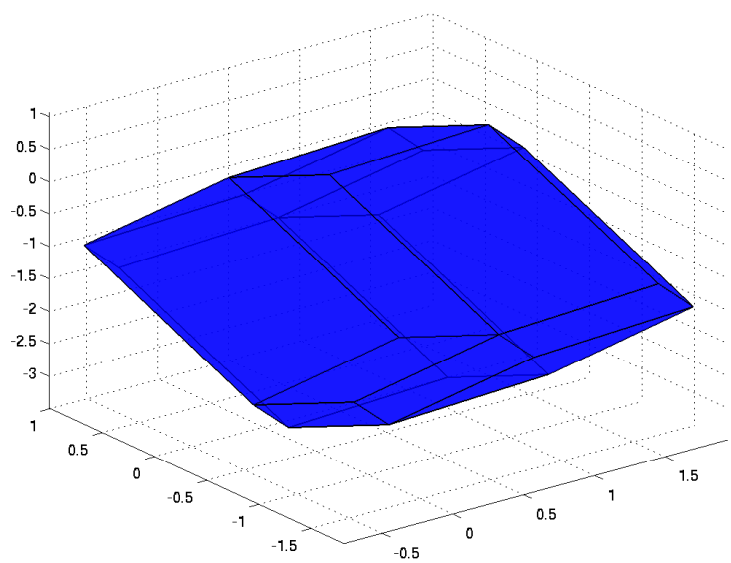
SEE ALSO

minus

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
   mailto:colin.jones@epfl.ch

POLYHEDRON

Create a Polyhedron object.

SYNTAX

```
P = Polyhedron(param, value, ...)

P = Polyhedron(dat)

P = Polyhedron(con, var)

P = Polyhedron(Q)
```

DESCRIPTION

Creates a Polyhedron:

$$P = \{x \mid Ax \le b, \ A_e x = b_e\}$$

or

$$P = \{x \mid x = V'\lambda + R'\gamma, \lambda, \gamma \ge 0, \ 1^T\lambda = 1\}$$

1. Polyhedral data specified as param, value pairs (detailed below).

2. Polyhedral data given in a structure. The same data as described below for param/value pair, but given as a structure.

3. Yalmip description `con[straints] var[iables]` specified

4. Polyhedron Q specified. The polyhedron object is a *handle* object. This means that executing $Q = P$ does NOT create a copy of P, both only another object by the same name. If you want to copy P, then call $Q = \texttt{Polyhedron}(P)$.

This class represents the following polyhedra:

- **Unbounded polyhedra** All polyhedra can be decomposed into the sum of a bounded polytope a cone:
  $$P = \text{conv}(V) + \text{cone}(R)$$

  and satisfy the Minkowski-Weyl theorem and can therefore be represented either as the intersection of a finite number of inequalities, or as the convex combination of a finite number of vertices (or rays). *MPT will store all irredundant polyhedra as a decomposition into a polytope and a cone.*

- **Lower-dimensional polyhedra** Theoretically there is no difference between full-dimensional and lower-dimensional polyhedra either in representation or in the algorithms that operate on them. However, experience has shown that if the affine hull of the polyhedron is not taken into account explicitly, then virtually all algorithms will fail. *MPT will store a polyhedron as the intersection of a full-dimensional polyhedron and an affine hull.*

- **Upointed Polyhedra** Operations on polyhedra with non-empty lineality space (i.e. unpointed polyhedra) adds significant complexity and difficulty. Thankfully, all convex sets can be decomposed into the Minkowski sum of their lineality space, with their restriction onto a linear subspace U that is perpendicular to lineal(P)

  $P = \text{lineal}(P) + (P \cap U)$ where $P \cap U$ has an empty lineality space. Therefore, it is always possible to represent an unpointed polyhedron as the lifting of a pointed one:

  $$P = \{Ey \,|\, y \in \tilde{P}\}$$

  where $\tilde{P} \in \mathbb{R}^m$ is pointed and $E \in \mathbb{R}^{n \times m}$ with n ¿ m.

  The requirement of dealing with an additional lifting operation for all polyhedra will add more coding complexity to MPT than desired. Therefore, in the interest of simplicity, MPT will handle unpointed polyhedra only in halfspace form, where the lifting map is not required, and any operation that cannot function on lifted polyhedra will return an error.

  MPT will have a limited ability to operate on unpointed polyhedra. It will handle unpointed polyhedra only in halfspace form, where the lifting map is not required, and any operation that does not function on unpointed polyhedra will throw an exception.

INPUT

P	Polyhedron in any format Class: `Polyhedron`
H	Inequality description (must be full-dimensional) $x \mid H \begin{pmatrix} x \\ -1 \end{pmatrix} <= 0$ Class: `double`
He	Affine set $x \mid H_e \begin{pmatrix} x \\ -1 \end{pmatrix} = 0$ Class: `double`
V	Points defining the set conv(V). Class: `double`
R	Rays defining the set cone(R). Class: `double`
irredundantVRep	If true, then the given V-representation is assumed irredundant. Class: `logical` Allowed values: `true` `false`
irredundantHRep	If true, then the given H-representation is assumed irredundant. Class: `logical` Allowed values: `true` `false`

lb	Add a constraint of the form lb $<= x$.
	Class: `double`
ub	Add a constraint of the form $x <=$ ub.
	Class: `double`

OUTPUT

P   Object of the `Polyhedron` class.

EXAMPLE(s)

Example 1

Create V-Polyhedron

```
P = Polyhedron([2 2;-5 -2; -5 3])
```

```
Polyhedron in R^2 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices   3 | Rays   0
Functions : none
```

Create H-Polyhedron

```
Q = Polyhedron([1 -1;0.5 -2;-1 0.4; -1 -2],[1;2;3;4])
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities   4 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : none
```

Plot the polyhedra

```
plot([P,Q]);
```
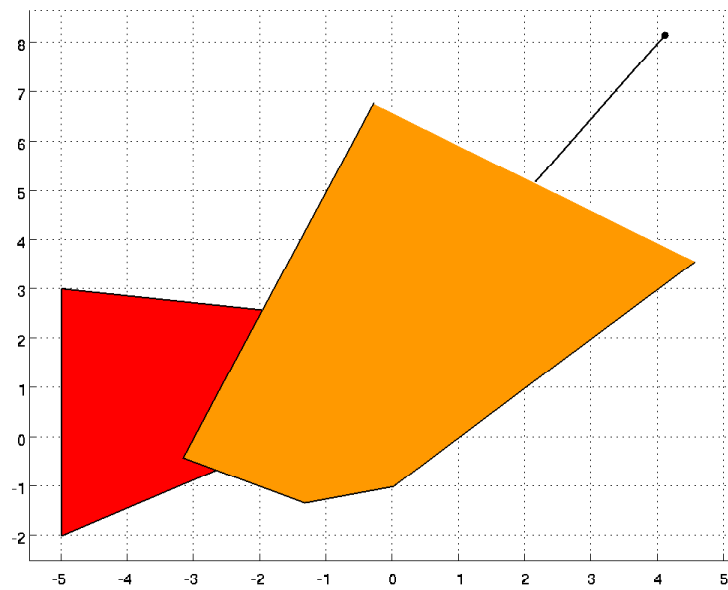
Example 2

Import polyhedron from YALMIP
Define yalmip variables and the set

```
x = sdpvar(3,1);

S = set([-1;-2;-3]<= x <= [1;2;3]);
```

Construct polyhedron

```
P = Polyhedron(S,x)
```

```
Polyhedron in R^3 with representations:
H-rep (redundant)   : Inequalities   6 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

Plot the polyhedron

```
plot(P);
```

SEE ALSO
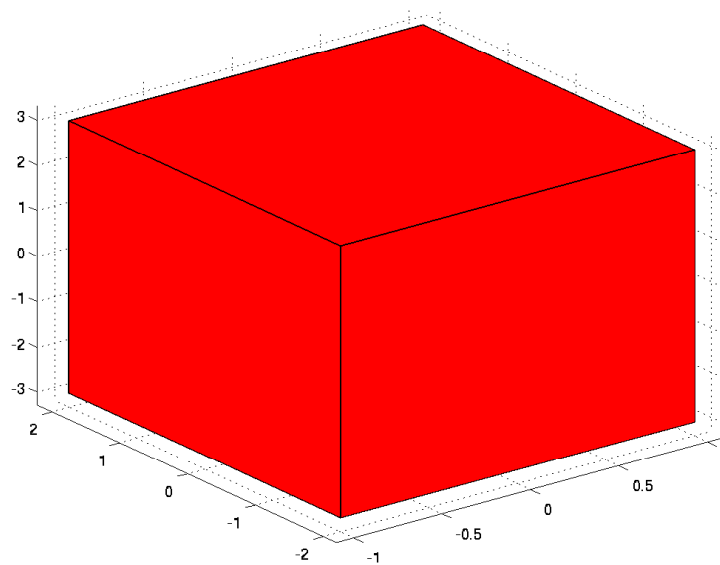
AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

PLOT

Plot the polyhedron.

SYNTAX

```
h = plot(P, 'Prop1', value1, 'Prop2', value2)

h = plot(P1, 'Prop1', value1, P2, 'Prop2', value2)

h = P.plot('Prop1', value1, 'Prop2', value2)
```

DESCRIPTION

Plot the polyhedron up to dimension three.
Figure properties, such as color, line width, etc., can be specified with "Property" - "Value"
pairs.

INPUT

Polyhedron	Polyhedron in any object with the dimension less or equal than 3. Class: `ConvexSet`
Prop1	Specification of figure properties. Class: `char`

Allowed values: `Color` The color of the plot specified by real RGB vector or a string name of the color (e.g. 'gray');

`Wire` Highlight or not the edges of the set. Default is false.

`LineStyle` Specify the type of the line to plot edges of the set. Accepted values are '-',':','-.','–', and'none'.

`LineWidth` Specify the width of the line. Default is 1.

`Alpha` Transparency of the color. The value must be inside [0,1] interval. Default is 1.

`Marker` Type of markings to use. Allowed values are ".", "o", "x", "+", "*", "s", "d", "v", "ˆ", "¡", "¿", "p", "h" or "none". Default is "none".

`MarkerSize` The size of the marker. Default is 6.

`ColorMap` Color map given either as a M-by-3 matrix, or as a string. Default is 'mpt'. Other available options are 'hsv', 'hot', 'gray', 'lightgray', 'bone', 'copper', 'pink', 'white', 'flag', 'lines', 'colorcube', 'vga', 'jet', 'prism', 'cool', 'autumn', 'spring', 'winter', 'summer'.

`ColorOrder` Either 'fixed' for fixed ordering of colors, or 'random' for a random order. Default is 'fixed'.

value1	Corresponding value to `Prop1`.

OUTPUT

    h   Handle related to graphics object.
        Class: `handle`

EXAMPLE(s)

Example 1

Plot two polyhedra.
The first polyhedron is an interval [-1, 1] in 1D.

```
P(1) = Polyhedron('lb',-1,'ub',1)
```

```
Polyhedron in R^1 with representations:
H-rep (redundant)   : Inequalities   2 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

The second polyhedron is in H-representation.

```
P(2) = Polyhedron('A',[0 -1;0.4 4;0.5 -2.1],'b',[1;2;3])
```

```
Array of 2 polyhedra.
```

Plot the sets with the dash-dotted line and the size 3.

```
P.plot('LineStyle','-.','LineWidth',3);
```
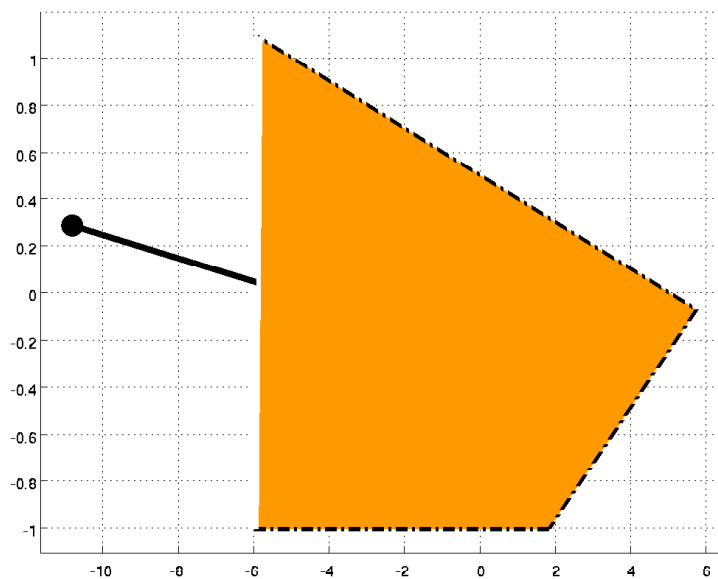
Example 2

Generate lower-dimensional polyhedron in 3D

```
P = ExamplePoly.randHrep('d',3,'ne',1);
```

Plot the polyhedron in 'darkred' color and some alpha transparency

```
P.plot('Color','darkred','Alpha',0.6);
```
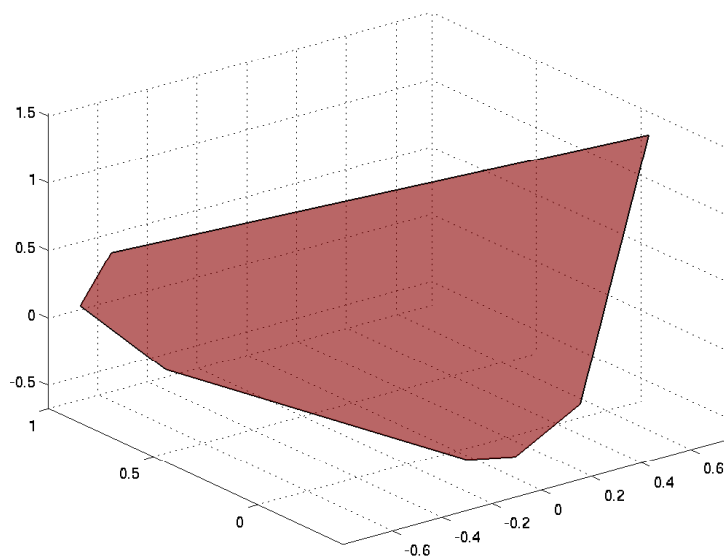

Example 3

Plot two polyhedra with different properties.

```
P1 = ExamplePoly.randVrep;
```

```
P2 = rand(2)*P1+rand(2,1);
```

Plot the polyhedron `P1` in lightgreen, dashed style and `P2` in wired frame and return handles.

```
h = plot(P1,'color','lightgreen','linestyle','--',P2,'wire',true);
```
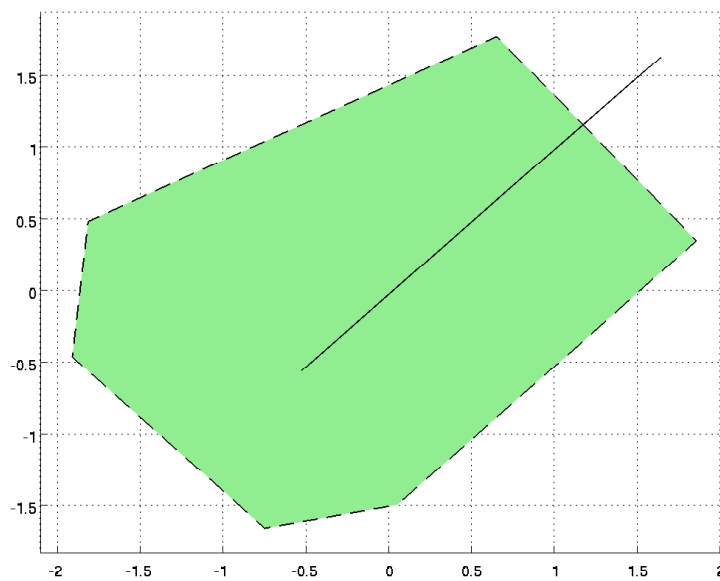
269

SEE ALSO

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
   mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

SEPARATE

Separate a point/polyhedron from another polyhedron.

SYNTAX

```
h = P.separate(x)

h = separate(P, x)

h = P.separate(S)

h = separate(P, S)
```

DESCRIPTION

Computes a separating hyperplane h between P and x/S :

$$h = \{y \mid h^T \begin{pmatrix} y \\ -1 \end{pmatrix} \leq 0 \ \forall y \in P, \ h^T \begin{pmatrix} x \\ -1 \end{pmatrix} = 0\} \geq 0\}$$

or

$$h = \{y \mid h^T \begin{pmatrix} y \\ -1 \end{pmatrix} \leq 0 \ \forall y \in P, \ h^T \begin{pmatrix} z \\ -1 \end{pmatrix} = 0\} \geq 0 \ \forall z \in S\}$$

INPUT

- P   Polyhedron in any format
  Class: `Polyhedron`

- S   Polyhedron in any format
  Class: `Polyhedron`

- x   Column vector of length `P.Dim`.
  Class: `Polyhedron`

OUTPUT

- h   Separating hyperplane $\{x \mid h^T \begin{pmatrix} x \\ -1 \end{pmatrix} = 0\}$, or [] if none
  exists.
  Class: `double`

EXAMPLE(s)

Example 1

Create a Polyhedron:

```
P = ExamplePoly.poly3d_sin;
```

Choose a point:

```
x = [2;2];
```

Separate and plot

```
h = P.separate(x);
P.plot; hold on; pplot(x,'ro');
Polyhedron('He',h).plot('linewidth',2);
```
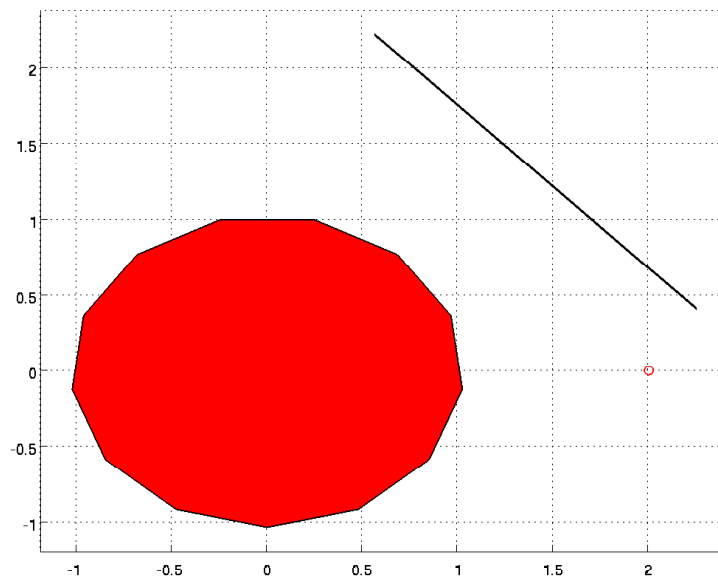

Example 2

Create two polyhedra:

```
P = ExamplePoly.poly3d_sin('d',3);
```

```
[U,s]=svd(randn(3));
```

```
S = ExamplePoly.poly3d_sin.affineMap(U(:,1:2)) + [2;2;2];
```

Separate and plot:

```
h = separate(P,S);
```

Plot the result

```
plot([P S]); hold on; Polyhedron('He',h).plot('alpha',0.3);
```

AUTHOR(s)
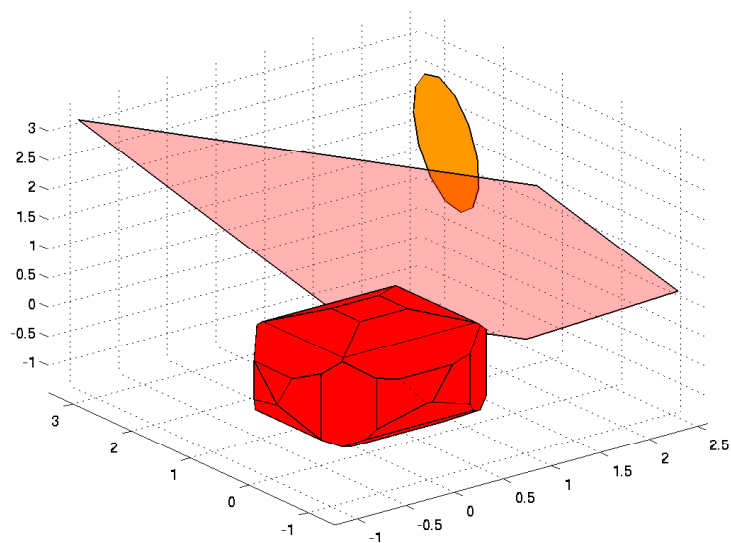
© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

AFFINEHULL

Compute the affine hull of the Polyhedron.

SYNTAX

```
hull = P.affineHull
```

```
hull = affineHull(P)
```

DESCRIPTION

Computes a minimum-rank matrix hull such that

$$P \subset \{x | hull \begin{bmatrix} x \\ -1 \end{bmatrix} = 0\}$$

Notes:

- If an H-rep is known, this function does not assume He is the affine hull, but rather searches for implicit equalities.

- The Polyhedron P is not modified after the call.

INPUT

P   Polyhedron in any format.
     Class: `Polyhedron`

OUTPUT

hull   Minimum-rank matrix representing the affine hull of P.
     Class: `double`

EXAMPLE(s)

Example 1

Create low-dimensional V-rep polyhedron:

```
P = ExamplePoly.randVrep('ne',1,'d',3);
```

Compute affine hull:

```
hull = P.affineHull
```

```
hull =

0.0588136143873345      -0.495678199302738      0.866512597426427      2.22044604925031e-16
```

Plot the result

```
P.plot; hold on; Polyhedron('He',hull).plot('alpha',0.1);
```
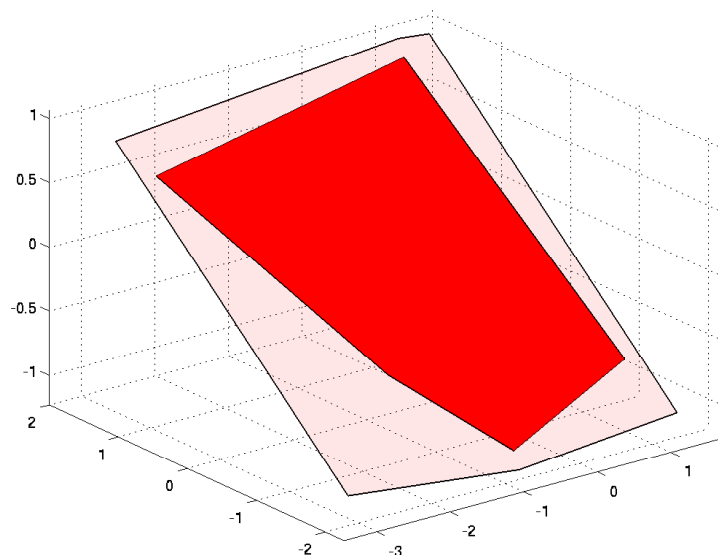
Example 2

Create H-rep polyhedron with implicit equality:

```
tmp = [randn(1,3) zeros(1,1)];

P = Polyhedron('H',[randn(20,3) ones(20,1);tmp;-tmp])


Polyhedron in R^3 with representations:
H-rep (redundant)   : Inequalities  22 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

Compute affine hull:

```
hull = P.affineHull
```

```
hull =

-1.68255489976003        0.841334198660793        -2.08184473266346                    0
```

Plot the result

```
P.plot; hold on; Polyhedron('He',hull).plot('alpha',0.1);
```
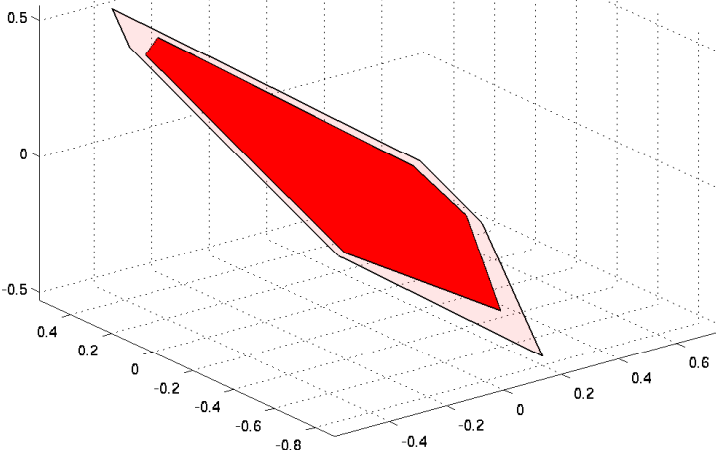
SEE ALSO

minVRep, affineMap

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

VOLUME

Compute the volume of the polyhedron.

SYNTAX

```
vol = P.volume
```

```
vol = volume(P)
```

DESCRIPTION

Computes the volume of the polyhedron P using V-representation. If the polyhedron is not in V-representation, it will be converted automatically.
Volume is `inf` if P is unbounded, and zero if P is not full-dimensional.

INPUT

P   Polyhedron in any format
    Class: `Polyhedron`

OUTPUT

vol   Volume of P.
      Class: `double`

EXAMPLE(s)

Example 1

Create polytope and compute volume:

```
P = ExamplePoly.randHrep('d',5,'n',40); disp(P.volume);
```

0.246212279404691

SEE ALSO

minVRep

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

LE

Test if a polyhedron is contained inside polyhedron.

SYNTAX

```
S <= P

tf = P.contains(S)

tf = contains(P, S)
```

DESCRIPTION

Check if the polyhedron `S` is contained inside the polyhedron `P`. The result it the logical statement if $S \subseteq P$ and false otherwise.

INPUT

P   Polyhedron in any format
    Class: `Polyhedron`

S   Polyhedron in any format.
    Class: `Polyhedron`

OUTPUT

tf   True if $S \subseteq P$ and false otherwise.
     Class: `logical`
     Allowed values: `true`
                       `false`

SEE ALSO

contains, lt, ge, gt

AUTHOR(s)

PROJECT

Project a point onto the given polyhedron.

SYNTAX

```
ret = P.project(x)

ret = project(P, x)
```

DESCRIPTION

Computes the projection of the point y onto this polyhedron by solving the optimization problem:

$$\min\{\|y - x\|_2^2 \mid x \in P\}$$

If P is a vector of m Polyhedra and x is a matrix of size $n \times$ `P.Dim`, then `ret` is a cell array with the structure fields that correspond the projection of each point in the matrix.

INPUT

P   Polyhedron in any format
     Class: `Polyhedron`

y   Vector of size `P.Dim` that represent single point y, or
     matrix of size P.Dim $\times n$ that corresponds to multiple
     points merged column-wise.
     Class: `double`

OUTPUT

`ret`	Optimal solution Class: `struct`
`ret.x`	Projected point or $[]$ if P is empty

`ret.exitflag`	Informs about the termination status of the optimization problem.
	Class: `double`
	Allowed values: `mptopt.OK`
	`mptopt.INFEASIBLE`
	`mptopt.UNBOUNDED`
	`mptopt.ERR`

`ret.dist`	Distance from y to the set P.

EXAMPLE(s)

Example 1

Create polyhedron:

```
P = ExamplePoly.poly3d_sin;
```

Choose a point and project:

```
y = [2;2];
```

```
ret = P.project(y);
```

Plot the result

```
P.plot; hold on; pplot([ret.x';y'],'or');
axis square; th=linspace(0,2*pi,100)';
pplot(ret.dist*[sin(th) cos(th)]+repmat(y',100,1),'k');
```
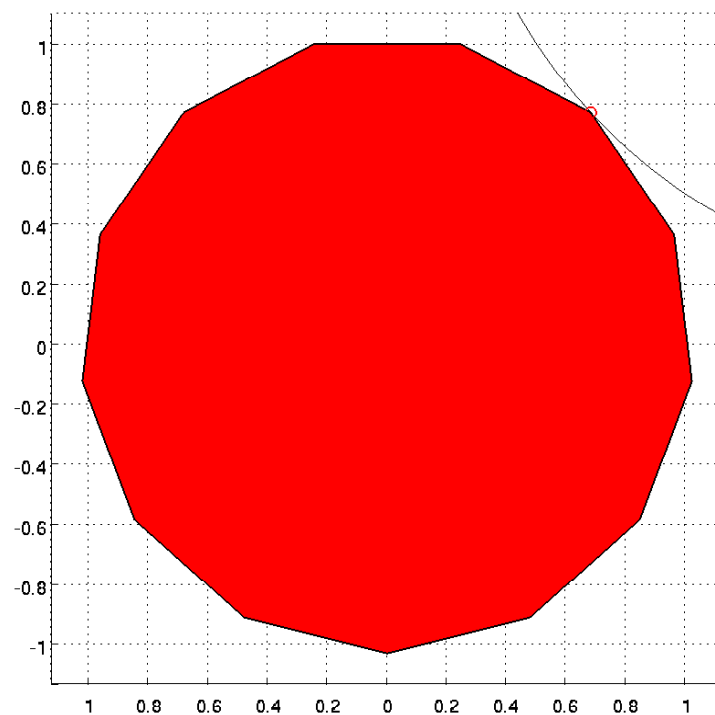

SEE ALSO

projection

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

AFFINEMAP

Compute the affine map of the Polyhedron.

SYNTAX

```
Q = P.affineMap(T)

Q = P.affineMap(T,method)

Q = affineMap(P,T,method)
```

DESCRIPTION

Computes an affine map $P \mapsto Q$ of polyhedron P to polyhedron Q based on the transformation matrix T. The polyhedron Q is given by

$$Q = \{y \in \mathbb{R}^n \mid y = Tx, x \in P \subset \mathbb{R}^d\}$$

The matrix T must be real with n rows and d columns.

- If $n < d$ then this operation is referred to as *projection*.

- If $n = d$ then this operation is referred to as *rotation/skew*.

- If $n > d$ then this operation is referred to as *lifting*.

INPUT

P	Polyhedron in any format. Class: `Polyhedron`
T	Transformation matrix. Class: `double`
method	Specific method to use in projection operation. Allowed methods are "vrep", "fourier", and "mplp". For details type "help Polyhedron/projection". Class: `string`

OUTPUT

Q	Polyhedron representing the affine map in H- or V-representation. Class: `Polyhedron`

EXAMPLE(s)

Example 1

Projection of the rectangle described in 2D.
Define the rectangle P in V-representation.

```
P = Polyhedron([0 0; 5 0; 5 3; 0 3]);
```

Compute the affine map of the rectangle with the matrix [-1 0.5]

```
Q = P.affineMap([-1 0.5])
```

```
Polyhedron in R^1 with representations:
H-rep             : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices   4 | Rays   0
Functions : none
```

We can see that Q is in dimension 1 while P is in dimension 2.

```
plot([P,Q],'LineWidth',3); axis([-6 6 -1 4]);
```
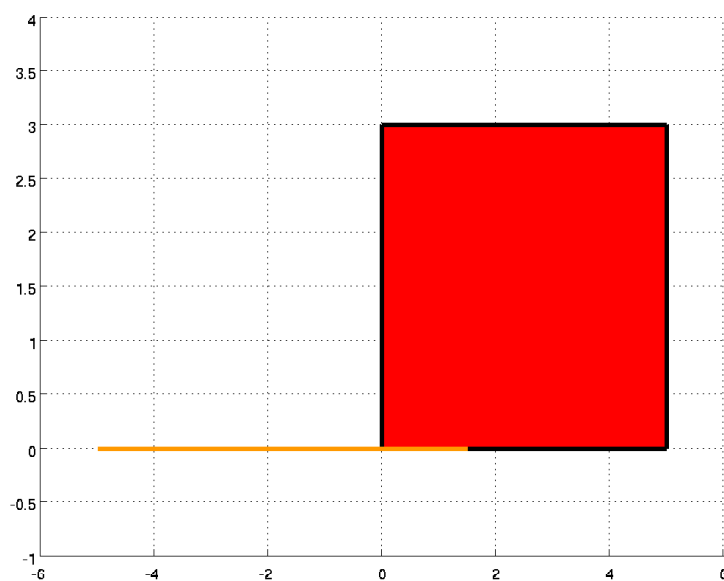

Example 2

Rotation of the rectangle described in 2D.
Compute the affine map of the rectangle with the matrix [-1 0.5; -2 3]

```
R = P.affineMap([-1 0.5; -2 3])
```

```
Polyhedron in R^2 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices   4 | Rays   0
Functions : none
```

We can see that `Q` remains in dimension 2.

```
plot([P,R]);
```
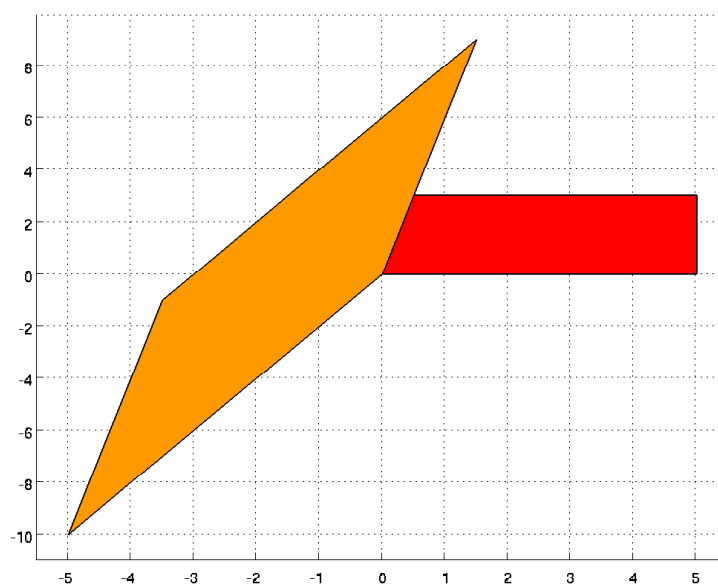

Example 3

Lifting of the rectangle described in 2D.
Compute the affine map of the rectangle with the matrix [-1 0.5; -2 3; 0.8 -1.4]

```
S = P.affineMap([-1 0.5; -2 3;-4 5])
```

```
Polyhedron in R^3 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices   4 | Rays   0
Functions : none
```

We can see that `S` is in dimension 3.

```
plot([P,S]);
```
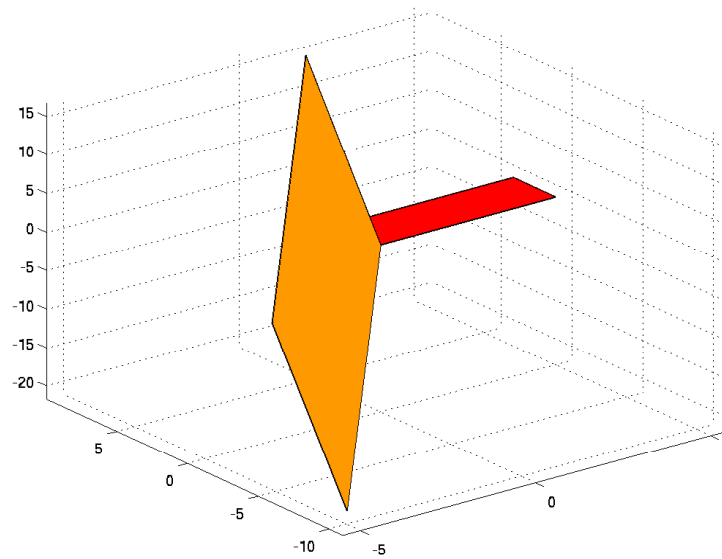
SEE ALSO

projection, affineHull

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

GE

Test if a polyhedron is contained inside polyhedron.

SYNTAX

```
S  >= P

tf = S.contains(P)

tf = contains(S, P)
```

DESCRIPTION

Check if the polyhedron `P` is contained inside the polyhedron `S`. The result it the logical statement if $P \subseteq S$ and false otherwise.

INPUT

P   Polyhedron in any format
    Class: `Polyhedron`

S   Polyhedron in any format.
    Class: `Polyhedron`

OUTPUT

tf   True if $P \subseteq S$ and false otherwise.
     Class: `logical`
     Allowed values: `true`
                      `false`

SEE ALSO

contains, le, lt, gt

AUTHOR(s)

MTIMES

Multiply two polyhedra, or a polyhedron with a matrix or scalar.

SYNTAX

`Q = P*S`

`Q = M*P`

`Q = a*P`

`Q = P*a`

DESCRIPTION

1. $Q = P * S$ where P and S are Polyhedra. Computes the product of the two polyhedra:

$$Q = \{(x, y) \mid x \in P, \ y \in S\}$$

2. $Q = M * P$ where P is a Polyhedron and M a matrix Computes the affine mapping `P.affineMap(M)`

3. $Q = a * P$ or $P * a$ where P is a Polyhedron and a is a scalar Computes the scaling

$$Q = \{ax \mid x \in P\}$$

Note that only single polyhedron `S` can be provided to compute a product.

INPUT

P   Polyhedron in any format
    Class: `Polyhedron`

S   Polyhedron in any format
    Class: `Polyhedron`

M   Matrix of size $n \times$ `P.Dim`
    Class: `double matrix`

a   Scaling factor
    Class: `double`

OUTPUT

Q   Polyhedron: Either $P * S$, $M * P$ or $a * P$ depending on
    the input
    Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create two polyhedra and take their product:

```
P = Polyhedron('V',[-1;0]);
```

```
S = Polyhedron('V',[1;2]);
```

```
plot([P S],'linewidth',2,'color','b'); hold on; plot(P*S); axis([-2 3 -2 3]);
```
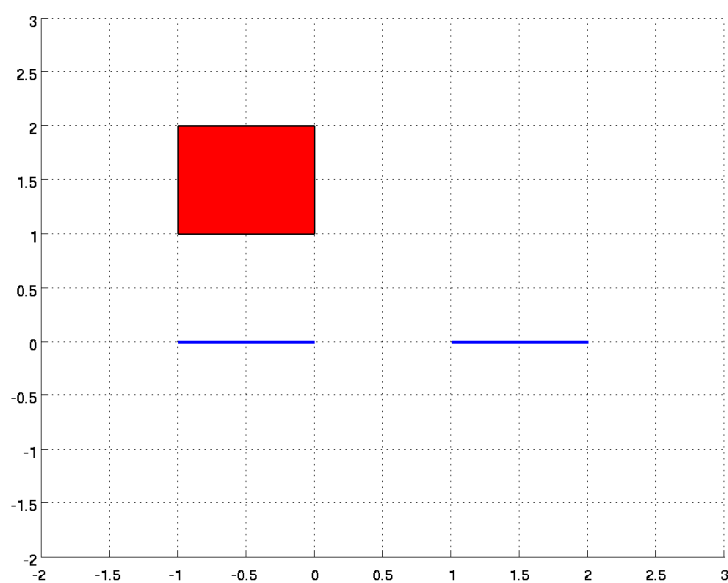

Example 2

Compute affine mapping:

```
P = Polyhedron('lb',-[1;1],'ub',[1;1]);
```

```
Q = randn(3,2)*P;
```

```
plot([P Q]);
```
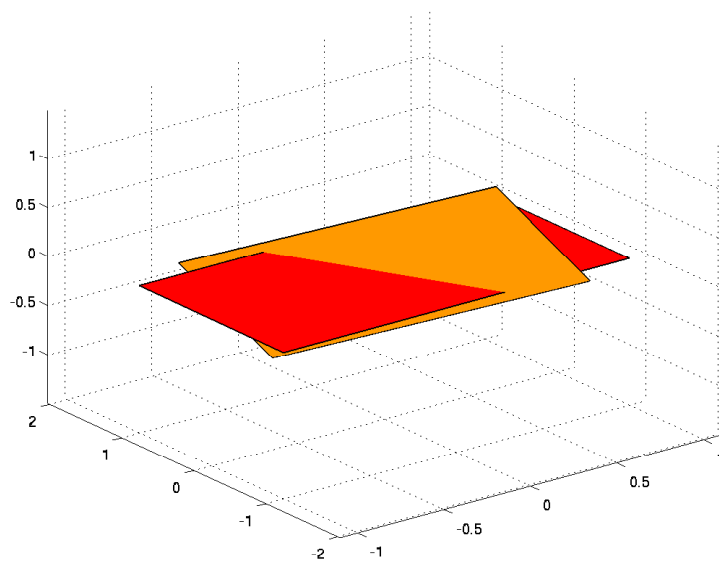
Example 3

Compute scaling:

```
P = ExamplePoly.poly3d_sin;
plot([1.5*P P P*0.5]);
```
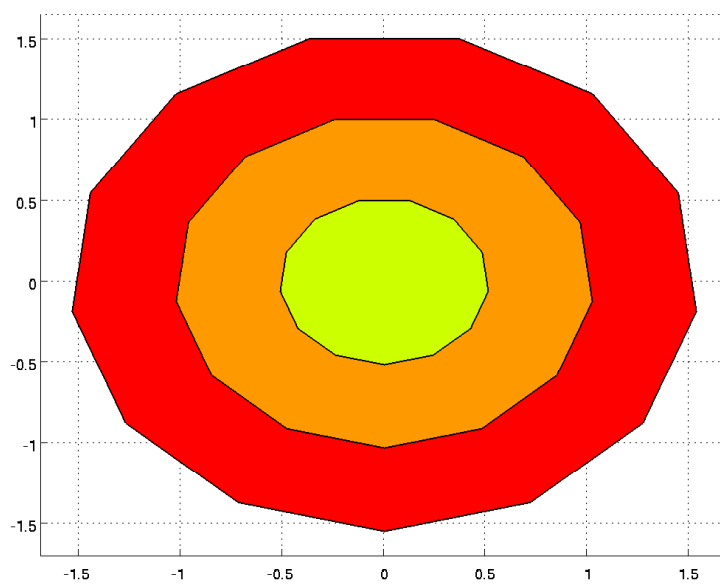

SEE ALSO

affineMap

LITERATURE

Fukuda: PolyFaq

AUTHOR(s)

INCIDENCEMAP

Compute the incidence map of this polyhedron.

SYNTAX

```
iMap = P.incidenceMap
```

```
iMap = incidenceMap(P)
```

DESCRIPTION

Computes map describing containment of vertices in facets.
Note: This function computes both irredundant V and H-representations of the polyhedron and can be time consuming.

INPUT

 P   Polyhedron in any format
     Class: Polyhedron

OUTPUT

 iMap   Incidence map or [] is P is empty.
               Class: struct

 iMap.V   Vertices of P
 iMap.R   Rays of P
 iMap.H   Inequalities of P
 iMap.He   Equalities of P
 iMap.incVH   Incidence   of V in   H. incVH(i,j) = 1   if
               [V(i,:);-1]*H(j,:)' = 0
               Class: sparse logical

 iMap.incRH   Incidence   of R in   H. incRH(i,j) = 1   if
               [R(i,:);0]*H(j,:)' = 0
               Class: sparse logical

EXAMPLE(s)

Example 1

Create a polytope:

```
P = ExamplePoly.poly3d_sin
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities  13 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
```

Functions : none

Compute the incidence map

```
iMap = P.incidenceMap;
full(iMap.incVH)
```

ans =

0	0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	1	1	0	0

SEE ALSO

minVRep, minHRep

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

MESHGRID

Generate X-Y grid for 2D bounded polyhedra.

SYNTAX

```
[X,Y] = P.meshGrid
```

```
[X,Y] = P.meshGrid(N)
```

```
[X,Y] = meshGrid(P,N)
```

DESCRIPTION

Generates X-Y grid points for plotting of two-dimensional polyhedra. Supported are only bounded and lower-dimensional polyhedra.
The output from this function is consistent with Matlab "meshgrid" function used for plotting functions over 2D polyhedra. The argument to this function is the number of points used to grid the polyhedron, default value is 20.

INPUT

P   Polyhedron given by V- or H-representation in dimension 2.
    Class: `Polyhedron`

N   The number of points for gridding. The value must be positive and integer-valued.
    Class: `double`
    Default: `20`

OUTPUT

X   Matrix with x-coordinates of the grid values.
    Class: `double`

Y   Matrix with y-coordinates of the grid values.
    Class: `double`

EXAMPLE(s)

Example 1

Define random polyhedron P in 2D.

```
P = ExamplePoly.randVrep;
```

Grid the polyhedron

```
[X,Y] = P.meshGrid;
```

Define Z coordinate as $z = 10/(1 + e^{(x-y)^2})$.

```
Z=10./(1+exp((X-Y).^2));
```

Plot the Polyhedron and the Z values.

```
plot(P,'color','limegreen','alpha',0.4); hold on; surf(X,Y,Z); view(3);
```
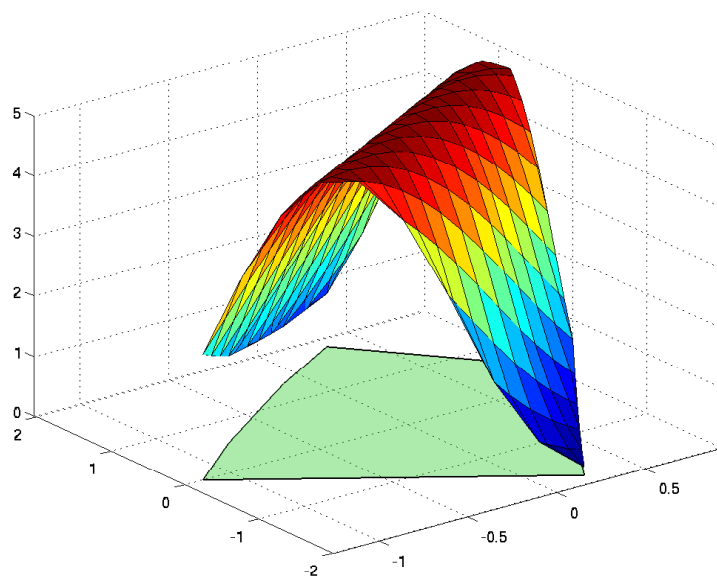

SEE ALSO

grid, plot, fplot

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
   mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

ISFULLSPACE

Test if a polyhedron represents the whole space $\mathbb{R}^n$.

SYNTAX

```
ts = P.isFullSpace

ts = isFullSpace(P)
```

DESCRIPTION

Return true if the polyhedron P is the whole space or false otherwise.

INPUT

P   Polyhedron in any format
    Class: Polyhedron

OUTPUT

ts   True if the polyhedron P is the full space and false otherwise.
     Class: logical
     Allowed values: true
                       false

EXAMPLE(s)

Example 1

An example of a polyhedron in the full space is when it contains a zero row in its H-representation and its right hand side is nonnegative.

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

P = Polyhedron(A,b);
```

The polyhedron P is the whole space.

```
P.isFullSpace
```

```
ans =

1
```

SEE ALSO

isEmptySet, isBounded, isFullDim

AUTHOR(s)

BE

Get `be` vector from equality description of the `Polyhedron`.

SYNTAX

`P.be`

DESCRIPTION

Get equality vector b_e from $A_e x = b_e$ description of the polyhedron `P`.

INPUT

P   `Polyhedron` object.
    Class: `Polyhedron`

SEE ALSO

Polyhedron

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

MINVREP

Compute an irredundant V-representation of a polyhedron.

SYNTAX

```
P.minVRep()
```

DESCRIPTION

Computes an irredundant V-representation of the polyhedron.

INPUT

P   Polyhedron in any format
     Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create redundant description of a polyhedron:

```
V = [1 1; 1 1; 1 1; 1 0];

P = Polyhedron(V)

Polyhedron in R^2 with representations:
H-rep            : Unknown (call computeHRep() to compute)
V-rep (redundant)   : Vertices   4 | Rays   0
Functions : none


P.V

ans =

1    1
1    1
1    1
1    0
```

Remove redundancies:

```
P.minVRep()

Polyhedron in R^2 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (irredundant) : Vertices   2 | Rays   0
Functions : none
```

```
P.V
```

```
ans =
```

```
1    1
1    0
```

SEE ALSO

minHRep

LITERATURE

Fukuda: PolyFaq

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

UPLUS

Unitary plus for a polyhedron.

SYNTAX

```
Q = +P;
```

DESCRIPTION

Creates a new copy `Q` of the polyhedron `P` by copying all fields.

INPUT

P   `Polyhedron` object.
    Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create H-polyhedron

```
P = ExamplePoly.randHrep;
```

Copy the polyhedron

```
Q = +P;
```

`P` and `Q` are the same.

```
P==Q
```

```
ans =

1
```

`Q` will not change if `P` is modified.

```
P.normalize
```

```
Polyhedron in R^2 with representations:
H-rep (irredundant) : Inequalities   5 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

The H-description is different because `P` was changed.

```
[P.H,Q.H]
```

```
ans =

Columns 1 through 5

-0.973873353820151        0.227091811210997         1.7721262710235       -0.549550768330796
0.128146518069415
0.525474934212241        -0.850809081706725         0.90121033341638       0.583076907496421
-0.944073819572629
-0.707421776611562        -0.706791645377718         0.475188542788992      -1.48871808326762
-1.48739201755454
0.930209391282812         0.367029274539865         1.30994430764807       0.710113694034025
0.280186930388549
0.999999574238128        -0.000922780343876191      0.386625296472837       2.58648252807323
-0.00238675625287499

Column 6

1
1
1
1
1
```

SEE ALSO

uminus

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

SHOOT

Maximize along a given ray within the polyhedron.

SYNTAX

```
s = P.shoot(r, x0)

s = shoot(P, r, x0)
```

DESCRIPTION

Computes the solution to the optimization problem:

$$\max_{\alpha} \{\alpha \mid x_0 + \alpha r \in P\}$$

The problem can be explained as to find the point that lies along the line given by the ray `r` which starts from the point `x0` such that it still lies inside the set P. Typically, the solution of this set is the point that lies on the boundary of the set, or can be `Inf` if the set is unbounded.

INPUT

P Polyhedron in any format
Class: `Polyhedron`

dir Vector of size `P.Dim`
Class: `double`

x0 Vector of size `P.Dim`
Class: `double`
Default: `Vector of all zeros`

OUTPUT

ret	Optimal solution Class: `struct`
ret.alpha	Maximum length of ray, [] if P is empty, `Inf` if unbounded.

ret.exitflag	Integer value informing about the termination status of the above optimization problem.
	Class: `double`
	Allowed values: `mptopt.OK`
	`mptopt.INFEASIBLE`
	`mptopt.UNBOUNDED`
	`mptopt.ERR`

ret.x	Extreme point of ray : $\alpha r + x_0$

EXAMPLE(s)

Example 1

Create polyhedron:

```
P = ExamplePoly.randHrep;
```

Choose direction and shoot:

```
r = randn(P.Dim,1); r = r/norm(r);
```

```
s = P.shoot(r);
```

Plot the result

```
P.plot; hold on; pplot([0 0;r'*s.alpha],'linewidth',3,'color','k');
```
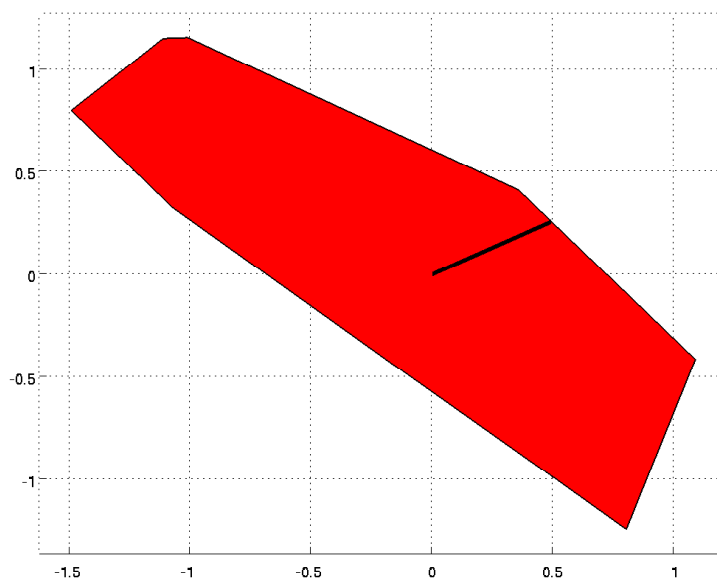

SEE ALSO

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

FACETINTERIORPOINTS

Compute points that lie on each of the facet of the Polyhedron.

SYNTAX

```
x = P.facetInteriorPoints

x = facetInteriorPoints(P)
```

DESCRIPTION

For each of the facet of the polyhedron compute a point in the relative interior of that facet. The output is a matrix formed by concatenating the interior point row-wise. Note that the polyhedron must be in the minimal H-representation in order to proceed.

INPUT

P Polyhedron in any format.
Class: `Polyhedron`

OUTPUT

x Matrix formed by interior points where the row of the matrix corresponds to facet of the polyhedron.
Class: `double`

EXAMPLE(s)

Example 1

Compute all facet interior points of a zonotope Z.

```
Z = ExamplePoly.randZono;

x = Z.facetInteriorPoints;
```

Plot the points on the zonotope.

```
Z.plot; hold on; plot(x(:,1),x(:,2),'kx','MarkerSize',5,'LineWidth',3);
```
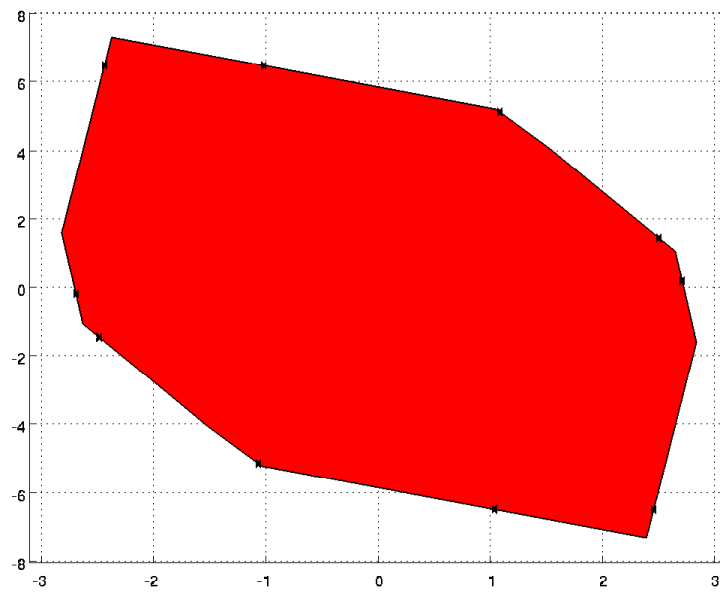
SEE ALSO

chebyCenter, interiorPoint

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

EQ

Returns true if the set covered by polyhedra P is the same as the set covered by S and false otherwise.

SYNTAX

```
tf = P.eq(S)
```

```
tf = P == S
```

DESCRIPTION

Returns true if P equals S and false otherwise by testing if both $P \subseteq S$ and $S \subseteq P$.

INPUT

P   Polyhedron in any format or array of Polyhedra in H-representation.
Class: `Polyhedron`

S   Polyhedron (or array of polyhedra) in the same dimension as P.
Class: `Polyhedron`

OUTPUT

tf   True if `S == P` and false otherwise.
Class: `logical`
Allowed values: `true`
                     `false`

EXAMPLE(s)

Example 1

Create a polytope:

```
P = ExamplePoly.poly3d_sin
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities  13 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

Create a V-representation of the same polytope

```
S = Polyhedron(Polyhedron(P).computeVRep)
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities 13 | Equalities   0
V-rep (redundant)  : Vertices  13 | Rays   0
Functions : none
```

Test equivalence:

```
P == S
```

```
ans =
```

```
1
```

Example 2

Create arrays of polytope in the same dimension

```
P(1) = ExamplePoly.randHrep;
```

```
P(2) = ExamplePoly.randHrep('ne',1);
```

Create a copy of this array

```
S = Polyhedron(Polyhedron(P))
```

```
Array of 2 polyhedra.
```

The arrays of polyhedra should cover the same set

```
P == S
```

```
ans =
```

```
1
```

SEE ALSO

neq, contains, le, lt, ge, gt

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

B

Get `b` vector from equality description of the `Polyhedron`.

SYNTAX

`P.b`

DESCRIPTION

Get equality vector b from $Ax \leq b$ description of the polyhedron `P`.

INPUT

P   `Polyhedron` object.
    Class: `Polyhedron`

SEE ALSO

Polyhedron

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

INVAFFINEMAP

Compute the inverse affine map of the Polyhedron.

SYNTAX

```
Q = P.invAffineMap(T)

Q = P.invAffineMap(T, t)
```

DESCRIPTION

Computes an inverse affine map `Q` of polyhedron `P` based on the transformation matrix T and vector t. The polyhedron `Q` is given by

$$Q = \{x \in \mathbb{R}^n \mid Tx + t \in P\}$$

The matrix `T` must be a square real matrix. The vector `t`, if omitted, defaults to a zero vector of corresponding dimension.

INPUT

- P   Polyhedron in any format.
  Class: `Polyhedron`

- T   Transformation matrix.
  Class: `double`

- t   Transformation vector.
  Class: `double`

OUTPUT

- Q   Polyhedron representing the affine map in H-representation.
  Class: `Polyhedron`

EXAMPLE(s)

Example 1

Inverse affine map of a unit box.

```
P = Polyhedron('lb',[-1; -1],'ub',[1; 1]);
```

Inverse affine map with the matrix [1 1; 0 1]

```
Q = P.invAffineMap([1 1; 0 1])
```

Polyhedron in R^2 with representations:

308

```
H-rep (redundant)   : Inequalities   4 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

Plot P and Q.

`plot([P,Q]);`
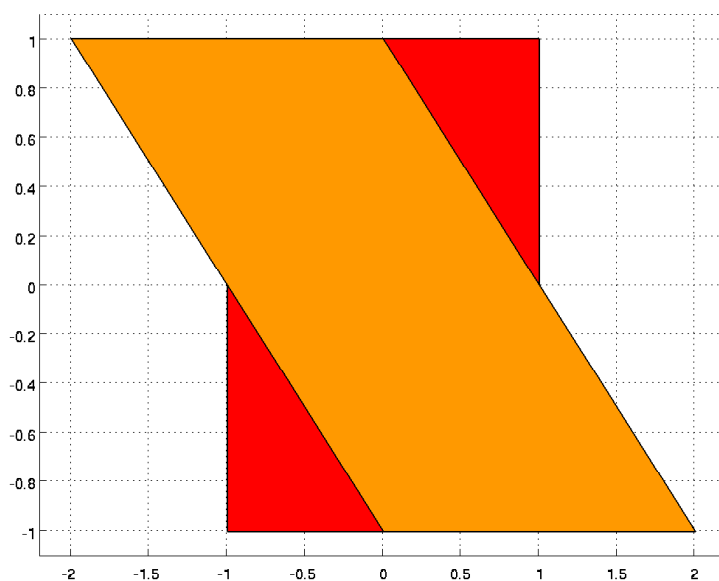
SEE ALSO

affineHull

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

TRIANGULATE

Triangulation of a polyhedron.

SYNTAX

```
T = P.triangulate;

T = triangulate(P)
```

DESCRIPTION

The polyhedron P is split into an array of triangular T_i polyhedra that built a partition $P = \cup T_i$. The `triangulate` function works over full-dimensional polyhedra, that are bounded and not empty.

INPUT

  P **Polyhedron** object.
      Class: **Polyhedron**

EXAMPLE(s)

Example 1

Create V-polyhedron

```
P = 10*Polyhedron(randn(5,2));
```

Generate the triangular partition

```
T = P.triangulate;
```

Plot the triangulation.

```
T.plot
```
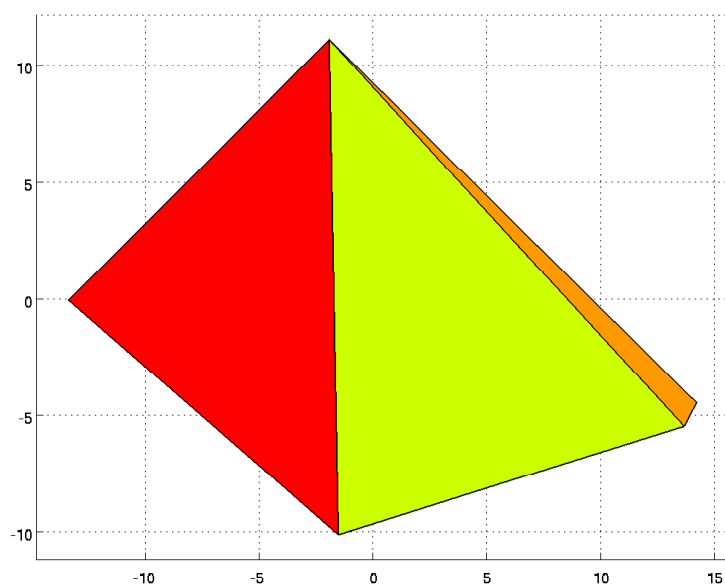
SEE ALSO

minVRep, computeVRep

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

LT

Test if a polyhedron is contained inside polyhedron.

SYNTAX

```
S < P

tf = P.contains(S)

tf = contains(P, S)
```

DESCRIPTION

Check if the polyhedron `S` is contained inside the polyhedron `P`. The result it the logical statement if $S \subset P$ and false otherwise.

INPUT

P   Polyhedron in any format
    Class: `Polyhedron`

S   Polyhedron in any format.
    Class: `Polyhedron`

OUTPUT

tf   True if $S \subset P$ and false otherwise.
     Class: `logical`
     Allowed values: `true`
                       `false`

SEE ALSO

contains, le, ge, gt

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

GT

Test if a polyhedron is contained inside polyhedron.

SYNTAX

```
S > P

tf = S.contains(P)

tf = contains(S, P)
```

DESCRIPTION

Check if the polyhedron `P` is contained inside the polyhedron `S`. The result it the logical statement if $P \subset S$ and false otherwise.

INPUT

P   Polyhedron in any format
     Class: `Polyhedron`

S   Polyhedron in any format.
     Class: `Polyhedron`

OUTPUT

tf   True if $P \subset S$ and false otherwise.
      Class: `logical`
      Allowed values: `true`
                       `false`

SEE ALSO

contains, le, lt, ge

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

PROJECTION

Compute the projection of the Polyhedron.

SYNTAX

```
p = P.projection(dims, method)

p = projection(P, dims, method)

p = projection(P, dims, method, solver)
```

DESCRIPTION

Computes the polyhedron p

$$p = \{x \mid x = y(\text{dims}),\ y \in P\}$$

INPUT

P   Polyhedron in any format
Class: `Polyhedron`

dims   Dimensions upon which to project
Class: `double`

method   Sets the method used to compute the projection. If omitted, then the method is chosen based on the properties of P and *dims*.
Class: `char`
Allowed values: `fourier` : Use fourier elimination. Good if projecting off a small number of dimensions. Projection will be highly redundant (use `q.minHRep()` to get irredundant form).
          `ifourier` : Fourier elimination with intermediate redundancy elimination. Dimensions are projected off one by one, followed by removal of redundant constraints. The result is still redundant, though.
          `mplp` : Use mplp algorithm to compute projection. Good if the projection is not very degenerate (dimensions of projected facets are equal to the faces that they were projected from)
          `vrep` : Compute vertex representation and then project

solver   If "mplp" method is selected, then this argument determines which solver to use for solving MPLP problem. By default the first parametric solver is selected from the list of available solvers.
Class: `char`

OUTPUT

p   Projection of P onto the dimensions *dims*
     Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create random polytope:

```
P = ExamplePoly.randVrep('d',3) + [0;0;5];
```

Compute projection:

```
p = P.projection(1:2)
```

```
Polyhedron in R^2 with representations:
H-rep            : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices  10 | Rays   0
Functions : none
```

Plot the result

```
plot([P,p-[0;1]])
```
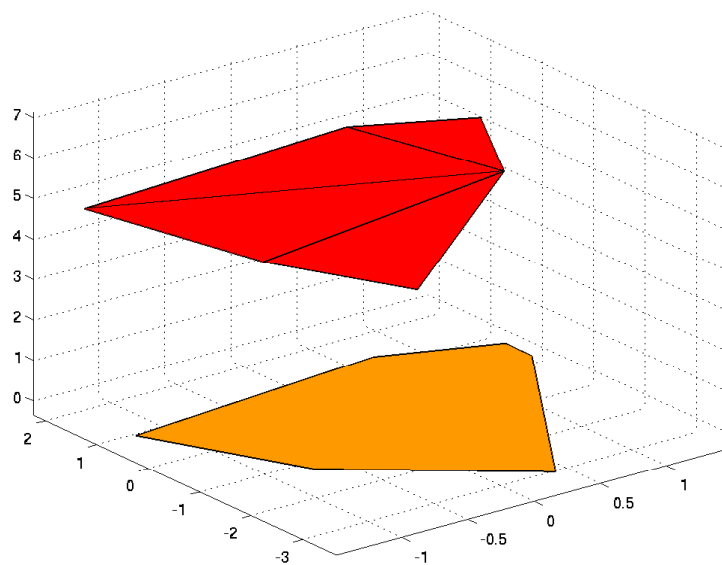
Example 2

Create random polyhedron:

```
P = ExamplePoly.randHrep('ne',1,'d',3,'nr',1) + [0;0;5];
```

Compute projection:

```
p = P.projection(1:2)
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities   4 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : none
```

Plot the result

```
plot([P,p]);
```
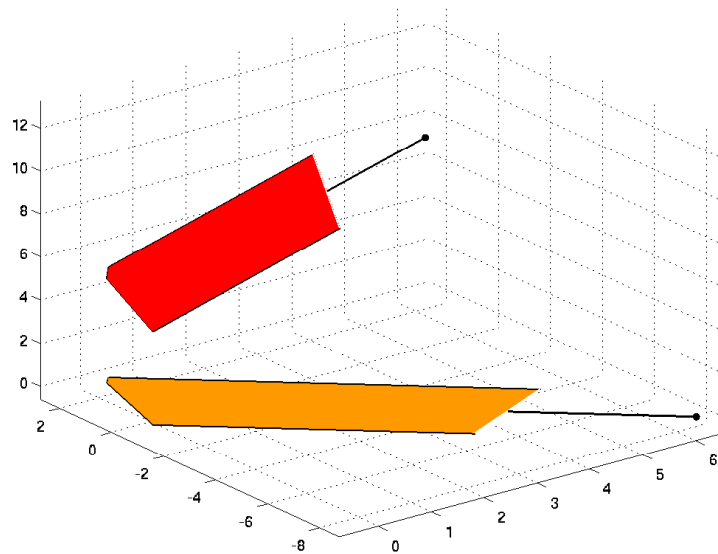

Example 3

Create zonotope:

```
Z = ExamplePoly.randZono('d',5,'n',8);
```

Project onto R3 and plot:

```
Z.projection(1:3).plot;
```
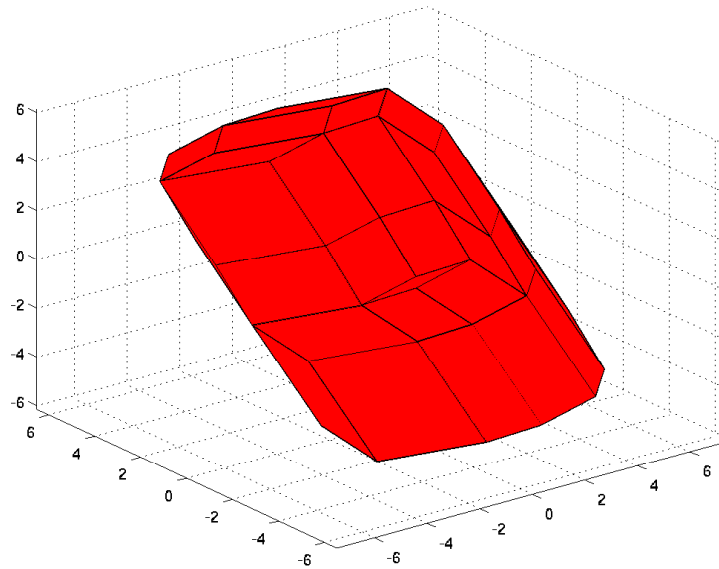
SEE ALSO

project

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

MINHREP

Compute an irredundant H-representation of a polyhedron.

SYNTAX

```
P.minHRep()
```

```
[P, sol] = P.minHRep()
```

DESCRIPTION

Computes an irredundant H-representation of the polyhedron:

$$P = \{x \mid Ax \le b\} \cap \{x \mid A_e x = b_e\}$$

Notes:

- If an H-representation is already known, then this function does redundancy elimination.

- Calling `P.minHRep()` will store the irredundant H-representation in P.

- For empty polyhedron P the result remains the same (no redundancy elimination).

INPUT

P   Polyhedron in any format
    Class: `Polyhedron`

OUTPUT

sol   Irredundant H-representation of P
        Class: `struct`

sol.H   Matrix of inequalities

$$H \begin{bmatrix} x \\ -1 \end{bmatrix} \le 0$$

        Class: `double matrix`

sol.He Matrix of equalities

$$H_e \begin{bmatrix} x \\ -1 \end{bmatrix} = 0$$

        Class: `double matrix`

EXAMPLE(s)

Example 1

Create redundant description of a polyhedron:

```
P = Polyhedron('H',[randn(20,3) ones(20,1)],'He',[randn(1,3) 0])
```

```
Polyhedron in R^3 with representations:
H-rep (redundant)   : Inequalities  20 | Equalities   1
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

Remove redundancies:

```
P.minHRep()
```

```
Polyhedron in R^3 with representations:
H-rep (irredundant) : Inequalities   6 | Equalities   1
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

Plot the result

```
plot(P);
```
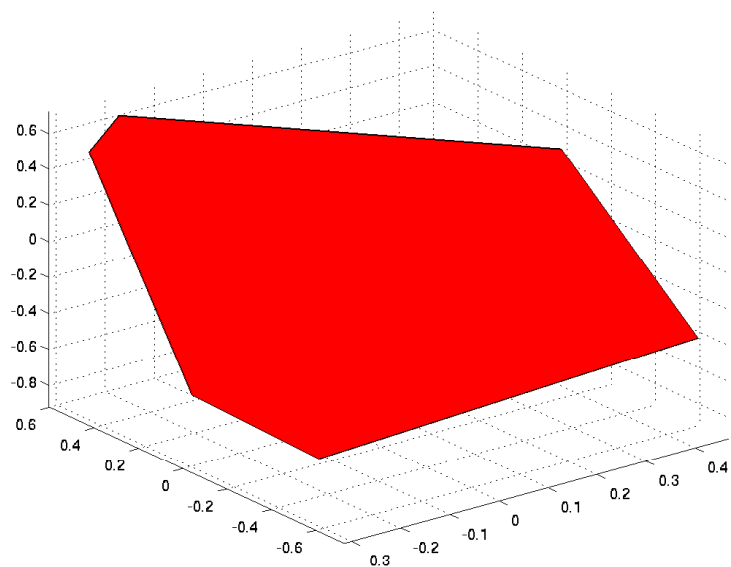

The polyhedron P has been permanently changed:

```
disp(P)
```

```
<a href="matlab:help Polyhedron">Polyhedron</a><a href="matlab:help handle">handle</a>

Properties:
irredundantVRep: 1
irredundantHRep: 1
hasHRep: 1
hasVRep: 1
A: [6x3 double]
b: [6x1 double]
Ae: [1.054537042432 0.942364915344315 -0.463781292098188]
be: 0
H: [6x4 double]
He: [1.054537042432 0.942364915344315 -0.463781292098188 0]
R: [0x3 double]
V: [6x3 double]
Dim: 3
Data: []

<a href="matlab:methods('Polyhedron')">Methods</a>,<a href="matlab:events('Polyhedron')">Events</a>,<a
href="matlab:superclasses('Polyhedron')">Superclasses</a>
```

Example 2

Create vertex representation of a polyhedron:

```
P = Polyhedron('V',randn(20,3),'R',-[1 0 0])


Polyhedron in R^3 with representations:
H-rep            : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices  20 | Rays    1
Functions : none
```

Compute inequality representation:

```
P.minHRep()


Polyhedron in R^3 with representations:
H-rep (irredundant) : Inequalities 18 | Equalities   0
V-rep (irredundant) : Vertices  10 | Rays    1
Functions : none
```

Plot the result

```
plot(P);
```
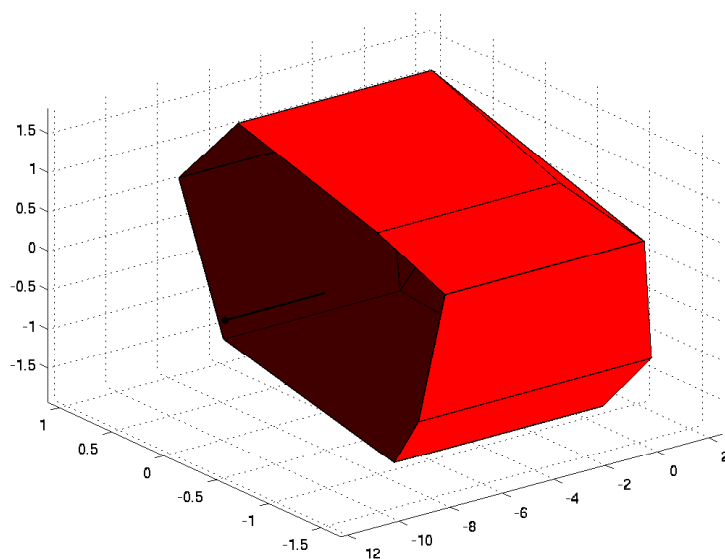
SEE ALSO

minVRep

LITERATURE

Fukuda: PolyFaq

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
        mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
        mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
        mailto:michal.kvasnica@stuba.sk

A

Get `A` matrix from inequality description of the `Polyhedron`.

SYNTAX

`P.A`

DESCRIPTION

Get inequality matrix A from $Ax \leq b$ description of the polyhedron `P`.

INPUT

P   `Polyhedron` object.
    Class: `Polyhedron`

SEE ALSO

Polyhedron

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

ISFULLDIM

Test if a polyhedron has a non-empty interior.

SYNTAX

```
tf = P.isFullDim
```

```
tf = isFullDim(P)
```

DESCRIPTION

Return true if the polyhedron P has a non-empty interior and false otherwise.
A polyhedron has a non-empty interior if and only if its dimension is the same as its dimension of representation. i.e. `rank affhull P = size(P.A,2)`, or equivalently if there exists a non-empty ball of dimension `size(P.A,2)` that is contained within it.

INPUT

P   Polyhedron in any format
     Class: `Polyhedron`

OUTPUT

tf   true if the polyhedron P has a non-empty interior and
     false otherwise.
     Class: `logical`
     Allowed values: `true`
                      `false`

EXAMPLE(s)

Example 1

Full-dimensional polyhedra:

```
Polyhedron('V',randn(30,5)).isFullDim
```

```
ans =

1
```

```
Polyhedron('V',randn(30,5),'R',randn(2,5)).isFullDim
```

```
ans =
```

1

Lower-dimensional polyhedra:

```
Polyhedron('V',[randn(10,2) zeros(10,1)]).isFullDim
```

```
ans =

0
```

```
Polyhedron('H',[rand(30,5) ones(30,1)],'He',[randn(1,5) 0]).isFullDim
```

```
ans =

0
```

SEE ALSO

isEmptySet, isBounded

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

NORMALIZE

Normalizes polyhedron in H-representation.

SYNTAX

`P.normalize`

`normalize(P)`

DESCRIPTION

Normalize polyhedron by scaling each inequality $a_i x \leq b$ and equality $a_{e,i} = b_{e,i}$ such that each row has unitary norm, i.e. $\|a_i\|_2 = 1$ and $\|a_{e,i}\|_2 = 1$. The normalization routine is very useful to prevent from numerical problems involved in badly scaled polyhedra.

INPUT

P   Polyhedron object.
    Class: Polyhedron

EXAMPLE(s)

Example 1

Badly scaled polyhedron

`P = Polyhedron([1000,0.4,-351; -0.1,-1e4,89.1],[780; 0.067]);`

The values are stored inside the polyhedron as they are provided.

`P.H`

```
ans =

1000                0.4              -351              780
-0.1            -10000              89.1              0.067
```

Normalize the polyhedron.

`P.normalize`

```
Polyhedron in R^3 with representations:
H-rep (redundant)  : Inequalities  2 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : none
```

The values of H-representation have changed:

```
P.H
```

```
ans =
```

```
0.943563707881528        0.000377425483152611       -0.331190861466416        0.735979692147592
-9.99960308263276e-06      -0.999960308263276        0.00890964634662579       6.69973406536395e-06
```

SEE ALSO

Polyhedron

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

CONTAINS

Test if a polyhedron/point is contained inside polyhedron.

SYNTAX

```
tf = P.contains(S)

tf = P.contains(S, fastbreak)

tf = contains(P, S, fastbreak)

P > S

P >= S

S < P

S <= P
```

DESCRIPTION

Check if the polyhedron or the point `S` is contained inside the polyhedron `P`. The result it the logical statement if $S \subseteq P$ and false otherwise.

`S` can be given as polyhedron, it can be a point given as a column (or set of points concatenated column-wise).

Notes:

- `P.contains(S)` with `S` a point assumes that `S` is a column vector. No automatic transposition is performed!

- `P.contains(S)` with `S` a matrix of points assumes that the points are stored column-wise. No automatic transposition is performed!

- If `P` is an array, `P.contains(S)` returns a $(n_p \times n_S)$ matrix of logicals, where n_p is the number of elements of `P` and n_x the number of points (i.e., the no. of columns) in `S`.

Remember that testing the set membership is numerically sensitive and depends on the settings of the absolute and relative tolerances.

INPUT

P	polyhedron or an array of polyhedra with `m` elements Class: `Polyhedron`
S	Polyhedron or a polyhedron array or a set of points. Multiple points must be concatenated in a matrix column-wise, where the number of rows corresponds to the dimension of `P` and the number of columns gives the number of points to test. No automatic transposition is performed! Class: `Polyhedron or double`

fastbreak   A logical flag. If true it indicates that the set member-
             ship test should be terminated as soon as at least one
             element of P contains the point S.
             Class: `logical`
             Allowed values: `0`
                              `1`
             Default: `0`

OUTPUT

tf   True if $S \subseteq P$ and false otherwise.

- If `S` is a single point or a single polyhedron: `status` is a $(m \times 1)$ vector of `true/false`, `status(i)=true` iff `P(i)` contains `S`.

- If `S` is a matrix of points (i.e., `S = [s_1, ..., s_n]`) `status` is a $(m \times n)$ matrix of `true/false`, `status(i, j)=true` iff `P(i)` contains point `S(:, j)`.

- If `S` is a polyhedron, then `P.contains(x)` iff $x \subseteq P$

Class: `logical`
Allowed values: `true`
                 `false`

EXAMPLE(s)

Example 1

Create two polytopes:

```
P = ExamplePoly.poly3d_sin('d',3);
```

```
S = ExamplePoly.poly3d_sin('d',3); S = S.affineMap(eye(3)*2);
```

Compare polytopes:

```
P.contains(S)
```

```
ans =

0
```

```
S.contains(P)
```

ans =

1

Plot the result

```
plot([P,S],'alpha',0.3);
```
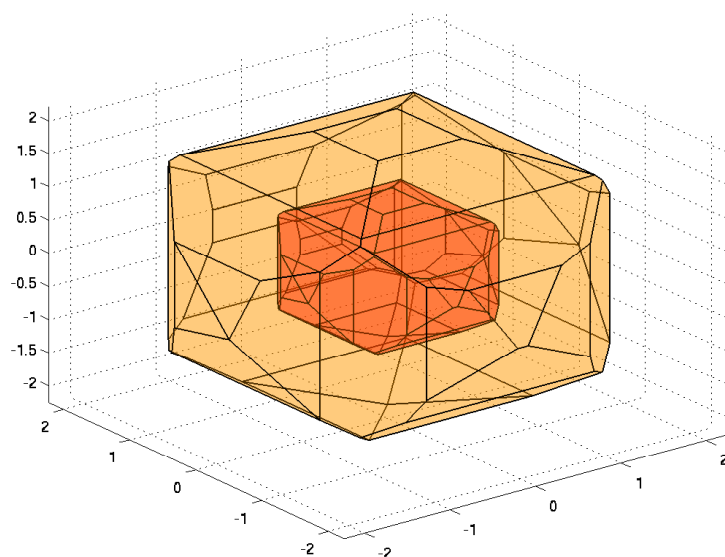

Example 2

Create two polyhedra:

```
P = ExamplePoly.randHrep;
```

```
S = P + ExamplePoly.randVrep('n',1,'nr',1);
```

Compare polytopes:

```
P.contains(S)
```

ans =

0

```
S.contains(P)
```

```
ans =

0
```

Plot the result

```
plot([S P]);
```
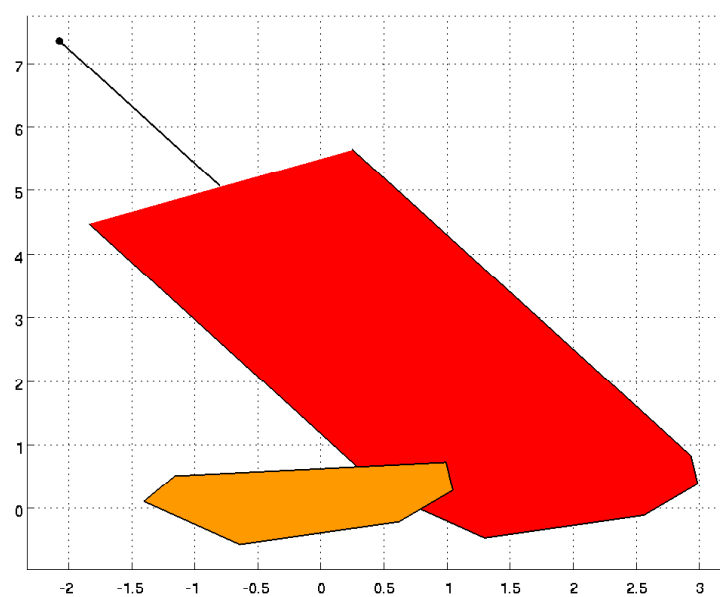

Example 3

Create random polyhedron.

```
P = ExamplePoly.randVrep;
```

Grid the polyhedron with 5 point over dimension.

```
x =   P.grid(5);
```

Test whether each grid point is contained in the polyhedron P.

```
P.contains(x')
```

```
ans =

1     1     1     1     1     1     1     1
```

SEE ALSO

interiorPoint, le, lt, ge, gt

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

UMINUS

Unitary minus for a polyhedron.

SYNTAX

```
Q = -P;
```

DESCRIPTION

Revert H- and V- representation of the polyhedron.

INPUT

P   **Polyhedron** object.
    Class: **Polyhedron**

EXAMPLE(s)

Example 1

Create H-polyhedron in 3D

```
P = ExamplePoly.randHrep('d',3,'ne',1);
```

Get the reversed polyhedron

```
Q = -P;
```

Plot the polyhedra

```
plot([P,Q]);
```
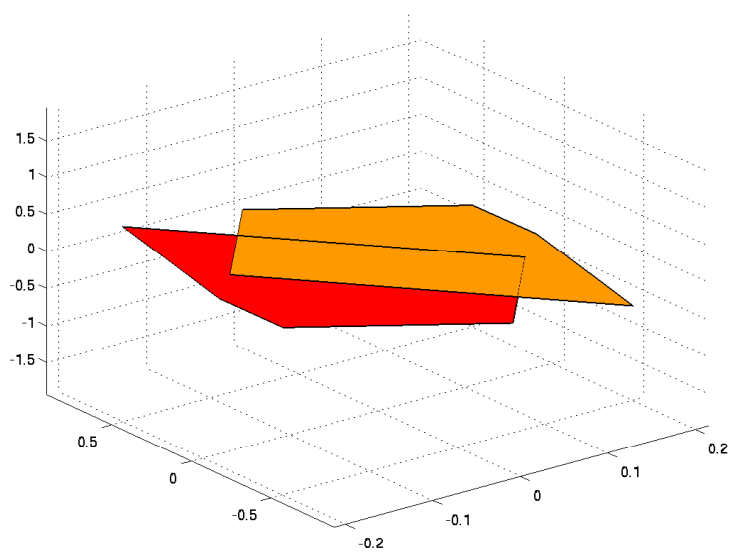
SEE ALSO

umplus

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

CHEBYCENTER

Compute the Chebyshev centre of the Polyhedron.

SYNTAX

```
s = P.chebyCenter

s = P.chebyCenter(facet)

s = P.chebyCenter(facet,bound)

s = chebyCenter(P,facet,bound)
```

DESCRIPTION

Computes the Chebyshev centre of a polyhedron via solving the following optimization problem

$$\max_{x,r} r$$
$$\text{s.t.} \quad a_i^T x + \|a_i\|_2 r \le b_i, \quad i = 1, \dots, m$$
$$A_e x = b_e$$

where x is the centre of the ball inscribed inside the H-representation of the polyhedron P and r is the radius of the ball. The polyhedron is given as $P = \{x \mid Ax \le b, A_e x = b_e\}$ with m inequalities and m_e equalities.

It is possible to compute the Chebyshev centre restricted to one or more facets of polyhedron by providing vector of indices `facet`. Note that the polyhedron must be in its minimal representation in order to have facets. If the facet indices are provided, the inequalities in the above optimization problem belonging to these indices change to equality constraints.

It is also possible to upper bound the radius r by specifying `bound` input. In this case the constraint $r <=$ bound is added.

INPUT

P   Polyhedron in H-representation.
        Class: `Polyhedron`

facet   Vector of indices specifying a subset of facets for computing the Chebyshev centre.
        Class: `double`
        Default: `[]`

bound   Upper bound on the radius of the inscribed ball.
        Class: `double`
        Default: `Inf`

OUTPUT

`s`	Structure with outputs from the Chebyshev centre optimization problem. Class: `struct`
`s.exitflag`	Informs if the optimization problem was terminated properly. Class: `double`
`s.x`	Centre of the ball inscribed in polyhedron `P`. Class: `double`
`s.r`	Radius of the ball inscribed in polyhedron `P`. Class: `double`

EXAMPLE(s)

Example 1

Chebyshev centre for 2D polyhedron.
Define the polyhedron `P` 2D.

```
P = ExamplePoly.randHrep;
```

Compute the centre of the ball inscribed ball

```
s = P.chebyCenter
```

```
s =

exitflag: 1
x: [2x1 double]
r: 0.728566101748245
```

Construct the circle out of the result.

```
x = sdpvar(2,1);
```

```
F = [norm(x-s.x)<=s.r];
```

```
circle = YSet(x,F);
```

plot the sets

```
plot(P); hold on; plot(circle);
```
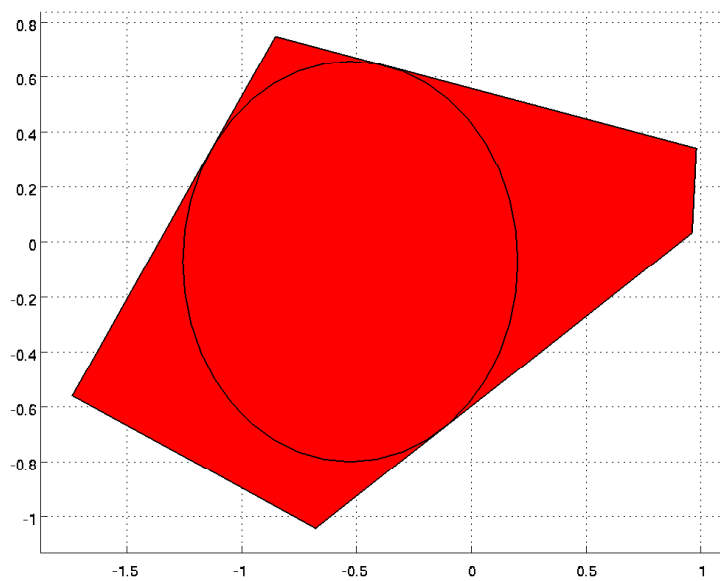
AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
     mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
     mailto:herceg@control.ee.ethz.ch

INTERSECT

Intersect two polyhedra.

SYNTAX

```
PnS = intersect(P, S)
```

DESCRIPTION

Compute the intersection of two polyhedra:

$$\texttt{PnS} = P \cap S$$

Note: If P or S are in V-rep, then this function will first compute their convex hull.

INPUT

P   Polyhedron in any format
Class: `Polyhedron`

S   Polyhedron in any format
Class: `Polyhedron`

OUTPUT

PnS   Intersection of P and S.
Class: `Polyhedron`

EXAMPLE(s)

Example 1

Intersect two polytopes.

```
P = ExamplePoly.randVrep;

S = ExamplePoly.randHrep;

PnS = intersect(P,S)


Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities  15 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : none


plot([P;S],'alpha',0.1);hold on;
plot(PnS,'color','r','alpha',0.6);
```
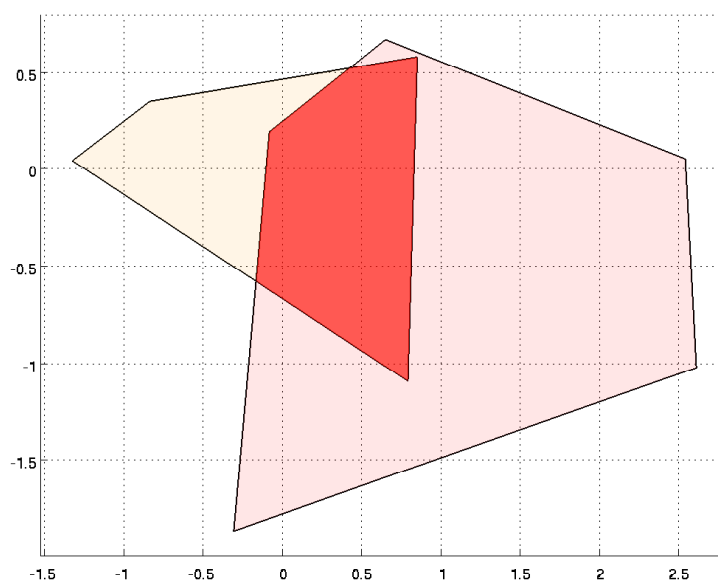
Example 2

Intersect a polytope and an unbounded polyhedron.

```
P = Polyhedron('V',[1 1;0 1;1 0],'R',-[1 0]);

S = Polyhedron('H',[randn(10,2) 4*ones(10,1)]);

PnS = intersect(P,S)


Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities  13 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none


plot(P,'color','b','alpha',0.1);hold on;
plot(S,'color','g','alpha',0.1); plot(PnS,'color','r','alpha',0.6);
```
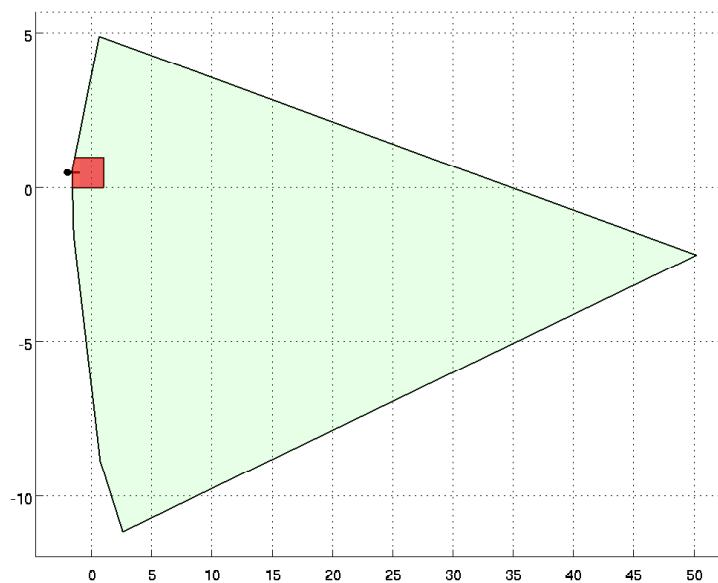
Example 3

Intersect a 3D polytope and a lower dimensional cone.

```
P = Polyhedron('lb',-0.5*ones(3,1),'ub',0.5*ones(3,1));

S = Polyhedron('R',[-1 0 .5;-1 2 2]);

PnS = intersect(P,S)


Polyhedron in R^3 with representations:
H-rep (redundant)   : Inequalities   8 | Equalities   1
V-rep               : Unknown (call computeVRep() to compute)
Functions : none


plot(P,'color','b','alpha',0.1);hold on;
plot(S,'color','g','alpha',0.1); plot(PnS,'color','r','alpha',1);
```

AUTHOR(s)
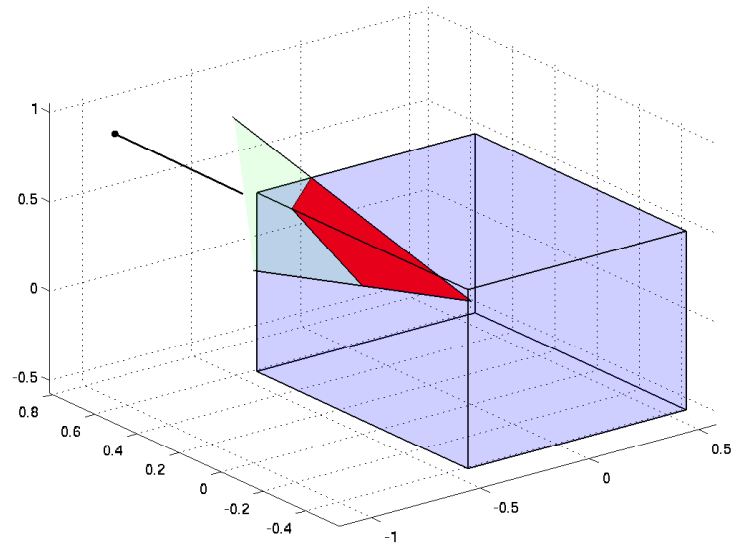
© **2010-2013** Colin Neil Jones: EPF Lausanne
   mailto:colin.jones@epfl.ch

SLICE

Slice the polyhedron through given dimensions at specified values.

SYNTAX

```
S = P.slice(dims, values)

S = slice(P, dims, values)

S = P.slice(dims, values, 'keepDim', true/false)
```

DESCRIPTION

Compute the intersection of P with a subspace spanning the dimensions `dims` at given `values`. If the argument `values` are omitted, the value is assumed to be zero, i.e. `values = zeros(size(dims))`.

For a polyhedron given in H-representation

$$P = x \mid Ax \leq b, \ A_{\text{eq}}x = b_{\text{eq}},$$

the `slice` operation over `dims` at given `values` returns a polyhedron in a reduced dimension `P.Dim-length(dims)`

$$S = x \mid A_{(:,\text{keep})}x <= b - A_{(:,\text{dims})}\text{values}.$$

This corresponds to the default choice `keepdim=false`.

Alternatively, by invoking `keepdim=true`, the polyhedron `S` will be returned in the same dimension as `P`

$$S = x \mid Ax \leq b, \ A_{\text{eq}}x = b_{\text{eq}}, x(\text{dims}) == \text{values}.$$

INPUT

P	Polyhedron in any format Class: `Polyhedron`
dims	Dimensions to cut through Class: `double`
values	Set of values at which to compute the slices. Class: `double` Default: `0`

OUTPUT

S	Polyhedron that represents the cut of the polyhedron `P` over the specified dimensions. Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create random symmetric polytope in dimension 3.

```
P = ExamplePoly.poly3d_sin('d',3);
```

Compute slices over the dimension 2 but preserve the dimension

```
S = P.slice(2,0,'keepDim',true);
```

Plot the result

```
P.plot('alpha',0.2,'color','lightblue'); hold on;
S.plot('color','blue','alpha',0.2,'linestyle','--','linewidth',3);
axis tight;
```
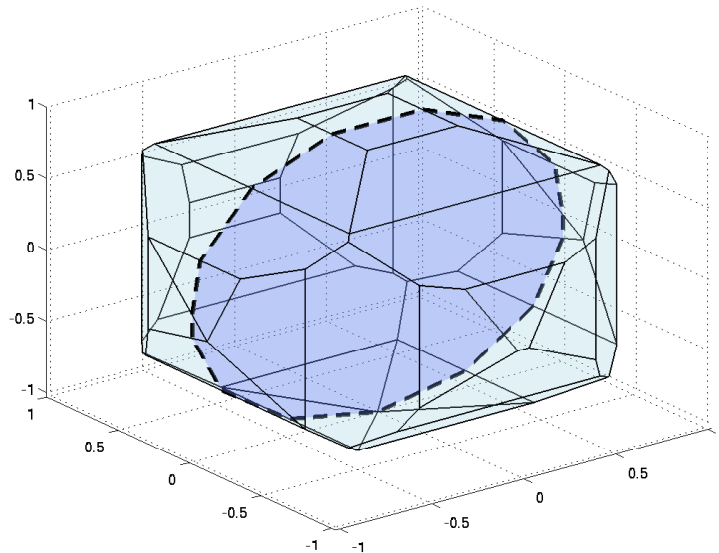

Example 2

Create a polytope in 4D from five points.

```
v = [-7,-3,-10,2; 1,-5,6,8; -2,6,-5,-5; 4,-4,9,-8; 3,-4,5,-3];
```

```
P = Polyhedron(v);
```

Slice it over dimensions 2 and 4 at values [1;-5]

```
S = P.slice([2,4],[1;-5])
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities   5 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

The resulting polyhedron S is in reduced dimension and can be plotted.

```
S.plot('color','green','alpha',0.5);
```
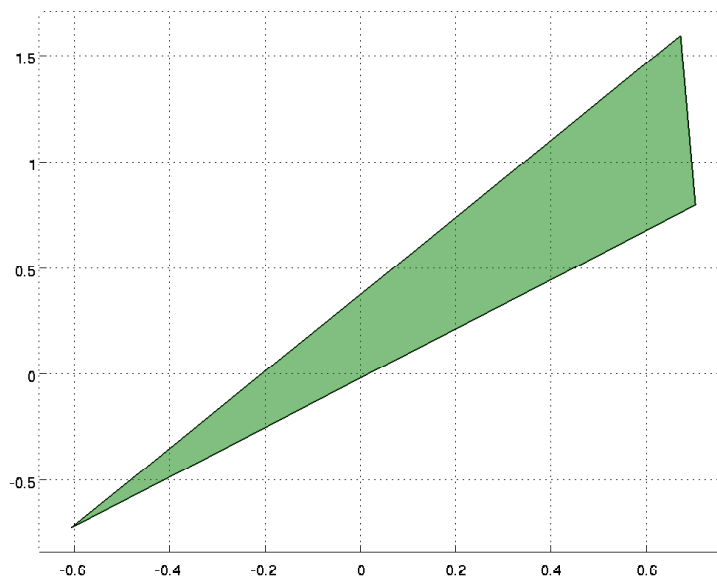

SEE ALSO

projection, project

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

INFNORMFUNCTION

Class representing inf-norm function.

SYNTAX

```
f = InfNormFunction(Q)
```

DESCRIPTION

The object for representing infinity-norm function given as $f = \|Qx\|_\infty$. The function is given as a maximum absolute value over elements of the vector $y = Qx$.

$$f = \max\left(|y_1|, \ldots, |y_n|\right)$$

where n is the dimension of the vector y. The weight `Q` does not need to be square. Function value is always scalar.

INPUT

Q Weighing matrix where the number of columns determines the dimension of the vector argument.
Class: `double`

OUTPUT

f The `InfNormFunction` object.
Class: `InfNormFunction`

EXAMPLE(s)

Example 1

Construct infinity-norm function $f = \max\left(|x_1|, |-x_2|, |2x_3|\right)$.

```
f = InfNormFunction(diag([1,-1,2]))
```

```
Inf-norm function in R^3
```

Evaluate the function in the point [1;-2;3].

```
f.feval([1;-2;3])
```

```
ans =

6
```

SEE ALSO

OneNormFunction, AffFunction, QuadFunction

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

HORZCAT

Vertical concatenation for `Function` class.

SYNTAX

```
F = vertcat(F1,F2)
```

```
F = [F1;F2]
```

```
F = [F1;F2;F3]
```

DESCRIPTION

The objects of `Function` class are concatenated into arrays only, no matrix concatenation is allowed. Note that only the objects of the same class can be concatenated into the same array.

INPUT

  F1 `Function` object.
      Class: `Function`

  F2 `Function` object.
      Class: `Function`

OUTPUT

  F An array of `Function` objects.
    Class: `Function`

SEE ALSO

Function, horzcat, disp

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

DISPLAY

Overload display for `Function` class.

SYNTAX

`display(F)`

`F.display`

DESCRIPTION

Default display for `Function` class.

INPUT

  F   `Function` object.
      Class: `Function`

SEE ALSO

Function, AffFunction, QuadFunction, setHandle

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

SETHANDLE

Assign function handle to existing `Function` object

SYNTAX

```
setHandle(F,@fun)
```

```
F.setHandle(@fun)
```

DESCRIPTION

Overwrites function handle of the `Function` object with a new one. This method is suitable for specification of functions where the parameters are stored under `Data` property. Look at examples for better explanation.

INPUT

F   Existing `Function` object which function we want to overwrite.
    Class: `Function`

F   Representation of the new function given as `function_handle` class.
    Class: `function_handle`

OUTPUT

F   Modified `Function` objects.
    Class: `Function`

EXAMPLE(s)

Example 1

Construct anonymous function $f(x) = k * x$ where k is a parameter. Assume that the value of k is not going to change in the future.
It suffices to declare the value of k before constructing the object, i.e.

```
k=2;
```

Then we can construct the object with this value of the parameter as

```
F1 = Function(@(x) k*x)
```

```
Function: @(x)k*x
```

Evaluating this function will always take $k = 2$ value, e.g in this case we get 2

```
feval(F1.Handle,1)
```

```
ans =
```

```
2
```

Changing the value of k does not affect the function

```
k=3;
```

```
feval(F1.Handle,1)
```

```
ans =
```

```
2
```

If the value of k may change in the future, one option is to make function depedent on two variables

```
F2 = Function(@(k,x) k*x)
```

```
Function: @(k,x)k*x
```

Evaluation of this function requires two arguments, i.e.

```
feval(F2.Handle,2,1)
```

```
ans =
```

```
2
```

Another option is to store the parameter k inside the `Data` property.
Let's create empty function with the initial value $k = 2$

```
F3 = Function([],struct('k',2))
```

```
Empty Function
```

Now we can assign the function handle with the reference to the parameter k stored inside `F3` object, i.e.

```
F3.setHandle(@(x) F3.Data.k*x)
```

Function: @(x)F3.Data.k*x

If we change the value of the parameter k inside the object

```
F3.Data.k = 3;
```

this new value will be taken for evaluation.

```
feval(F3.Handle,1)
```

ans =

3

Example 2

Construct anonymous function $f(x) = a * sin(b * x)$ with parameters a and b.
Firstly, construct empty `Function` object and store the parameters in the `Data` property.

```
F = Function([],struct('a',10.4,'b',-0.56))
```

Empty Function

Secondly, set the handle pointing to the parameters stored inside `Data` property

```
F.setHandle(@(x)F.Data.a*sin(F.Data.b*x))
```

Function: @(x)F.Data.a*sin(F.Data.b*x)

The values of the parameters a, b can be modified after construction of the object. The `Function` object will then take the new values of the parameters.

```
F.Data.a = 12.78; F.Data.b = -0.93
```

Function: @(x)F.Data.a*sin(F.Data.b*x)

Example 3

For functions arrays you must correctly refer to the appropriate index of the function data
Allocate an array of `Function` objects

```
F(2,1) = Function
```

Array of 2 Functions.

Assign the data

```
F(1).Data = struct('a',-1,'b',2);
```

```
F(1).setHandle(@(x)F(1).Data.a*sin(F(1).Data.b*x));
```

```
F(2).Data = struct('a',-3,'b',4);
```

```
F(2).setHandle(@(x)F(2).Data.a*sin(F(2).Data.b*x));
```

Example 4

Assignment of multiple function handles is possible as well.
Allocate the array of 2 `Function` objects.

```
F(2) = Function
```

Array of 2 Functions.

Assign two function handles must be done in a cell

```
F.setHandle({@(x)x,@(x)x^2})
```

Array of 2 Functions.

SEE ALSO

AffFunction, QuadFunction, setHandle

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ISEMPTY

Test if the object of the `Function` class contains a function handle.

SYNTAX

`isempty(F)`

`F.isEmptyFunction`

DESCRIPTION

The objects of `Function` class is considered as empty when it does not have a function handle associated to it, i.e. the property `Handle` is empty.

INPUT

  F   `Function` object.
      Class: `Function`

SEE ALSO

Function, setHandle

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

HORZCAT

Horizontal concatenation for `Function` class.

SYNTAX

```
F = horzcat(F1,F2)

F = [F1,F2]

F = [F1,F2,F3]
```

DESCRIPTION

The objects of `Function` class are concatenated into arrays only, no matrix concatenation is allowed. Note that only the objects of the same class can be concatenated into the same array.

INPUT

  F1   `Function` object.
       Class: `Function`

  F2   `Function` object.
       Class: `Function`

OUTPUT

  F   An array of `Function` objects.
       Class: `Function`

SEE ALSO

Function, vertcat, disp

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

353

FUNCTION

Function representation for MPT

SYNTAX

```
F = Function(@fun, data)

F = Function(@fun)

F = Function([],struct('p',2))
```

DESCRIPTION

The `Function` class represents functions handles that are associated to sets in MPT. The class combines the `function_handle` with changeable parameters which allows specifications of very general functions. When constructing the `Function` object, the correctness of the given function is not tested. It is up to the user to provide correct mapping, or test it via `feval` method for `ConvexSet` class. Domain of the function is specified by the set it is associated to. If the point lies outside of the domain, an error is thrown. The range of the function is not known at the time of the construction of the `Function` object. It can be determined after evaluation of the related function for given point. User can associate any data with the function, including parameters of the function, under the `Data` field. These data can be modified anytime after the construction of the object.

INPUT

Handle   A `function_handle` that represents the function. It can be an anonymous function, e.g. $f(x) = x^3$ that corresponds to syntax $@(x)x^3$ or it can be a link to another file that evaluates the expression, i.e. `@file`. Both of the expressions are fine as long as the given handle can be evaluated at the time of the construction. If more arguments (or parameters) are present in the function, the arguments are separated with a comma, e.g. $f(x, y, z) = x(y - z)^2$ corresponds to $@(x, y, z)x * (y - z)^2$. For more info about constructing handles see help for `function_handle` class.
Class: `function_handle`

Data   Any data related to the function. It can be function parameters, variable bounds, domain, range, measured data, symbolic representation, etc. The data can be changed after construction of the `Function` object.

OUTPUT

F   `Function` object.
Class: `Function`

EXAMPLE(s)

Example 1

Construct empty `Function`

`F=Function`

`Empty Function`

Construct an 5x1 array of empty `Function` objects

`F(5,1) = Function`

`Array of 5 Functions.`

Example 2

Construct anonymous function $f(x, y) = (x - y)/(x^2 + y^2)^0.5$

`F = Function(@(x,y) (x-y)/(x^2+y^2)^0.5)`

`Function: @(x,y)(x-y)/(x^2+y^2)^0.5`

Example 3

Construct anonymous function $f(x) = a * sin(b * x)$ with parameters a and b.
Firstly, construct empty `Function` object and put the parameter values inside the `Data` field

`F = Function([],struct('a',10.4,'b',-0.56))`

`Empty Function`

Secondly, set the handle pointing to the parameters stored inside `Data` property

`F.setHandle(@(x) F.Data.a*sin(F.Data.b*x))`

`Function: @(x)F.Data.a*sin(F.Data.b*x)`

The values of the parameters a, b can be modified after construction of the object. The `Function` object will then take the new values of the parameters.

`F.Data.a = 12.78; F.Data.b = -0.93`

`Function: @(x)F.Data.a*sin(F.Data.b*x)`

Example 4

For functions arrays you must correctly refer to the appropriate index of the function data
Allocate an array of `Function` objects

```
F(2,1) = Function
```

`Array of 2 Functions.`

Assign the data

```
F(1).Data = struct('a',-1,'b',2);
```

```
F(1).setHandle(@(x) F(1).Data.a*sin(F(1).Data.b*x));
```

```
F(2).Data = struct('a',-3,'b',4);
```

```
F(2).setHandle(@(x) F(2).Data.a*sin(F(2).Data.b*x));
```

SEE ALSO

`AffFunction`, `QuadFunction`, `setHandle`

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ONENORMFUNCTION

Class representing 1-norm function.

SYNTAX

```
f = OneNormFunction(Q)
```

DESCRIPTION

The object for representing 1-norm function given as $f = \|Qx\|_1$. The function is given as a sum of absolute values of the product $y = Qx$.

$$f = \sum_{i=1}^{n} |y_i|$$

where n is the dimension of the vector y. The weight `Q` does not need to be square. Function value is always scalar.

INPUT

Q   Weighing matrix where the number of columns determines the dimension of the vector argument.
     Class: `double`

OUTPUT

f   The `OneNormFunction` object.
     Class: `OneNormFunction`

EXAMPLE(s)

Example 1

Construct 1-norm function $f = |5x_1| + |-2x_2|$.

```
f = OneNormFunction(diag([5,-2]))
```

```
1-norm function in R^2
```

Evaluate the function in the point [-1;2].

```
f.feval([-1;2])
```

```
ans =

9
```

SEE ALSO

InfNormFunction, AffFunction, QuadFunction

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

DISPLAY

Overload display for `AffFunction` class.

SYNTAX

`display(F)`

`F.display`

DESCRIPTION

Default display for `AffFunction` class.

INPUT

   F   `AffFunction` object.
       Class: `Function`

SEE ALSO

`Function`, `AffFunction`, `QuadFunction`

AUTHOR(s)

 © **2010-2013** Martin Herceg: ETH Zurich
       mailto:herceg@control.ee.ethz.ch

AFFFUNCTION

Representation of affine functions in the form F*x + g

SYNTAX

```
L = AffFunction(F,g)

L = AffFunction(F)

L = AffFunction(F,g,Data)
```

DESCRIPTION

The `AffFunction` class represents affine functions of the form $f(x) = F * x + g$ where F is a real matrix and g is a real column vector. Dimensions of F and g must coincide such that the output is a vector or scalar.

INPUT

F   Real matrix representing the coefficients in the linear
        term F in $f(x) = F * x + g$.
        Class: `double`

g   Real vector representing the affine terms g in $f(x) = F * x + g$.
        Class: `double`

Data   Any data related to the function.

OUTPUT

L   `AffFunction` object.

EXAMPLE(s)

Example 1

Construct affine function $l(x) = 3x + 1$

```
L = AffFunction(3,1)
```

Affine Function: R^1 ->R^1

Construct linear function $k(x) = 2\pi x$

```
k = AffFunction(2*pi)
```

Affine Function: R^1 ->R^1

360

Example 2

Construct vectorized affine function $f(x) = \begin{pmatrix} x_1 + 2x_2 + 1 \\ 3x_1 + 4x_2 - 1 \end{pmatrix}$ with respect to vector x with two elements

```
F = AffFunction([1 2;3 4],[1; -1])
```

Affine Function: R^2 ->R^2

Example 3

Construct affine function $y = Fx + g$ where F and g are regression coefficients from the data x and y.
The data we want to store

```
data.x = 0:0.01:0.5;
```

```
data.y = sin(data.x);
```

```
data.file= 'DSCa001';
```

Compute the regression coefficients and store them in `h`

```
h = polyfit(data.x,data.y,1);
```

We can store the data from which the function was obtained under `Data` property

```
A=AffFunction(h(1),h(2),data)
```

Affine Function: R^1 ->R^1

SEE ALSO

Function, QuadFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

361

NORMFUNCTION

Class representing 1- or infinity-norm function.

SYNTAX

```
f = NormFunction(flag)
```

```
f = NormFunction(flag, Q)
```

DESCRIPTION

The common object for representing 1- and infinity- norm functions given as $f = \|x\|_p$ where $p \in \{1, \infty\}$. The one norm is given as a sum of absolute values

$$f_1 = \sum_{i=1}^{n} |x_i|$$

and the infinity norm is given as the maximum over the absolute values

$$f_\infty = \max\left(|x_1|, \ldots, |x_n|\right)$$

where n is the dimension of the vector x. If the weighing matrix `Q` is provided, then the product $f = \|Qx\|_p$ is considered. The weight `Q` does not need to be square. Function value is always scalar.

2-norms are not supported because they are neither quadratic, nor piecewise linear.

Do not use these objects in the user interface. Use `OneNormFunction` and `InfNormFunction` objects instead.

INPUT

1   Flag indicating the type of the norm. It can be either `1` or `Inf`.
    Class: `double`
    Allowed values: `1`
                      `Inf`
Q   Weighing matrix where the number of columns determines the dimension of the vector argument.
    Class: `double`

OUTPUT

f   The `NormFunction` object.
    Class: `NormFunction`

EXAMPLE(s)

Example 1

Construct 1-norm function $f_1 = |2x_1| + |-3x_2|$.

```
f1=NormFunction(1,diag([2,-3]))
```

```
1-norm function in R^2
```

Evaluate the function in the point [2;1].

```
f1.feval([2;1])
```

```
ans =

7
```

Construct infinity-norm function $f_\infty = \max(|2x_1|, |-3x_2|)$.

```
finf=NormFunction(Inf,diag([2,-3]))
```

```
Inf-norm function in R^2
```

Evaluate the function in the point [2;1].

```
finf.feval([2;1])
```

```
ans =

4
```

SEE ALSO

OneNormFunction, InfNormFunction, AffFunction, QuadFunction

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

QUADFUNCTION

Representation of quadratic functions in the form x'*H*x + F*x + g

SYNTAX

```
Q = QuadFunction(H,F,g)

Q = QuadFunction(H,F)

Q = QuadFunction(H)

Q = QuadFunction(H,F,g,Data)
```

DESCRIPTION

The `QuadFunction` class represents quadratic functions of the form $f(x) = x' * H * x + F * x + g$ where H is a real matrix, F is a real matrix and g is a real column vector. Dimensions of H, F and g must coincide such that the output is a scalar.

INPUT

H	Real matrix representing the coefficients in the quadratic term H in $f(x) = x' * H * x + F * x + g$. Class: `double`
F	Real matrix representing the coefficients in the linear term F in $f(x) = F * x + g$. Class: `double`
g	Real vector representing the affine terms g in $f(x) = F * x + g$. Class: `double`
Data	Any data related to the function.

OUTPUT

Q	`QuadFunction` object.

EXAMPLE(s)

Example 1

Construct a quadratic function $q_1(x) = x^2 + 1$

```
q1 = QuadFunction(1,0,1)
```

Quadratic Function: R^1 ->R^1

Construct a quadratic function $q_2(x) = 0.5x^2 + 3x - 1$

```
q2 = QuadFunction(0.5,3,-1)
```

```
Quadratic Function: R^1 ->R^1
```

Example 2

Construct a quadratic function $f(x) = x' * H * x + F * x + g$ where H, F, g will be randomly generated and x is a vector in dimension 2.

```
f = QuadFunction(randn(2),randn(1,2),randn(1))
```

```
Quadratic Function: R^2 ->R^1
```

Example 3

Construct quadratic function $x' * H * x + Fx + g$ where H, F and g are regression coefficients from the data x and y.
The data we want to store

```
data.x = 0:0.01:0.5;
```

```
data.y = cos(data.x);
```

```
data.file= 'DSCa002';
```

Compute the regression coefficients and store them in `h`

```
h = polyfit(data.x,data.y,2);
```

We can store the data from which the function was obtained under `Data` property

```
QuadFunction(h(1),h(2),h(3),data)
```

```
Quadratic Function: R^1 ->R^1
```

SEE ALSO

`Function`, `AffFunction`

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

DISPLAY

Overload display for `QuadFunction` class.

SYNTAX

`display(F)`

`F.display`

DESCRIPTION

Default display for `QuadFunction` class.

INPUT

  F `QuadFunction` object.
      Class: `Function`

SEE ALSO

Function, AffFunction, QuadFunction

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_CALL_CPLEX

A gateway function to CPLEX solver (without errorchecks)

SYNTAX

```
R = mpt_call_cplex(S)
```

DESCRIPTION

The function implements call to CPLEX solver based on formulation from `Opt` class. QP, LP, MILP and MIQP problems are supported.

It is assumed that QP/LP/MIQP/MILP entering this function (for LP/MILP $H = 0$) is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & lb \leq x \leq ub \\
& Ax \leq b \\
& A_e x = b_e \\
& x \in \{C, I, B, N, S\}
\end{aligned}
$$

where the set $\{C, I, B, N, S\}$ represents

- C - continuous variables, $x \in (-\infty, \infty)$

- I - integer variables $x \in (\ldots, -1, 0, 1, \ldots)$

- B - binary variables $x \in \{0, 1\}$

- N - semi-integer variables (possibly bounded above) $x \in [0, 1, \overline{x} \leq \infty)$

- S - semi-continuous variables (possibly bounded above) $x \in [0, \overline{x} \leq \infty)$

which is given by strings in `vartype` field. CPLEX accepts this format directly.

INPUT

S	structure of the Opt class Class: `struct`
S.H	Quadratic part of the objective function. Class: `double` Default: `[]`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \leq b$.

Class: `double`

`S.Ae`	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
`S.be`	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
`S.lb`	Lower bound for the variables $x \geq \text{lb}$. Class: `double` Default: `[]`
`S.ub`	Upper bound for the variables $x \leq \text{ub}$. Class: `double` Default: `[]`
`S.n`	Problem dimension (number of variables). Class: `double`
`S.m`	Number of inequalities in $Ax \leq b$. Class: `double`
`S.me`	Number of equalities in $A_e x = b_e$. Class: `double`
`S.problem_type`	A string specifying the problem to be solved. Class: `char`
`S.vartype`	A string specifying the type of variable. Supported characters are C (continuous), I (integer), B (binary), N (semi-integer), S (semi-continuous). Example: First variable from three is binary, the rest is continuous: `S.vartype='BCC';` Class: `char`
`S.test`	Call (false) or not to call (true) MPT global settings. Class: `logical` Default: `false`

OUTPUT

`R`	result structure Class: `struct`
`R.xopt`	Optimal solution.

Class: `double`

R.obj   Optimal objective value.
Class: `double`

R.lambda   Lagrangian multipliers.
Class: `double`

R.exitflag   An integer value that informs if the result was feasible
(1), or otherwise (different from 1).
Class: `double`

R.how   A string that informs if the result was feasible ('ok'), or
if any problem appeared through optimization.
Class: `char`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

369

MPT_SOLVEMP

A gateway function to solve parametric optimization problems (without errorchecks)

SYNTAX

```
R = mpt_solvemp(S)
```

DESCRIPTION

The main routine for fast calls to parametric optimization solvers. In fact, it is a subroutine of `Opt` as a part of `solve` method. The `Opt` class serves as general wrapper for preprocessing the data involved in optimization, including necessary error checks. Once the data are valid, then are passed to `mpt_solvemp` function that calls the appropriate parametric solver.
It is assumed that MPLP/MPQP entering this function are of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + F\theta + f^T x \\
\text{s.t.} \quad & Ax \le b + B\theta \\
& A_e x = b_e + E \\
& lb \le x \le ub \\
& A_\theta \theta \le b_\theta
\end{aligned}
$$

where the matrices H, F, A, A_e A_θ, B, E, and vectors f, b, b_e, b_θ, `lb`, `ub` are the problem data. Vector x represents decision variables and θ are parameters. The PLCP must be given as:

$$
\begin{aligned}
w - Mz &= q + Q\theta \\
w &\ge 0 \\
z &\ge 0 \\
w(\theta)^T z(\theta) &= 0 \\
A_\theta \theta &\le b_\theta
\end{aligned}
$$

where the matrices M, Q, A_θ, and vectors q,b_θ are the problem data, then z, w are the decision variables and θ are the parameters.
The routine `mpt_solve` processes data from any of the above optimization problems by passing it to the appropriated solver. Following solvers are supported:

- **PLCP** is the default solver, called automatically when invoking parametric optimization. It can solve MPQP/MPLP and PLCP problems.

- **MPLP** is the solver from MPT2.6 which is still supported for development purposes.

- **MPQP** is the solver from MPT2.6 which is still supported for development purposes.

Note that this function must contain all properties of `Opt` class that have been properly validated in order to perform a correct call to given parametric solver. It is recommended to use `Opt.solve` method instead.

INPUT

S   Object of the Opt class
                           Class: `Opt`

S.H   Quadratic part of the objective function.
                           Class: `double`
                           Default: `[]`

S.f   Linear part of the objective function.
                           Class: `double`

S.pF   Linear part of the objective function for parameters.
                           Class: `double`
                           Default: `[]`

S.A   Linear part of the inequality constraints $Ax \leq b + B\theta$.
                           Class: `double`

S.b   Right hand side of the inequality constraints $Ax \leq b + B\theta$.
                           Class: `double`

S.pB   Right hand side of the inequality constraints for parameters $Ax \leq b + B\theta$.
                           Class: `double`

S.Ae   Linear part of the equality constraints $A_e x = b_e + E\theta$.
                           Class: `double`
                           Default: `[]`

S.be   Right hand side of the equality constraints $A_e x = b_e + E\theta$.
                           Class: `double`
                           Default: `[]`

S.pE   Right hand side of the equality constraints for parameters $A_e x = b_e + E\theta$.
                           Class: `double`
                           Default: `[]`

S.lb   Lower bound for the decision variables $x \geq$ lb.
                           Class: `double`
                           Default: `[]`

S.ub   Upper bound for the decision variables $x \leq$ ub.
                           Class: `double`
                           Default: `[]`

S.Ath   Linear part of the inequality constraints $A_\theta \theta \leq b_\theta$.
                           Class: `double`

Default: []

S.bth   Right hand side of the inequality constraints $A_\theta \theta \leq b_\theta$.
                      Class: `double`
                      Default: []

S.M   Linear matrix involved in LCP.
                      Class: `double`
                      Default: []

S.q   Right hand side vector involved in LCP.
                      Class: `double`
                      Default: []

S.Q   Linear matrix involved in parametric formulation of
                      LCP.
                      Class: `double`
                      Default: []

S.n   Number of decision variables.
                      Class: `double`

S.d   Number of parameters.
                      Class: `double`

S.m   Number of inequalities in $Ax \leq b + B\theta$.
                      Class: `double`

S.me   Number of equalities in $A_e x = b_e + E\theta$.
                      Class: `double`

S.problem_type   A string specifying the problem to be solved
                      Class: `char`
                      Default: []

S.vartype   A string array reserved for MPMILP/MPMIQP.
                      Class: `char`
                      Default: ''

S.solver   S string specifying which solver should be called.
                      Class: `char`
                      Default: []

S.isParametric   Logical scalar indicating that the problem is parametric.
                      Class: `double or logical`
                      Default: 1

S.varOrder   Order of variables if the problem was processed by
                      YALMIP first.
                      Class: `double`

Default: []

S.Internal	Internal property of `Opt` class. Class: `struct` Default: `[]`
S.recover	Affine map for MPLP/MPQP problems after transformation to LCP. Class: `struct`
S.recover.uX	Matrix of the affine map $x = \text{uX} \begin{pmatrix} w \\ z \end{pmatrix} + \text{uTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in LCP $w(\theta), z(\theta) \mapsto x$ and in the original LP/QP. Class: `double` Default: `[]`
S.recover.uTh	Matrix of the affine map $x = \text{uX} \begin{pmatrix} w \\ z \end{pmatrix} + \text{uTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in LCP $w(\theta), z(\theta) \mapsto x$ and in the original LP/QP. Class: `double` Default: `[]`
S.recover.lambdaX	Matrix of the affine map $x = \text{lambdaX} \begin{pmatrix} w \\ z \end{pmatrix} + \text{lambdaTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in LCP $w(\theta), z(\theta) \mapsto \lambda$ and the Lagrangian multipliers in the original LP/QP. Class: `double` Default: `[]`
S.recover.lambdaTh	Matrix of the affine map $x = \text{lambdaX} \begin{pmatrix} w \\ z \end{pmatrix} + \text{lambdaTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in LCP $w(\theta), z(\theta) \mapsto \lambda$ and the Lagrangian multipliers in the original LP/QP. Class: `double` Default: `[]`
	Default: `[]`

OUTPUT

R	result structure Class: `struct`

R.xopt	Optimal solution with the associated functions for optimizer, multipliers and the objective value. Class: `PolyUnion`
R.exitflag	An integer value that informs if the result was feasible (1), or otherwise (different from 1) Class: `double`
R.how	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization Class: `char`
R.solveTime	Information about the time that elapsed during the computation in seconds. Class: `double`
R.stats	Further details from the parametric solver. Class: `struct`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_CALL_LCP

A gateway function to LCP solver (without errorchecks)

SYNTAX

```
R = mpt_call_lcp(S)
```

DESCRIPTION

The function implements calls to LCP solver based on the optimization problem to be solved. Supported problems are LCP, LP, and QP.

If the problem type is LCP, then LCP mex-function is called directly. Otherwise a transformation to LCP problem is performed. Assume that QP/LP is written in a form

$$
\begin{aligned}
\min \quad & \frac{1}{2} x^T H x + f^T x \\
\text{s.t.} \quad & A x \leq b \\
& A_e x = b_e
\end{aligned}
$$

We want to transform it to

$$
\begin{aligned}
\min \quad & \frac{1}{2} x^T H_f x + f_f^{\ T} x \\
\text{s.t.} \quad & A_f x \geq b_f \\
& x \geq 0
\end{aligned}
$$

which corresponds to the following LCP

$$
\begin{aligned}
w - M z &= q \\
w &\geq 0 \\
z &\geq 0 \\
w^T z &= 0
\end{aligned}
$$

where

$$
\begin{aligned}
M &= \begin{pmatrix} H_f & -A_f^{\ T} \\ A_f & 0 \end{pmatrix} \\
q &= \begin{pmatrix} f_f \\ -b_f \end{pmatrix}
\end{aligned}
$$

If LP or QP contains equality constraints, these are removed first. It is required that the system of linear equations $A_e x = b_e$ is consistent, i.e. no linearly dependent rows are found and the number of equalities is strictly less than number of variables. The principle is based on factorizing equality constraints $A_e x = b_e$ in basic x_{Bc} and non-basic variables x_{Nc}, i.e.

$$
A_e = \begin{pmatrix} A_{e,Bc} & A_{e,Nc} \end{pmatrix}
$$

which gives

$$A_{e,Bc}x_{Bc} + A_{e,Nc}x_{Nc} = b_e$$

where the index sets `Bc`, `Nc` denote the columns from which factored system is built. The factored submatrix $A_{e,Bc}$ must be invertible in order to express basic variables as a function of non-basic variables, i.e.

$$x_{Bc} = -A_{e,Bc}{}^{-1}A_{e,Nc}x_{Nc} + A_{e,Bc}{}^{-1}b_{e,Bc}$$

With the substitution

$$C = -A_{e,Bc}{}^{-1}A_{e,Nc}$$

and

$$D = A_{e,Bc}{}^{-1}b_{e,Bc}$$

the relation between basic and non-basic variables is simplified to

$$x_{Bc} = Cx_{Nc} + D \tag{35}$$

The above QP/LP problem $(23) - (25)$ can be expressed only in non-basic variables x_{Nc} as follows:

$$
\begin{aligned}
\min \quad & \frac{1}{2}x_{Nc}{}^T\tilde{H}x_{Nc} + \tilde{f}^Tx_{Nc} + \tilde{c} \\
\text{s.t.} \quad & \tilde{A}x_{Nc} \leq \tilde{b}
\end{aligned}
$$

where

$$
\begin{aligned}
\tilde{H} &= C^TH_{Bc,Bc}C + C^TH_{Bc,Nc} + H_{Nc,Bc}C + H_{Nc,Nc} \\
\tilde{f} &= D^TH_{Bc,Bc}C + D^TH_{Nc,Bc} + f_{Bc}{}^TC + f_{Nc}{}^T \\
\tilde{c} &= \frac{1}{2}D^TH_{Bc,Bc}D + f_{Bc}{}^TD \\
\tilde{A} &= A_{Bc}C + A_{Nc} \\
\tilde{b} &= b - A_{Bc}D
\end{aligned}
$$

Original solution to QP problem $(23) - (25)$ can be obtained via relation ().
Problem $(35) - (36)$ can be transformed to $(26) - (28)$ effectively when considering the rank of the matrix $\tilde{A}$. If the rank of matrix $\tilde{A}$ is less than the number of inequalities in (36) vector x_N can be expressed as a difference of two positive numbers, i.e.

$$
\begin{aligned}
x_{Nc} &= x_{Nc}{}^+ - x_{Nc}{}^- \\
x_{Nc}{}^+ &\geq 0 \\
x_{Nc}{}^- &\geq 0
\end{aligned}
$$

With the substitution

$$v = \begin{pmatrix} x_{Nc}{}^+ \\ x_{Nc}{}^- \end{pmatrix}$$

and putting back to $(35) - (36)$ we get

$$
\begin{aligned}
\min \quad & \frac{1}{2} v^T \begin{pmatrix} \tilde{H} & -\tilde{H} \\ -\tilde{H} & \tilde{H} \end{pmatrix} v + \begin{pmatrix} \tilde{f}^T & -\tilde{f}^T \end{pmatrix} v + \tilde{c} \\
\text{s.t.} \quad & \begin{pmatrix} -\tilde{A} & \tilde{A} \end{pmatrix} v \geq -\tilde{b} \\
& v \geq 0
\end{aligned}
$$

which is a form equivalent to $(26) - (28)$. If the rank of matrix $\tilde{A}$ is greater or equal than the number of inequalities in (36), we can factorize the matrix $\tilde{A}$ rowwise

$$
\tilde{A} = \begin{pmatrix} \tilde{A}_B \\ \tilde{A}_N \end{pmatrix}
$$

where B, N are index sets corresponding to rows from which submatrices are built. The factored system (36) can be written as

$$
\begin{aligned}
- \tilde{A}_B x_{Nc} & \geq -\tilde{b}_B \\
- \tilde{A}_N x_{Nc} & \geq -\tilde{b}_N
\end{aligned}
$$

where the matrix $\tilde{A}_B$ form by rows in the set B must be invertible. Using the substitution

$$
-\tilde{A}_B x_{Nc} = -\tilde{b}_B + y
$$

the system $(46)(47)$ can be rewritten in variable y

$$
\begin{aligned}
\min \quad & \frac{1}{2} y^T \left(\tilde{A}_B^{-T} \tilde{H} \tilde{A}_B^{-1} \right) y + \left(-\tilde{A}_B^{-T} \tilde{H} \tilde{A}_B^{-1} \tilde{b}_B - \tilde{A}_B^{-T} \tilde{f} \right) y + \tilde{c} \\
\text{s.t.} \quad & \tilde{A}_N \tilde{A}_B^{-1} y \geq \tilde{A}_N \tilde{A}_B^{-1} \tilde{b}_B - \tilde{b}_N \\
& y \geq 0
\end{aligned}
$$

which is equivalent to $(26) - (28)$.

INPUT

`S`	Structure of the Opt class. Class: `struct`
`S.H`	Quadratic part of the objective function. Class: `double` Default: `[]`
`S.f`	Linear part of the objective function. Class: `double`
`S.A`	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
`S.b`	Right hand side of the inequality constraints $Ax \leq b$.

377

Class: `double`

`S.Ae`	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
`S.be`	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
`S.lb`	Lower bound for the variables $x \geq$ lb. Class: `double` Default: `[]`
`S.ub`	Upper bound for the variables $x \leq$ ub. Class: `double` Default: `[]`
`S.M`	Data matrix for linear-complementarity problem $w - Mx = q$. Class: `double`
`S.q`	Right hand side vector for linear-complementarity problem $w - Mx = q$. Class: `double`
`S.n`	Problem dimension (number of variables). Class: `double`
`S.m`	Number of inequalities in $Ax \leq b$. Class: `double`
`S.me`	Number of equalities in $A_e x = b_e$. Class: `double`
`S.problem_type`	A string specifying the problem to be solved (only LP, QP and LCP problems are allowed). Class: `char`

S.routine   An integer specifying which subroutine to use for fac-
              torization. For details, see help for LCP solver.

              - The number 0 indicates to use fast recursive fac-
                torization based on rank-1 updates from LUMOD
                package.

              - The number 1 indicates to use LU factorization
                based on DGESV routine of LAPACK.

              - The number 2 indicates to use QR factorization
                based on DGELS routine of LAPACK.

              Class: `double`

S.test   Call (false) or not to call (true) MPT global settings.
              Class: `logical`
              Default: `false`

OUTPUT

R   Result structure
              Class: `struct`

R.xopt   The optimal values for variable z.
              Class: `double`

R.lambda   The optimal values for variable w.
              Class: `double`

R.obj   If the LCP problem was created by conversion from
              LP/QP, this value represent the optimal cost of the ap-
              propriate LP/QP.
              Class: `double`

R.exitflag   An integer value that informs if the result was feasible
              (1), or otherwise (different from 1).
              Class: `double`

R.how   A string that informs if the result was feasible ('ok'), or
              if any problem appeared through optimization.
              Class: `char`

SEE ALSO

mpt_solve

379

LITERATURE

Stephen Boyd and Lieven Vandenberghe: Convex Optimization; Cambridge University Press
Richard W. Cottle, Jong-Shi Pang, Richard E. Stone: Linear complementarity problem,
Academic Press Inc. 1992

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

MPT_SOLVE

A gateway function to solve non-parametric optimization problems (without errorchecks)

SYNTAX

```
R = mpt_solve(S)
```

DESCRIPTION

The function is the main routine for fast calls for solving non-parametric optimization problems in MPT. In fact, it is a subroutine of `Opt` as a part of `solve` method. The `Opt` class serves as general wrapper for preprocessing the data involved in optimization, including necessary error checks. Once the data are valid, then are passed to `mpt_solve` function that calls the appropriate solver.

It is assumed that QP/LP/MIQP/MILP and entering this function (for LP/MILP $H = 0$) is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & \text{lb} \leq x \leq \text{ub} \\
& Ax \leq b \\
& A_e x = b_e \\
& x \in \{C, I, B, N, S\}
\end{aligned}
$$

where the set $\{C, I, B, N, S\}$ represents

- C - continuous variables, $x \in (-\infty, \infty)$

- I - integer variables $x \in (\ldots, -1, 0, 1, \ldots)$

- B - binary variables $x \in \{0, 1\}$

- N - semi-integer variables (possibly bounded above) $x \in [0, 1, \overline{x} \leq \infty)$

- S - semi-continuous variables (possibly bounded above) $x \in [0, \overline{x} \leq \infty)$

which is given by strings in `vartype` field. The matrices H, A, A_e, and vectors f,b, b_e, `lb`, `ub` are the problem data, then x are the decision variables.

The LCP must be given as:

$$
\begin{aligned}
w - Mz &= q \\
w &\geq 0 \\
z &\geq 0 \\
w^T z &= 0
\end{aligned}
$$

where the matrices M, Q, and vectors q are the problem data, z, w are the decision variables to be determined.

The function `mpt_solve` processes the problem data and passes it to the appropriate solver. The particular solver can be specified by providing `solver` name or it is selected automatically from the list of available solvers. Based on the solver name, the appropriated function is called:

- **CDD** solver is installed by default, and can be called via `mpt_call_cdd`. Solves LP problems.

- **CLP** solver is installed by default, and can be called via `mpt_call_clp`. Solves LP/QP problems.

- **CPLEX** is a commercial solver and must be installed additionally. It can be called via `mpt_call_cplex`. CPLEX solves LP/QP/MILP/MIQP problems.

- **GLPK** solver is installed by default, and can be called via `mpt_call_glpk`. It solves LP/QP/MILP problems.

- **GUROBI** is a commercial solver and must be installed additionally. It can be called via `mpt_call_gurobi`. It solves LP/QP/MILP/MIQP problems.

- **LCP** is a default solver, and can be called via `mpt_call_lcp`. It can be used to solve LP/QP and LCP problems.

- **LINPROG** is Matlab LP solver, and can be called via `mpt_call_linprog`.

- **QUADPROG** is Matlab QP solver, and can be called via `mpt_call_quadprog`.

- **NAG** is a commercial solver and must be installed additionally. It can be called via `mpt_call_nag`. It solves LP/QP problems.

- **QPC** is LP/QP solver that need to be installed additionally. It can be called via `mpt_call_qpc`.

- **QPOASES** is LP/QP solver that is installed by default. It can be called via `mpt_call_qpoases`.

- **QPSPLINE** is a QP solver for strictly convex problems and can be called via `mpt_call_qpspline`.

- **SEDUMI** is a semidefinite solver for general convex problems and can be called via `mpt_call_sedumi`.

INPUT

S	structure of the Opt class Class: `struct`
S.H	quadratic part of the objective function Class: `double` Default: `[]`
S.f	linear part of the objective function Class: `double`
S.A	linear part of the inequality constraints $Ax \leq b$ Class: `double`
S.b	right hand side of the inequality constraints $Ax \leq b$ Class: `double`
S.Ae	linear part of the equality constraints $A_e x = b_e$

Class: `double`
Default: `[]`

S.be   right hand side of the equality constraints $A_e x = b_e$
Class: `double`
Default: `[]`

S.lb   lower bound for the variables $x \geq \text{lb}$
Class: `double`
Default: `[]`

S.ub   upper bound for the variables $x \leq \text{ub}$
Class: `double`
Default: `[]`

S.M   Positive semi-definite matrix defining LCP.
Class: `double`
Default: `[]`

S.q   Right hand side vector defining LCP.
Class: `double`
Default: `[]`

S.n   problem dimension (number of variables)
Class: `double`

S.m   number of inequalities in $Ax \leq b$
Class: `double`

S.me   number of equalities in $A_e x = b_e$
Class: `double`

S.problem_type   a string specifying the problem to be solved
Class: `char`

S.vartype   A string specifying the type of variable. Supported characters are C (continuous), I (integer), B (binary), N (semi-integer), S (semi-continuous). Example: First variable from three is binary, the rest is continuous: `S.vartype='BCC';`

Class: `char`

S.solver   S string specifying which solver should be called.
Class: `char`

S.test   Call (false) or not to call (true) MPT global settings
Class: `logical`
Default: `false`

OUTPUT

R   result structure
                 Class: `struct`

R.xopt   Optimal solution
                 Class: `double`

R.obj   Objective value
                 Class: `double`

R.lambda   Lagrangian multipliers
                 Class: `double`

R.exitflag   An integer value that informs if the result was feasible
                 (1), or otherwise (different from 1)
                 Class: `double`

R.how   A string that informs if the result was feasible ('ok'), or
                 if any problem appeared through optimization
                 Class: `char`

EXAMPLE(s)

Example 1

Solve an LP stored in file `sc50b`.
Load the data

```
load sc50b
```

Solve the problem using LINPROG

```
[x,fval,exitflag,output] = linprog(f,A,b,Aeq,beq,lb,[],[]);
```

```
Optimization terminated.
```

It is the same as solving using `mpt solve`.
Put the data to a structure `S`.

```
S.f=f; S.A=A; S.b=b; S.Ae=Aeq; S.be=beq; S.lb=lb;
```

Solve with the default solver

```
R1 = mpt_solve(S)
```

```
R1 =

xopt: [48x1 double]
lambda: [1x1 struct]
```

```
obj: -69.9999999999999
how: 'ok'
exitflag: 1
```

Solve with LCP solver

```
S.solver = 'LCP';
```

```
R2 = mpt_solve(S)
```

```
R2 =
```

```
xopt: [48x1 double]
lambda: [1x1 struct]
obj: -69.9999999999999
how: 'ok'
exitflag: 1
```

You can see that the results are the same.

```
norm(x-R1.xopt)
```

```
ans =
```

```
5.71049616954122e-11
```

```
norm(x-R2.xopt)
```

```
ans =
```

```
5.71049616954122e-11
```

SEE ALSO

Opt

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_PLCP

Parametric linear complementarity solver (PLCP) (without errorchecks)

SYNTAX

`R = mpt_plcp(S)`

DESCRIPTION

Implementation of the lexicographic PLCP solver. The PLCP is given as:

$$w - Mz = q + Q\theta$$
$$w \geq 0$$
$$z \geq 0$$
$$w(\theta)^T z(\theta) = 0$$
$$A_\theta \theta \leq b_\theta$$

where the matrices M, Q, A_θ, and vectors q, b_θ are the problem data, then z, w are the decision variables and θ are the parameters.
Structure of the algorithm:

1. **Initialisation phase:** For any feasible starting point θ_f solve non-parametric LCP to get feasible basis. The basis is used to determine the initial region P_0 with the corresponding optimal pair w and z.

2. **Exploration phase:** For each facet of the initial region P_0 compute its neighbors P_i by performing lex-pivot steps in the PLCP subproblem. Compute the graph based on the found neighbors. Store basis of every region to a corresponding hash-table.

3. **Termination phase:** Verify the graph of the solution, evaluate the statistics.

The algorithm uses variable step approach for exploration of the parameter space by default. The fixed step approach can be turned on via option `fixed_step`. Parameter exploration is based on a graph search: depth-first search (BFS) and breadth-first search (BFS) methods are considered, the default is BFS.
When the computer runs in a parallel mode, the exploration phase run in a parallel for-loop automatically, which can potentially increases the computational speed.
Any change in default options must be done using `mptopt` class. Following options can be modified:

- **bfs** - Logical value that determines if to use BFS for exploration of the parameter space (default=1).

- **dfs** - Logical value that determines if to use DFS for exploration of the parameter space (default=0).

- **debug** - Integer value that determines the debugging level (default=0).

- **maxlayers** - For BFS-type of exploration, this value limits the number of "layers" starting from the initial region P_0. The first layer is defined as the collection of regions that are adjacent to P_0, the second layer is given as the new found neighbors to all regions in the first layer etc. The default value is `Inf`.

- **maxregions** - The maximal number of regions to be produced by PLCP algorithm. This option is useful when solving large PLCP problems when there are memory problems. The default value is `Inf`.

- **QRfactor** - Logical value that select the type of factorization to use for pivoting. If true, then recursive QR factorization is used instead of direct sparse LU factorization by default. The default value is 0.

- **checkoverlaps** - Logical value that launches overlap checking if true. Very time consuming operation, therefore the default value is 0.

- **rescue** - If the variable step approach fails to find all the neighbors, this logical statement indicates if the use the fixes-step approach as a backup. If true, then the fixed-step approach is run whenever variable step approach does not find overlaps. The disadvantage of fixed step is, however, overlaps can be produced, specially for degenerate cases. The default value is 0.

- **maxsteps** - If the fixed step approach is turned on, this value limits the number of steps to be performed to find neighbor. The step size is given by the `region_tol`. The default value is 200.

- **maxpivots** - The maximum limit on the lex-pivot steps to be performed when finding a neighbor. Typically, it suffices 1-2 pivots to find a neighbor. If the problem is very degenerate, or badly posed, or due to numerical problems involved in factorization, the pivot steps are repeated up to 100-times, which is the default value.

- **cacheregions** - This flag causes that regions that have been discovered are remembered and used for faster exploration of the parameter space. The default value is 1.

Note that to properly preprocess data to PLCP, use `Opt` class whenever possible. This will avoid unnecessary numerical problems caused by improper formulation of the problem.

INPUT

S	Structure of the `Opt` class. Class: `struct`
S.M	Data matrix for linear-complementarity problem $w - Mx = q$. Class: `double`
S.q	Right hand side vector for linear-complementarity problem $w - Mx = q$. Class: `double`
S.Ath	Linear part of the inequality constraints $A_\theta \theta \leq b_\theta$. Class: `double`

Default: `[]`

S.bth	Right hand side of the inequality constraints $A_\theta \theta \le b_\theta$. Class: `double` Default: `[]`
S.recover	Affine map for MPLP/MPQP problems after transformation to LCP. Class: `struct`
S.recover.uX	Matrix of the affine map $x = \mathrm{uX} \begin{pmatrix} w \\ z \end{pmatrix} + \mathrm{uTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in LCP $w(\theta), z(\theta) \mapsto x$ and in the original LP/QP. Class: `double` Default: `[]`
S.recover.uTh	Matrix of the affine map $x = \mathrm{uX} \begin{pmatrix} w \\ z \end{pmatrix} + \mathrm{uTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in LCP $w(\theta), z(\theta) \mapsto x$ and in the original LP/QP. Class: `double` Default: `[]`
S.recover.lambdaX	Matrix of the affine map $x = \mathrm{lambdaX} \begin{pmatrix} w \\ z \end{pmatrix} + \mathrm{lambdaTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in LCP $w(\theta), z(\theta) \mapsto \lambda$ and the Lagrangian multipliers in the original LP/QP. Class: `double` Default: `[]`
S.recover.lambdaTh	Matrix of the affine map $x = \mathrm{lambdaX} \begin{pmatrix} w \\ z \end{pmatrix} + \mathrm{lambdaTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in LCP $w(\theta), z(\theta) \mapsto \lambda$ and the Lagrangian multipliers in the original LP/QP. Class: `double` Default: `[]` Default: `[]`
S.Internal	Internal data that came from transformation from LP/QP to LCP. Class: `struct`
S.Internal.H	Quadratic part of the objective function. Class: `double` Default: `[]`

S.Internal.f	Linear part of the objective function. Class: `double`
S.Internal.pF	Linear part of the objective function for parameters. Class: `double` Default: `[]`
	Default: `[]`

OUTPUT

R	Result structure Class: `struct`
R.xopt	Partition of the polyhedra with the associated function values for z and w variables. Class: `PolyUnion`
R.exitflag	An integer value that informs if the result was feasible (1), or otherwise (different from 1). Class: `double`
R.how	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization. Class: `char`
R.solveTime	How long did the computation take in seconds. Class: `double`
R.stats	Statistical information about the computation: the total number of pivots performed, the total number of facets traversed. Class: `struct`
R.stats.pivs	The total number of pivots performed. Class: `double`
R.stats.facetsTraversed	The total number of facets that have been traversed. Class: `double`

SEE ALSO

mpt_solvemp, lcp

LITERATURE

Richard W. Cottle, Jong-Shi Pang, Richard E. Stone: Linear complementarity problem, Academic Press Inc. 1992

C.N. Jones, M. Morari: Multiparametric Linear Complementarity Problems, IEEE Conference on Decision and Control, 2006

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_CALL_QPC

A gateway function to QPC solver (without errorchecks)

SYNTAX

```
R = mpt_call_qpc(S)
```

DESCRIPTION

The function implements call to QPC solver based on formulation from `Opt` class. Only QP and LP problems are supported.

It is assumed that QP/LP entering this function (for LP $H = 0$) is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2} x^T H x + f^T x \\
\text{s.t.} \quad & \text{lb} \leq x \leq \text{ub} \\
& A x \leq b \\
& A_e x = b_e
\end{aligned}
$$

which is passed to QPC solver directly. Sparse inputs are converted to full if needed.

QPC offers two types of algorithms to solve QP/LP. For an interior point method specify in the field "solver" a string "qpip". Otherwise active set method is chosen by default.

INPUT

S	Structure of the `Opt` class. Class: `struct`
S.H	Quadratic part of the objective function. Class: `double` Default: `[]`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \leq b$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`

S.lb   Lower bound for the variables $x \geq$ lb.
                Class: `double`
                Default: `[]`

S.ub   Upper bound for the variables $x \leq$ ub.
                Class: `double`
                Default: `[]`

S.n   Problem dimension (number of variables).
                Class: `double`

S.m   Number of inequalities in $Ax \leq b$.
                Class: `double`

S.me   Number of equalities in $A_e x = b_e$.
                Class: `double`

S.problem_type A string specifying the problem to be solved.
                Class: `char`

S.test   Call (false) or not to call (true) MPT global settings.
                Class: `logical`
                Default: `false`

S.solver   Specific routine to be called of QPC. To call interior
                point method, specify "qpip". To call active set method,
                specify "qpas" or leave empty.
                Class: `char`

OUTPUT

R   result structure
                Class: `struct`

R.xopt   Optimal solution.
                Class: `double`

R.obj   Optimal objective value.
                Class: `double`

R.lambda   Lagrangian multipliers.
                Class: `double`

R.exitflag   An integer value that informs if the result was feasible
                (1), or otherwise (different from 1).
                Class: `double`

R.how   A string that informs if the result was feasible ('ok'), or
                if any problem appeared through optimization

Class: `char`

SEE ALSO

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_DETECT_SOLVERS

Searches for installed solvers on the path.

SYNTAX

```
s = mpt_detect_solvers
```

DESCRIPTION

Searches for installed solvers on the Matlab path. The following list of solvers are detected:

- CDD

- CLP

- CPLEX

- GLPK

- GUROBI

- LCP

- LINPROG

- NAG

- QPC

- QPOASES

- QPSPLINE

- QUADPROG

- SEDUMI

The found solvers are sorted according to a preference list that can be changed and according to the type of the optimization problem they solve. The search for solvers is only performed at the first time start of the toolbox, the solvers list is stored under global options.
To change the order of solvers, use `mptopt` option handler.

OUTPUT

s	Structure with the solvers list sorted according to preference. Class: `struct`
s.LP	List of available LP solvers. Class: `char`
s.QP	List of available QP solvers. Class: `char`
s.MILP	List of available MILP solvers.

Class: `char`

s.MIQP   List of available MIQP solvers.
                Class: `char`

s.LCP   List of available LCP solvers.
                Class: `char`

s.parametric   List of available parametric solvers.
                Class: `char`

SEE ALSO

mptopt, mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_CALL_SEDUMI

A gateway function to SEDUMI solver (without errorchecks)

SYNTAX

```
R = mpt_call_sedumi(S)
```

DESCRIPTION

The function implements calls to solve LP directly and QP via transformation to second order cone.
For LP in a form

$$\begin{aligned} \min \quad & f^T x \\ \text{s.t.} \quad & Ax \le b \\ & A_e x = b_e \end{aligned}$$

we need to get a following form accepted by SEDUMI

$$\begin{aligned} \min \quad & f_f{}^T x \\ \text{s.t.} \quad & A_f x \ge b_f \\ & x \ge 0 \end{aligned}$$

This can be achieved by introducing variables $x^+ \ge 0$, $x^- \ge 0$, and $y \ge 0$

$$\begin{aligned} x &= x^+ - x^- \\ y &= -Ax + b \end{aligned}$$

and putting them in one vector $z = [x^+, \ x^-, \ y]$. The LP to be solved by SEDUMI is formed by

$$\begin{aligned} f_f &= \begin{pmatrix} f & -f & 0 \end{pmatrix} \\ A_f &= \begin{pmatrix} -A & A & -I \\ A_e & -A_e & 0 \end{pmatrix} \\ b_f &= \begin{pmatrix} -b \\ b_e \end{pmatrix} \end{aligned}$$

and solved in z variables. For QP of the form

$$\begin{aligned} \min \quad & \frac{1}{2} x^T H x f^T x \\ \text{s.t.} \quad & Ax \le b \\ & A_e x = b_e \end{aligned}$$

we need to impose constraints a in quadratic cone $\mathcal{K}$ and to express the above problem as

$$
\begin{aligned}
\min \quad & c_n^T x \\
\text{s.t.} \quad & A_n x = b_n \\
& x \in \mathcal{K}
\end{aligned}
$$

Since QP is convex, we can express $(82) - (84)$ in an epigraph form

$$
\begin{aligned}
\min \quad & t \\
\text{s.t.} \quad & Ax \leq b \\
& A_e x = b_e \\
& \frac{1}{2} x^T H x + f^T x \leq t
\end{aligned}
$$

over quadratic constraints (91). From the literature for convex programming follows that the quadratic constraints (91) can be written as conic constraints

$$
\left\{ (x,t) \mid \left(x^T Q^T Q x + f^T x - t \leq 0 \right) = \left(\frac{1 - f^T x + t}{2} \leq \left\| \begin{array}{c} Qx \\ \frac{1+f^T x - t}{2} \end{array} \right\|_2 \right) \right\}
$$

where the matrix Q is a Cholesky factor of $0.5H = Q^T Q$. This equivalence allows us to rewrite the epigraph form of QP $(88) - (91)$ to a primal form $(85) - (87)$ accepted by SEDUMI. Equality and inequality constraints are treated the same way as in LP case, i.e. by introducing the new variables $x = x^+ - x^-$, $x^+ \geq 0$, $x^- \geq 0$, and $y = -Ax + b$, $y \geq 0$. Moreover, to express conic constraints we need two more variables $v \in \mathcal{R}^1$, $u \in \mathcal{R}^{n+1}$

$$
\begin{aligned}
v &= \frac{1 - f^T x + t}{2} \\
u &= \left(\begin{array}{c} Qx \\ \frac{1+f^T x - t}{2} \end{array} \right)
\end{aligned}
$$

Collecting all variables to one column vector $z = [t, \ x^+, \ x^-, \ y, \ v, \ u]$ the linear equality constraints (86) can be expressed in z variable as $A_n z = c_c$ where

$$
\begin{aligned}
A_n &= \begin{pmatrix} 0 & -A & A & -I & 0 & 0 \\ 0 & A_e & -A_e & 0 & 0 & 0 \\ 0 & Q & -Q & 0 & 0 & -I \\ \frac{-1}{2} & \frac{f^T}{2} & \frac{-f^T}{2} & 0 & 0 & -1 \\ \frac{1}{2} & \frac{-f^T}{2} & \frac{f^T}{2} & 0 & -1 & 0 \end{pmatrix} \\
b_n &= \begin{pmatrix} -b \\ b_e \\ 0 \\ \frac{-1}{2} \\ \frac{-1}{2} \end{pmatrix}
\end{aligned}
$$

The objective function in (85) is composed as $c_n^T = [1, \ 0, \ 0, \ 0, \ 0, \ 0]$. The individual constraints $\mathcal{K}$ in (87) are given in z vector as follows: t is free variable, x^+, x^-, y are restricted to be nonnegative and u, v form the conic constraint $v \geq \|u\|_2$.

INPUT

S	Structure of the `Opt` class Class: `struct`
S.H	Quadratic part of the objective function. Class: `double` Default: `[]`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \leq b$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.lb	Lower bound for the variables $x \geq$ lb. Class: `double` Default: `[]`
S.ub	Upper bound for the variables $x \leq$ ub. Class: `double` Default: `[]`
S.n	Problem dimension (number of variables). Class: `double`
S.m	Number of inequalities in $Ax \leq b$. Class: `double`
S.me	Number of equalities in $A_e x = b_e$. Class: `double`
S.problem_type	A string specifying the problem to be solved (only LP and QP are allowed). Class: `char`
S.test	Call (false) or not to call (true) MPT global settings. Class: `logical`

Default: `false`

OUTPUT

R   result structure
                Class: `struct`

R.xopt   optimal solution
                Class: `double`

R.obj   Optimal objective value.
                Class: `double`

R.lambda   Lagrangian multipliers
                Class: `double`

R.exitflag   An integer value that informs if the result was feasible
                (1), or otherwise (different from 1).
                Class: `double`

R.how   A string that informs if the result was feasible ('ok'), or
                if any problem appeared through optimization.
                Class: `char`

SEE ALSO

mpt_solve

LITERATURE

Stephen Boyd and Lieven Vandenberghe: Convex Optimization; Cambridge University Press

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_CALL_LINPROG

A gateway function to LINPROG solver (without errorchecks)

SYNTAX

```
R = mpt_call_linprog(S)
```

DESCRIPTION

The function implements call to LINPROG solver based on formulation from `Opt` class. Only LP problems are supported.
It is assumed that LP entering this function is of the form

$$
\begin{aligned}
\min \quad & f^T x \\
\text{s.t.} \quad & \text{lb} \leq x \leq \text{ub} \\
& Ax \leq b \\
& A_e x = b_e
\end{aligned}
$$

which is passed to LINPROG solver directly.

INPUT

S	structure of the Opt class Class: `struct`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \leq b$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.lb	Lower bound for the variables $x \geq$ lb. Class: `double` Default: `[]`
S.ub	Upper bound for the variables $x \leq$ ub. Class: `double`

Default: []

S.n	Problem dimension (number of variables). Class: `double`
S.m	Number of inequalities in $Ax \leq b$. Class: `double`
S.me	Number of equalities in $A_e x = b_e$. Class: `double`
S.problem_type	A string specifying the problem to be solved. Class: `char`
S.test	Call (false) or not to call (true) MPT global settings. Class: `logical` Default: `false`

OUTPUT

R	result structure Class: `struct`
R.xopt	Optimal solution. Class: `double`
R.obj	Optimal objective value. Class: `double`
R.lambda	Lagrangian multipliers. Class: `double`
R.exitflag	An integer value that informs if the result was feasible (1), or otherwise (different from 1). Class: `double`
R.how	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization. Class: `char`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

401

MPT_CALL_QPSPLINE

A gateway function to QPspline solver (without errorchecks)

SYNTAX

```
R = mpt_call_qpspline(S)
```

DESCRIPTION

The function implements call to QPspline solver based on formulation from `Opt` class. Only QP problems are supported with positive definite Hessian.
It is assumed that QP entering this function is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & Ax \le b \\
& A_e x = b_e
\end{aligned}
$$

which must be transformed to

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H_f x + {f_f}^T x \\
\text{s.t.} \quad & l \le A_f x \le u
\end{aligned}
$$

which accepts QPspline. The lower bound l is always set as `-MPTOPTIONS.infbound`.
If QP contains equality constraints, these are removed first. It is required that the system of linear equations $A_e x = b_e$ is consistent, i.e. no linearly dependent rows are found and the number of equalities is strictly less than number of variables. The principle is based on factorizing equality constraints $A_e x = b_e$ in basic x_{Bc} and non-basic variables x_{Nc}, i.e.

$$
A_e = \begin{pmatrix} A_{e,Bc} & A_{e,Nc} \end{pmatrix}
$$

which gives

$$
A_{e,Bc} x_{Bc} + A_{e,Nc} x_{Nc} = b_e
$$

where the index sets `Bc`, `Nc` denote the columns from which factored system is built. The factored submatrix $A_{e,Bc}$ must be invertible in order to express basic variables as a function of non-basic variables, i.e.

$$
x_{Bc} = -A_{e,Bc}^{-1} A_{e,Nc} x_{Nc} + A_{e,Bc}^{-1} b_{e,Bc}
$$

With the substitution

$$
C = -A_{e,Bc}^{-1} A_{e,Nc}
$$

and

$$
D = A_{e,Bc}^{-1} b_{e,Bc}
$$

the relation between basic and non-basic variables is simplified to

$$
x_{Bc} = C x_{Nc} + D \tag{104}
$$

The above QP problem $(97)-(99)$ can be expressed only in non-basic variables x_{Nc} as follows:

$$
\begin{aligned}
\min \quad & \frac{1}{2} x_{Nc}^T \tilde{H} x_{Nc} + \tilde{f}^T x_{Nc} + \tilde{c} \\
\text{s.t.} \quad & \tilde{A} x_{Nc} \leq \tilde{b}
\end{aligned}
$$

where

$$
\begin{aligned}
\tilde{H} &= C^T H_{Bc,Bc} C + C^T H_{Bc,Nc} + H_{Nc,Bc} C + H_{Nc,Nc} \\
\tilde{f} &= D^T H_{Bc,Bc} C + D^T H_{Nc,Bc} + f_{Bc}^T C + f_{Nc}^T \\
\tilde{c} &= \frac{1}{2} D^T H_{Bc,Bc} D + f_{Bc}^T D \\
\tilde{A} &= A_{Bc} C + A_{Nc} \\
\tilde{b} &= b - A_{Bc} D
\end{aligned}
$$

Original solution to QP problem $(97)-(99)$ can be obtained via relation ().

INPUT

`S`	structure of the Opt class Class: `struct`
`S.H`	Quadratic part of the objective function which is strictly convex $H \succ 0$. Class: `double`
`S.f`	Linear part of the objective function. Class: `double`
`S.A`	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
`S.b`	Right hand side of the inequality constraints $Ax \leq b$. Class: `double`
`S.Ae`	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
`S.be`	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
`S.lb`	Lower bound for the variables $x \geq l_b$. Class: `double` Default: `[]`

S.ub	Upper bound for the variables $x \leq u_b$. Class: `double` Default: `[]`
S.n	Problem dimension (number of variables). Class: `double`
S.m	Number of inequalities in $Ax \leq b$. Class: `double`
S.me	Number of equalities in $A_e x = b_e$. Class: `double`
S.problem_type	A string specifying the problem to be solved. Class: `char`
S.test	Call (false) or not to call (true) MPT global settings. Class: `logical` Default: `false`

OUTPUT

R	result structure Class: `struct`
R.xopt	Optimal solution. Class: `double`
R.obj	Optimal objective value. Class: `double`
R.lambda	Lagrangian multipliers Class: `double`
R.exitflag	An integer value that informs if the result was feasible (1), or otherwise (different from 1). Class: `double`
R.how	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization. Class: `char`

SEE ALSO

mpt_solve

AUTHOR(s)

MPT_CALL_MPQP

A gateway function to MPQP solver (without errorchecks)

SYNTAX

```
R = mpt_call_mpqp(S)
```

DESCRIPTION

The function call to MPQP solver from `Opt` class.
Using option settings for MPQP solver taken from MPT2.6.

INPUT

S	Object of the `Opt` class Class: `Opt`
S.H	Quadratic part of the objective function. Class: `double`
S.f	Linear part of the objective function. Class: `double`
S.pF	Linear part of the objective function for parameters. Class: `double` Default: `[]`
S.A	Linear part of the inequality constraints $Ax \leq b + B\theta$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \leq b + B\theta$. Class: `double`
S.pB	Right hand side of the inequality constraints for parameters $Ax \leq b + B\theta$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e + E\theta$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e + E\theta$. Class: `double` Default: `[]`
S.pE	Right hand side of the equality constraints for parameters $A_e x = b_e + E\theta$. Class: `double`

Default: []

S.lb	Lower bound for the decision variables $x \geq \text{lb}$.
	Class: double
	Default: []

S.ub	Upper bound for the decision variables $x \leq \text{ub}$.
	Class: double
	Default: []

S.Ath	Linear part of the inequality constraints $A_\theta \theta \leq b_\theta$.
	Class: double
	Default: []

S.bth	Right hand side of the inequality constraints $A_\theta \theta \leq b_\theta$.
	Class: double
	Default: []

S.M	Linear matrix involved in LCP.
	Class: double
	Default: []

S.q	Right hand side vector involved in LCP.
	Class: double
	Default: []

S.Q	Linear matrix involved in parametric formulation of LCP.
	Class: double
	Default: []

S.n	Number of decision variables.
	Class: double

S.d	Number of parameters.
	Class: double

S.m	Number of inequalities in $Ax \leq b + B\theta$.
	Class: double

S.me	Number of equalities in $A_e x = b_e + E\theta$.
	Class: double

S.problem_type	A string specifying the problem to be solved
	Class: char
	Default: []

S.varOrder	Order of variables if the problem was processed by YALMIP first.
	Class: double

Default: []

S.Internal   Internal property of Opt class.
                    Class: struct
                    Default: []

S.recover   Affine map for MPQP problems if there were
                    any equalities present and have been removed by
                    eliminateEquations method.
                    Class: struct

S.recover.Y   Matrix of the affine map $x = \mathrm{Y}y + \mathrm{th}\begin{pmatrix}\theta\\1\end{pmatrix}$. The map is
                    from the optimization variables involed in the reduced
                    MPQP $y(\theta) \mapsto x$ to the original MPQP.
                    Class: double
                    Default: []

S.recover.th   Matrix of the affine map $x = \mathrm{Y}y + \mathrm{th}\begin{pmatrix}\theta\\1\end{pmatrix}$. The map is
                    from the optimization variables involed in the reduced
                    MPQP $y(\theta) \mapsto x$ to the original MPQP.
                    Class: double
                    Default: []

                    Default: []

OUTPUT

R   result structure
                                     Class: struct

R.xopt   Optimal solution
                                     Class: PolyUnion

R.mpqpsol   Structure with the solution as returned by MPQP
                                     solver.
                                     Class: struct

R.mpqpsol.Pn   Array of polytopes in MPT2 format.
                                     Class: polytope

R.mpqpsol.Fi   Cell array of matrices of the control law given as $x = F_i\theta + G_i$.
                                     Class: cell

R.mpqpsol.Gi   Cell array of matrices of the control law given as $x = F_i\theta + G_i$.
                                     Class: cell

`R.mpqpsol.activeConstraints`	Index set of active constraints Class: `cell`
`R.mpqpsol.Phard`	Feasible domain. Class: `polytope`
`R.mpqpsol.details`	More details about the solution and the computation. Class: `struct`

SEE ALSO

Opt, mpt_solvemp

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_SOLVERS_OPTIONS

Global option settings for solvers.

SYNTAX

```
s = mpt_solvers_options
```

DESCRIPTION

This function returns a structure with default option settings for solvers used in MPT. Used by `mptopt` class by first time initialization of MPT. Later these settings are changed via `mptopt` class.

OUTPUT

`s`	options structure Class: `struct`
`s.clp`	settings for CLP solver Class: `struct`
`s.clp.solver`	Which method to choose for solving LP/QP. Class: `double` Allowed values: 1 primal 2 dual Default: `1`
`s.clp.maxnumiterations`	Maximum number of iterations allowed Class: `double` Default: `99999999`
`s.clp.maxnumseconds`	Maximum time allowed for computation in seconds Class: `double` Default: `3600`
`s.clp.primaltolerance`	Tolerance on the primal solution. Class: `double` Default: `1e-7`
`s.clp.dualtolerance`	Tolerance on the dual solution. Class: `double` Default: `1e-7`
`s.clp.primalpivot`	Which pivot method to choose for primal solution. Class: `double` Allowed values: 1 steepest 2 Dantzig Default: `1`
`s.clp.dualpivot`	Which pivot method to choose for dual solution.

410

	Class: `double`
	Allowed values: 1 steepest
	2 Dantzig
	Default: `1`
`s.clp.verbose`	Verbosity level.
	Class: `double`
	Allowed values: 0 silent
	1 verbose
	2 loud
	Default: `0`
`s.cplexint`	Settings for CPLEX solver interfaced by Automatic
	Control Laboratory (cplexint).
	Class: `struct`
`s.cplexint.verbose`	Verbosity level.
	Class: `double`
	Allowed values: 0 silent
	1 verbose
	2 loud
	Default: `0`
`s.cplexint.logfile`	Redirect output to a file.
	Class: `logical`
	Allowed values: 0
	1
	Default: `0`
`s.cplexint.lic_rel`	After how many runs to release the license.
	Class: `double`
	Default: `1e3`
`s.cplex`	Settings for CPLEX solver interfaced by IBM.
	Class: `struct`
`s.cplex.lpmethod`	Which method to use for solving LP.
	Class: `char`
	Allowed values: 0 automatic
	1 primal simplex
	2 dual simplex
	3 network simplex
	4 barrier
	5 sifting
	6 concurrent
	Default: `2`
`s.cplex.qpmethod`	Which method to use for solving QP.
	Class: `char`

Allowed values:	0	automatic
	1	primal simplex
	2	dual simplex
	3	network simplex
	4	barrier
	5	sifting
	6	concurrent

Default: 2

s.plcp	settings for PLCP solver
	Class: struct

s.plcp.bfs	Perform breadth first search when explorting the parameter space.
	Class: logical
	Allowed values: 0
	1
	Default: 1

s.plcp.dfs	Perform breadth first search when explorting the parameter space.
	Class: logical
	Allowed values: 0
	1
	Default: 0

s.plcp.debug	Debugging level
	Class: double
	Allowed values: 0 no debugging
	1 no plots
	2 including plots
	Default: 0

s.plcp.fixedstep	Always perform a step over the facet with a fixed step size given as region_tol. An alternative way to detect regions while exploring the parameter space. The approach is useful when numerical problems occurs with the standard method where the step size is computed automatically.
	Class: logical
	Allowed values: 0
	1
	Default: 0

s.plcp.QRfactor	Inside the main pivoting funtion use recursive QR factorization instead of direct LU factorization. It will speed up the computation time but will suffer from numerical problems for large LCP problems.
	Class: logical
	Allowed values: 0
	1

Default: 0

s.plcp.checkoverlaps   Check for overlaps while exploring the parameter space. This option enforces to search through all discovered regions at each step for overlaps. If the overlap is detected, the overlapping part is discarded via set-difference operation. Since this requires solving an LP for each region, this option reduces the computational time significantly.
Class: `logical`
Allowed values: 0
                 1
Default: 0

s.plcp.rescue   If the variable step approach fails to find adjacent region, retry with the fixed step. The step size is given as the Chebyshev radius of the smallest allowable region, i.e. half of the `region_tol`.
Class: `logical`
Allowed values: 0
                 1
Default: 0

s.plcp.maxsteps   Maximum number of steps to perform with the fixed step approach in case no new regions have been found. Minimum 2-steps must be provided.
Class: `double`
Default: 200

s.plcp.maxlayers   Maximum value of layers to be explored for breadth first search (BFS). This option is valid only with BFS option. Useful only if there are too many regions far from the initial region that can be discarded from optimal solution. The layered list of regions is returned in the field `layer_list`.
Class: `double`
Default: `Inf`

s.plcp.maxregions   Maximum number of regions to be generated.
Class: `double`
Default: `Inf`

s.plcp.maxpivots   The maximum number of pivots to be performed to find a neighboring region. Typically, the are 1, 2 pivots to be performed to find a neighboring regions. For a degenerate case, more pivots are needed. The maximum value of the pivots is given by the `RecursionLimit` of MATLAB which 500 by default. If the value of maxpivots is greater than `RecursionLimit`, change also this value by `set(0,'RecursionLimit',N)`.
Class: `double`

413

Default: 100

s.plcp.adjcheck	Force verification of the adjacency list in the postprocessing phase. This option activates a subfunction in the PLCP solver where the adjacency list is checked for any missing links in the graph and performs correction. The option is turned off by default because the verification can be time consuming for large partitions. Class: `logical` Allowed values: 0 1 Default: 0

s.lcp settings for LCP solver
Class: `struct`

s.lcp.zerotol Less than this treshold the value is considered as zero.
Class: `double`
Default: `1e-10`

s.lcp.lextol Lexicographic tolerance - a small treshold from which values are considered as equal.
Class: `double`
Default: `1e-9`

s.lcp.nstepf If `options.routine` is 0, then every `nstepf` pivot steps the basis is refactorized to avoid numerical problems for LUMOD. For other routines the factorization is performed at each step.
Class: `double`
Default: 50

s.lcp.maxpiv Maximum number of pivots to be performed.
Class: `double`
Default: `1e4`

s.lcp.clock Show the information about the computational time.
Class: `logical`
Allowed values: 0
                 1
Default: 0

s.lcp.verbose Verbose output.
Class: `logical`
Allowed values: 0
                 1
Default: 0

s.lcp.routine Routine which should be used to obtain a basis solution.
Class: `double`

Allowed values: 0 Corresponds to LUmod package that performs factorization in the form $LA = U$. Depending on the change in A factors L, U are updated. This is the fastest method.

1 Corresponds to DGESV simple driver from LAPACK package which solves the system $AX = B$ by factorizing A and overwriting B with the solution X. Since the factorization is performed at each pivot step, this method tends to be much slower than method 0.

2 Corresponds to DGELS simple driver which solves overdetermined or underdetermined real linear systems min $\|b - Ax\|_2$ involving an M-by-N matrix A, or its transpose, using a QR or LQ factorization of A. Since the factorization is performed at each pivot step, this method tends to be much slower than method 0.

Default: `1`

s.lcp.timelimit   Time limit in seconds. If this limit is exceeded, the pivoting algorithm is terminated and current basis is returned.
Class: `double`
Default: `3600`

s.lcp.normalize   Input matrices M, q get scaled by D_1 and D_2 when invoking this option: $M_n = D_1 M D_2$, $q_n = D_1 q$, and solution is recovered as $z = D_2 z_n$, $w = Mz + q$.
Class: `logical`
Allowed values: 0
                1
Default: `1`

s.lcp.normalizethres   If the `normalize` option is on, then the matrix scaling is performed only if 1 norm of matrix M (maximum absolute column sum) is above this threshold.
Class: `double`
Default: `1e6`

s.glpk   settings for GLPK solver
Class: `struct`

s.glpk.msglev   Level of messages output by solver routines.
Class: `double`
Allowed values: 0 No output.
                1 Error messages only.
                2 Normal output.
                3 Full output (includes informational messages).
Default: `0`

`s.glpk.lpsolver`	Which method to choose for solving primal LP.
	Class: `double`
	Allowed values: `1` Revised simplex method.
	                `2` Interior point method.
	                `3` Simplex method with exact arithmetic.
	Default: `1`

`s.glpk.scale`	Which method of scaling to use.
	Class: `double`
	Allowed values: `0` No scaling.
	                `1` Equilibration scaling.
	                `2` Geometric mean scaling, then equilibration scaling.
	                `3` Geometric then Equilibrium scaling.
	                `4` Round to nearest power of 2 scaling.
	Default: `1`

`s.glpk.dual`	Which method to choose for solving dual LP.
	Class: `double`
	Allowed values: `0` Do not use the dual simplex.
	                `1` If initial basic solution is dual feasible, use the dual simplex.
	                `2` Use two phase dual simplex, or if primal simplex if dual fails.
	Default: `1`

`s.glpk.price`	Pricing option (for both primal and dual simplex). Class: `double` Allowed values: `0` Textbook pricing. `1` Steepest edge pricing. Default: `1`
`s.glpk.r_test`	Ratio test technique. Class: `double` Allowed values: `0` Standart (textbook). `1` Harris's two-pass ratio test. Default: `1`
`s.glpk.relax`	Relaxation parameter used in the ratio test. If it is zero, the textbook ratio test is used. If it is non-zero (should be positive), Harris two-pass ratio test is used. In the latter case on the first pass of the ratio test basic variables (in the case of primal simplex) or reduced costs of non-basic variables (in the case of dual simplex) are allowed to slightly violate their bounds, but not more than `relax*tolbnd` or `relax*toldj` (thus, `relax` is a percentage of `tolbnd` or `toldj`). Class: `double` Default: `0.07`

`s.glpk.tolbnd`	Relative tolerance used to check ifthe current basic solution is primal feasible. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose. Class: `double` Default: `1e-7`
`s.glpk.toldj`	Absolute tolerance used to check if the current basic solution is dual feasible. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose. Class: `double` Default: `1e-7`
`s.glpk.tolpiv`	Relative tolerance used to choose eligible pivotal elements of the simplex table. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose. Class: `double` Default: `1e-9`
`s.glpk.round`	Solution rounding option. Class: `logical` Allowed values: `0` Report all primal and dual values "as is" (default). `1` Replace tiny primal and dual values by exact zero. Default: `0`
`s.glpk.objll`	Lower limit of the objective function. If on the phase II the objective function reaches this limit and continues decreasing, the solver stops the search. This parameter is used in the dual simplex method only. Class: `double` Default: `-1e12`
`s.glpk.objul`	Upper limit of the objective function. If on the phase II the objective function reaches this limit and continues increasing, the solver stops the search. This parameter is used in the dual simplex only. Class: `double` Default: `1e12`
`s.glpk.itlim`	Simplex iterations limit. If this value is positive, it is decreased by one each time when one simplex iteration has been performed, and reaching zero value signals the solver to stop the search. Negative value means no iterations limit. Class: `double` Default: `1e4`

`s.glpk.itcnt`	Output frequency, in iterations. This parameter specifies how frequently the solver sends information about the solution to the standard output. Class: `double` Default: `200`

`s.glpk.usecuts`	`glp_intopt` generates and adds cutting planes to the MIP problem in order to improve its LP relaxation before applying the branch-and-bound method. Class: `double` Allowed values: 0 All cuts off. 1 Gomoy's mixed integer cuts. 2 Mixed integer rounding cuts. 3 Mixed cover cuts. 4 Clique cuts. 5 All cuts. Default: `1`

`s.glpk.pprocess`	Pre-processing technique option (for MIP only). Class: `double` Allowed values: 0 Disable preprocessing. 1 Perform preprocessing for root only. 2 Perform preprocessing for all levels. Default: `2`

`s.glpk.binarize`	Binarizeation option (for MIP), used only if presolver is enabled. Class: `logical` Allowed values: 0 Do not use binarization. 1 Replace general integer variables by binary ones. Default: `0`

`s.glpk.tmlim`	Searching time limit, in seconds. If this value is positive, it is decreased each time when one simplex iteration has been performed by the amount of time spent for the iteration, and reaching zero value signals the solver to stop the search. Negative value means no time limit Class: `double` Default: `-1`

`s.glpk.branch`	Branching heuristic option (for MIP only). Class: `double` Allowed values: 0 Branch on the first variable. 1 Branch on the last variable. 2 Branch on the most fractional variable. 3 Branch using a heuristic by Driebeck and Tomlin. Default: `2`

`s.glpk.btrack`	Backtracking heuristic option (for MIP only). Class: `double`

418

	Allowed values:	0	Depth first search.
		1	Breadth first search.
		2	Best local bound.
		3	Backtrack using the best projection heuristic.

Default: 2

s.glpk.tolint   Relative tolerance used to check if the current basic solution is integer feasible. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose.
Class: `double`
Default: `1e-6`

s.glpk.outdly   Output delay, in seconds. This parameter specifies how long the solver should delay sending information about the solution to the standard output. Non-positive value means no delay.
Class: `double`
Default: `0`

s.glpk.tolobj   Relative tolerance used to check if the value of the objective function is not better than in the best known integer feasible solution. It is not recommended that you change this parameter unless you have a detailed understanding of its purpose.
Class: `double`
Default: `1e-7`

s.glpk.presol   If this flag is set, the routine `lpx_simplex` solves the problem using the built-in LP presolver. Otherwise the LP presolver is not used.
Class: `logical`
Allowed values: 0
                   1
Default: 1

s.glpk.save   If this parameter is nonzero save a copy of the original problem to file.
Class: `logical`
Allowed values: 0
                   1
Default: 0

s.glpk.mipgap   The relative mip gap tolerance. If the relative mip gap for currently known best integer feasible solution falls below this tolerance, the solver terminates the search. This allows obtaining suboptimal interger feasible solutions if solving the problem to optimality takes too long.
Class: `double`

419

Default: 0

s.gurobi	settings for GUROBI solver Class: struct

s.gurobi.BarIterLimit	Limits the number of barrier iterations performed (barrier only). Class: double Default: 1e12

s.gurobi.CutOff	If the objective value for the optimal solution is better than the specified cutoff, the solver will return the optimal solution. Otherwise, it will terminate with a CUTOFF status. Class: double Default: 1e12

s.gurobi.IterationLimit	Limits the number of simplex iterations performed. Class: double Default: 1e6

s.gurobi.NodeLimit	Limits the number of MIP nodes explored (MIP only). Class: double Default: 1e12

s.gurobi.SolutionLimit	Limits the number of feasible solutions found (MIP only). Class: double Default: 1e12

s.gurobi.TimeLimit	Limits the total time expended (in seconds). Class: double Default: 1e12

s.gurobi.BarConvTol	Barrier convergence tolerance (barrier only). The barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. Value must be in range [1e-10, 1]. Class: double Default: 1e-8

s.gurobi.BarConvTol	Barrier convergence tolerance (barrier only). The barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. Value must be in range [1e-10, 1]. Class: double Default: 1e-8

s.gurobi.BarQCPConvTol   The barrier solver terminates when the relative differ-
                          ence between the primal and dual objective values is
                          less than the specified tolerance. ightening this toler-
                          ance may lead to a more accurate solution, but it may
                          also lead to a failure to converge. Values must be in
                          range [0, 1];
                          Class: `double`
                          Default: `1e-6`

s.gurobi.FeasibilityTol   All constraints must be satisfied to a tolerance of
                          `FeasibilityTol`. Tightening this tolerance can pro-
                          duce smaller constraint violations, but for numerically
                          challenging models it can sometimes lead to much larger
                          iteration counts. Value must be in range [1e-9, 1e-2].
                          Class: `double`
                          Default: `1e-6`

s.gurobi.IntFeasTol   Integer feasibility tolerance (MIP only). An integrality
                          restriction on a variable is considered satisfied when the
                          variable's value is less than `INTFEASTOL` from the nearest
                          integer value. Value must be in range [1e-9, 1e-1].
                          Class: `double`
                          Default: `1e-5`

s.gurobi.OptimalityTol   Dual feasibility tolerance. Reduced costs must all be
                          smaller than OptimalityTol in the improving direction
                          in order for a model to be declared optimal. Value must
                          be in range [1e-9, 1e-2].
                          Class: `double`
                          Default: `1e-6`

s.gurobi.MIPGap   Relative MIP optimality gap (MIP only). The MIP en-
                          gine will terminate (with an optimal result) when the
                          gap between the lower and upper objective bound is less
                          than MIPGap times the upper bound.
                          Class: `double`
                          Default: `1e-4`

s.gurobi.PSDTol   Positive semi-definite tolerance (QP/MIQP only). Sets
                          a limit on the amount of diagonal perturbation that
                          the optimizer is allowed to perform on the Q matrix
                          in order to correct minor PSD violations. If a larger
                          perturbation is required, the optimizer will terminate
                          with an `GRB_ERROR_Q_NOT_PSD` error.
                          Class: `double`
                          Default: `1e-6`

s.gurobi.InfUnbdInfo | Determines whether simplex (and crossover) will compute additional information when a model is determined to be infeasible or unbounded. Set this parameter if you want to query the unbounded ray for unbounded models (through the UnbdRay attribute), or the infeasibility proof for infeasible models (through the FarkasDual and FarkasProof attributes).
Class: `logical`
Allowed values: 0
                  1
Default: `0`

s.gurobi.NormAdjust | Chooses from among multiple pricing norm variants. The details of how this parameter affects the simplex pricing algorithm are subtle and difficult to describe, so we've simply labeled the options 0 through 3. The default value of -1 chooses automatically. Changing the value of this parameter rarely produces a significant benefit.
Class: `double`
Default: `-1`

s.gurobi.PerturbValue | Magnitude of the simplex perturbation. Note that perturbation is only applied when progress has stalled, so the parameter will often have no effect. Values must be in range [0, 0.01].
Class: `double`
Default: `0.0002`

s.gurobi.ScaleFlag | Enables or disables model scaling. Scaling usually improves the numerical properties of the model, which typically leads to reduced solution times, but it may sometimes lead to larger constraint violations in the original, unscaled model.
Class: `logical`
Allowed values: 0
                  1
Default: `1`

s.gurobi.ObjScale | Divides the model objective by the specified value to avoid numerical errors that may result from very large objective coefficients. The default value of 0 decides on the scaling automatically. A value less than zero uses the maximum coefficient to the specified power as the scaling (so `ObjScale=-0.5` would scale by the square root of the largest objective coefficient).
Class: `double`
Default: `0`

s.gurobi.BarCorrectors   Limits the number of central corrections performed in each barrier iteration. The default value chooses automatically, depending on problem characteristics. The automatic strategy generally works well, although it is often possible to obtain higher performance on a specific model by selecting a value manually. The values must be in range [-1, Inf)
Class: `double`
Default: `-1`

s.gurobi.Method   Algorithm used to solve continuous models or the root node of a MIP model. Concurrent optimizers run multiple solvers on multiple threads simultaneously, and choose the one that finishes first. Deterministic concurrent (Method=4) gives the exact same result each time, while Method=3 is often faster but can produce different optimal bases when run multiple times. In the current release, the default Automatic (Method=-1) will typically choose non-deterministic concurrent (Method=3) for an LP, barrier (Method=2) for a QP or QCP, and dual (Method=1) for the MIP root node. Only simplex and barrier algorithms are available for continuous QP models. Only primal and dual simplex are available for solving the root of an MIQP model. Only barrier is available for continuous QCP models.

The default setting is rarely significantly slower than the best possible setting, so you generally won't see a big gain from changing this parameter. There are classes of models where one particular algorithm is consistently fastest, though, so you may want to experiment with different options when confronted with a particularly difficult model.

Note that if memory is tight on an LP model, you should consider choosing the dual simplex method (Method=1). The default will invoke the concurrent optimizer, which typically consumes a lot more memory than dual simplex alone.
Class: `double`
Allowed values: `-1` automatic
             `0` primal simplex
             `1` dual simplex
             `2` barrier
             `3` concurrent
             `4` deterministic concurrent
Default: `1`

s.gurobi.Presolve   Controls the presolve level. A value of -1 corresponds to an automatic setting.
Class: `double`

423

	Allowed values:	-1 Automatic
		0 Off
		1 Conservative
		2 Aggressive

Default: -1

s.gurobi.TimeLimit	Limits the total time expended (in seconds). Class: double Default: 1e12

s.gurobi.Threads	Controls the number of threads to apply to parallel MIP. The default value of 0 sets the thread count equal to the maximum value, which is the number of processors in the machine. Value should range from 0 to maximum number of processors. Class: double Default: 0

s.gurobi.OutputFlag	Verbosity level. Class: logical Allowed values: 0 1 Default: 0

s.gurobi.DisplayInterval	Controls the frequency at which log lines are printed (in seconds). Class: double Default: 5

s.nag	settings for NAG solver Class: struct

s.nag.qp	settings for QP solver Class: struct

s.nag.qp.ftol	The maximum acceptable violation in each constraint at a "feasible" point. Class: double Default: 1e-9

s.nag.qp.rank_tol	Enables the user to control the condition number of the triangular factor R. Class: double Default: 1e-20

s.nag.qp.crash_tol	A constraint of the form $a^T x \geq l$ will be included in the initial working set if $|a^T x - l| \leq$ crash_tol $\times (1 + |l|)0.0 <$ options.crash_tol ≤ 1.0. Class: double

Default: `0.1`

`s.nag.qp.reset_ftol`	This option is part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems. Class: `double` Default: `5`
`s.nag.qp.max_iter`	maximum number of iterations to be performed Class: `double` Default: `1e6`
`s.nag.qp.fcheck`	every fcheck iterations, a numerical test is made to see if the current solution satisfies the constraints in the working set Class: `double` Default: `50`
`s.nag.qp.inf_bound`	defines the "infinite" bound in the definition of the problem constraints Class: `double` Default: `1e12`
`s.nag.lp`	settings for LP solver Class: `struct`
`s.nag.lp.ftol`	The maximum acceptable violation in each constraint at a "feasible" point. Class: `double` Default: `1e-9`
`s.nag.lp.optim_tol`	Enables the user to control the condition number of the triangular factor R. Class: `double` Default: `1e-13`
`s.nag.lp.crash_tol`	A constraint of the form $a^T x \geq l$ will be included in the initial working set if $|a^T x - l| \leq \mathrm{crash\_tol} \times (1 + |l|)0.0 < \mathrm{options.crash\_tol} \leq 1.0$. Class: `double` Default: `0.1`
`s.nag.lp.reset_ftol`	This option is part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems. Class: `double` Default: `5`
`s.nag.lp.max_iter`	maximum number of iterations to be performed

425

Class: `double`
Default: `1e6`

s.nag.lp.fcheck — every fcheck iterations, a numerical test is made to see if the current solution satisfies the constraints in the working set
Class: `double`
Default: `50`

s.nag.lp.inf_bound — defines the "infinite" bound in the definition of the problem constraints
Class: `double`
Default: `1e12`

s.qpip — settings for QPC interior point solver "qpip"
Class: `struct`

s.qpip.mu — Desired complementarity gap target (point on central-path).
Class: `double`
Default: `0`

s.qpip.method — If `method=1`, then a faster but less accurate linear solve step is used. Conversely, if `method=0` then a slower but more accurate linear solve step is used.
Class: `logical`
Default: `0`

s.qpspline — settings for QPspline solver
Class: `struct`

s.qpspline.maxiter — The maximum number of iterations.
Class: `double`
Default: `10000`

s.qpspline.abs_tol — Absolute tolerance.
Class: `double`
Default: `1e-8`

s.qpspline.nstepf — After these `nstepf` steps the basis will be refactored with new Q, R, factors.
Class: `double`
Default: `30`

s.qpspline.nqrelems — Do recursive QR factorization if the number of changed elements (rows/cols) is less than this treshold. If the treshold is large, the recursive factorization may actually consume more time than direct factorization. Large treshold causes also accumulation of numerical errors.
Class: `double`

Default: `20`

`s.qpspline.timelimit`	Time limit on the computations in seconds.
	Class: `double`
	Default: `3600`

`s.qpspline.verbose`	Show the iteration progress.
	Class: `logical`
	Allowed values: `0`
	`1`
	Default: `0`

`s.quadprog`	settings for QUADPROG solver
	Class: `struct`

`s.quadprog.MaxIter`	Maximum number of iterations allowed.
	Class: `double`
	Default: `1e6`

`s.quadprog.TolFun`	Termination tolerance on the function value.
	Class: `double`
	Default: `1e-10`

`s.quadprog.TolX`	Termination tolerance on the solution.
	Class: `double`
	Default: `1e-10`

`s.quadprog.Display`	Display progress of optimization.
	Class: `char`
	Allowed values: `notify`
	`iter`
	`final`
	Default: `off`

`s.quadprog.Algorithm`	Algorithm for solving the QP
	Class: `char`
	Allowed values: `active-set`
	`interior-point`
	`interior-point-convex`
	`levenberg-marquardt`
	`sqp`
	`trust-region-dogleg`
	`trust-region-reflective`
	Default: `active-set`

`s.quadprog.LargeScale`	Use large-scale or medium-scale algorithms.
	Class: `char`
	Allowed values: `on`
	`off`
	Default: `off`

427

`s.linprog`	settings for LINPROG solver Class: `struct`
`s.linprog.MaxIter`	Maximum number of iterations allowed. Class: `double` Default: `1e6`
`s.linprog.TolFun`	Termination tolerance on the function value. Class: `double` Default: `1e-10`
`s.linprog.TolX`	Termination tolerance on the solution. Class: `double` Default: `1e-10`

`s.linprog.Display`	Display progress of optimization.
	Class: `char`
	Allowed values: `notify`
	               `iter`
	               `final`
	Default: `off`

`s.sedumi`	settings for SEDUMI solver Class: `struct`
`s.sedumi.fid`	Verbosity level Class: `logical` Allowed values: `0` silent `1` loud Default: `0`

`s.sedumi.alg`	Type of algorithm that solves the problem.
	Class: `double`
	Allowed values: `0` The first-order wide region algorithm is used, not recommended.
	                 `1` The centering-predictor-corrector algorithm is used with v-linearization.
	                 `2` The xz-linearization is used in the corrector, similar to Mehrotra's algorithm.
	Default: `2`

`s.sedumi.theta`	The wide region parameter which varies $0 < \mathtt{theta} \leq 1$. Class: `double` Default: `0.25`
`s.sedumi.beta`	The neighborhood region parameter which varies $0 < \mathtt{beta} \leq 1$. Class: `double` Default: `0.5`

s.sedumi.stepdif	Set primal/dual differentiation step length. Class: double Default: 2
s.sedumi.w	The weights for the relative primal, dual and gap residuals as w(1):w(2):1 in order to find the optimal step differentiation. Class: double Default: [1 1]
s.sedumi.eps	The desired accuracy. Class: double Default: 1e-10
s.sedumi.bigeps	In case the desired accuracy cannot be achieved, SEDUMI tries to satisfy this accuracy. Class: double Default: 1e-6
s.sedumi.maxiter	Maximum iterations allowed. Class: double Default: 1e6
s.sedumi.cg	Settings for preconditioned conjugate gradient method (CG), which is only used if results from Cholesky are inaccurate. Class: struct
s.sedumi.cg.maxiter	Maximum number of CG-iterates (per solve). Theoretically needed is |add|+2*|skip|, the number of added and skipped pivots in Cholesky. Class: double Default: 49
s.sedumi.cg.restol	Terminates if residual is a restol fraction of duality gap. Should be smaller than 1 in order to make progress. Class: double Default: 5e-3
s.sedumi.cg.refine	Number of refinement loops that are allowed. The maximum number of actual CG-steps will thus be 1+(1+refine)*maxiter. Class: double Default: 1
s.sedumi.cg.stagtol	Terminates if relative function progress less than stagtol. Class: double Default: 4e-14

`s.sedumi.cg.qprec`	Stores cg-iterates in quadruple precision if true. Class: `logical` Allowed values: 0 1 Default: 0
`s.sedumi.chol`	Parameters for controling the Cholesky solve. Class: `struct`
`s.sedumi.chol.canceltol`	Relative tolerance for detecting cancelation during Cholesky. Class: `double` Default: `1e-12`
`s.sedumi.chol.maxu`	Adds to diagonal if `max(abs(L(:,j))) > maxu` otherwise. Class: `double` Default: `5e5`
`s.sedumi.chol.abstol`	Skips pivots falling below `abstol`. Class: `double` Default: `1e-20`
`s.sedumi.chol.maxuden`	Pivots in dense-column factorization so that these factors satisfy `max(abs(Lk)) <= maxuden`. Class: `double` Default: `5e2`
`s.sedumi.chol.errors`	If this field is true then SEDUMI outputs some error measures as defined in the Seventh DIMACS Challenge. Class: `logical` Allowed values: 0 1 Default: 0

SEE ALSO

mptopt, mpt_solve

LITERATURE

Stephen Boyd and Lieven Vandenberghe: Convex Optimization; Cambridge University Press

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_CALL_NAG

A gateway function to the NAG Toolbox LP and QP solvers

SYNTAX

```
R = mpt_call_nag(S)
```

DESCRIPTION

The function implements call to NAG solver based on formulation from `Opt` class. Only QP and LP problems are supported.

It is assumed that QP/LP entering this function (for LP $H = 0$) is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & \text{lb} \le x \le \text{ub} \\
& Ax \le b \\
& A_e x = b_e
\end{aligned}
$$

which must be transformed to

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & \text{lb} \le x \le \text{ub} \\
& \text{lA} \le A_m x \le \text{uA}
\end{aligned}
$$

Inequality (113) and equality (114) constraints are merged to

$$
\text{lA} = \begin{pmatrix} -\text{MPTOPTIONS.infbound} \\ b_e \end{pmatrix}
$$

$$
A_m = \begin{pmatrix} A \\ A_e \end{pmatrix}
$$

$$
\text{uA} = \begin{pmatrix} b \\ b_e \end{pmatrix}
$$

because NAG accepts equalities written as double-sided inequalities.

INPUT

S	structure of the Opt class Class: `struct`
S.H	Quadratic part of the objective function. Class: `double` Default: `[]`
S.f	Linear part of the objective function.

Class: `double`

S.A   Linear part of the inequality constraints $Ax \leq b$.
Class: `double`

S.b   Right hand side of the inequality constraints $Ax \leq b$.
Class: `double`

S.Ae   Linear part of the equality constraints $A_e x = b_e$.
Class: `double`
Default: `[]`

S.be   Right hand side of the equality constraints $A_e x = b_e$.
Class: `double`
Default: `[]`

S.lb   Lower bound for the variables $x \geq$ lb.
Class: `double`
Default: `[]`

S.ub   Upper bound for the variables $x \leq$ ub.
Class: `double`
Default: `[]`

S.n   Problem dimension (number of variables).
Class: `double`

S.m   Number of inequalities in $Ax \leq b$.
Class: `double`

S.me   Number of equalities in $A_e x = b_e$.
Class: `double`

S.problem_type   A string specifying the problem to be solved.
Class: `char`

S.test   Call (false) or not to call (true) MPT global settings.
Class: `logical`
Default: `false`

S.solver   Specific call of NAG routine to be called. By default, the interface function "mexnagqp" or "mexnaglp" are called. Left as option for future.
Class: `char`

OUTPUT

R   result structure

Class: `struct`

`R.xopt`	Optimal solution. Class: `double`
`R.obj`	Optimal objective value. Class: `double`
`R.lambda`	Lagrangian multipliers. Class: `double`
`R.exitflag`	An integer value that informs if the result was feasible (1), or otherwise (different from 1). Class: `double`
`R.how`	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization. Class: `char`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

MPT_CALL_PLCP

A gateway function to PLCP solver (without errorchecks)

SYNTAX

```
R = mpt_call_plcp(S)
```

DESCRIPTION

The function implements call to PLCP solver from `Opt` class. Supported problems are PLCP, MPLP, and MPQP. For PLCP, the problem is directly passed to `mpt_plcp` solver. For MPLP/MPQP, the problem is first transformed to PLCP and then PLCP solver is called.

INPUT

S	Object of the `Opt` class with the PLCP data. Class: `Opt`
S.Ath	Linear part of the inequality constraints $A_\theta \theta \leq b_\theta$. Class: `double` Default: `[]`
S.bth	Right hand side of the inequality constraints $A_\theta \theta \leq b_\theta$. Class: `double` Default: `[]`
S.M	Linear matrix involved in LCP. Class: `double` Default: `[]`
S.q	Right hand side vector involved in LCP. Class: `double` Default: `[]`
S.Q	Linear matrix involved in parametric formulation of LCP. Class: `double` Default: `[]`
S.n	Number of decision variables. Class: `double`
S.d	Number of parameters. Class: `double`
S.varOrder	Order of variables if the problem was processed by YALMIP first. Class: `double` Default: `[]`

434

S.Internal   Internal property of `Opt` class.
Class: `struct`
Default: `[]`

OUTPUT

R   result structure
Class: `struct`

R.xopt   Optimal solution
Class: `PolyUnion`

R.exitflag   An integer value that informs if the result was feasible (1), or otherwise (different from 1)
Class: `double`

R.how   A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization
Class: `char`

R.solveTime   Information about the time that elapsed during the computation in seconds.
Class: `double`

R.stats   Further details from the parametric solver.
Class: `struct`

SEE ALSO

Opt, mpt_solvemp

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_CALL_QPOASES

A gateway function to QPoases solver (without errorchecks)

SYNTAX

```
R = mpt_call_qpoases(S)
```

DESCRIPTION

The function implements call to QPoases solver based on formulation from `Opt` class. Only QP and LP problems are supported.

It is assumed that QP/LP entering this function (for LP $H = 0$) is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & \text{lb} \leq x \leq \text{ub} \\
& Ax \leq b \\
& A_e x = b_e
\end{aligned}
$$

which must be transformed to

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & \text{lb} \leq x \leq \text{ub} \\
& \text{lA} \leq A_m x \leq \text{uA}
\end{aligned}
$$

which accepts QPspline. Inequality (121) and equality (122) constraints are merged to

$$
\text{lA} = \begin{pmatrix} -\text{MPTOPTIONS.infbound} \\ b_e \end{pmatrix}
$$

$$
A_m = \begin{pmatrix} A \\ A_e \end{pmatrix}
$$

$$
\text{uA} = \begin{pmatrix} b \\ b_e \end{pmatrix}
$$

since QPoases accepts equalities as double-sided inequalities.

INPUT

S	Structure of the `Opt` class. Class: `struct`
S.H	Quadratic part of the objective function. Class: `double` Default: `[]`
S.f	Linear part of the objective function.

Class: `double`

`S.A`	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
`S.b`	Right hand side of the inequality constraints $Ax \leq b$. Class: `double`
`S.Ae`	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
`S.be`	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
`S.lb`	Lower bound for the variables $x \geq \text{lb}$. Class: `double` Default: `[]`
`S.ub`	Upper bound for the variables $x \leq \text{ub}$. Class: `double` Default: `[]`
`S.n`	Problem dimension (number of variables). Class: `double`
`S.m`	Number of inequalities in $Ax \leq b$. Class: `double`
`S.me`	Number of equalities in $A_e x = b_e$. Class: `double`
`S.problem_type`	A string specifying the problem to be solved. Class: `char`
`S.test`	Call (false) or not to call (true) MPT global settings. Class: `logical` Default: `false`

OUTPUT

`R`	result structure Class: `struct`
`R.xopt`	Optimal solution. Class: `double`

`R.obj`	Optimal objective value. Class: `double`
`R.lambda`	Lagrangian multipliers. Class: `double`
`R.exitflag`	An integer value that informs if the result was feasible (1), or otherwise (different from 1). Class: `double`
`R.how`	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization. Class: `char`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

MPT_CALL_CLP

A gateway function to CLP solver (without errorchecks)

SYNTAX

```
R = mpt_call_clp(S)
```

DESCRIPTION

The function implements call to CLP solver based on formulation from `Opt` class. Only QP and LP problems are supported.

It is assumed that QP/LP entering this function (for LP $H = 0$) is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & \text{lb} \le x \le \text{ub} \\
& Ax \le b \\
& A_e x = b_e
\end{aligned}
$$

which is passed to CLP solver directly.

INPUT

S	Structure of the Opt class Class: `struct`
S.H	Quadratic part of the objective function. Class: `double` Default: `[]`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \le b$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \le b$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.lb	Lower bound for the variables $x \ge \text{lb}$. Class: `double`

Default: []

S.ub   Upper bound for the variables $x \leq$ ub.
                Class: double
                Default: []

S.n   Problem dimension (number of variables).
                Class: double

S.m   Number of inequalities in $Ax \leq b$.
                Class: double

S.me   Number of equalities in $A_e x = b_e$.
                Class: double

S.problem_type A string specifying the problem to be solved.
                Class: char

S.test   Call (false) or not to call (true) MPT global settings.
                Class: logical
                Default: false

OUTPUT

R   result structure
                Class: struct

R.xopt   Optimal solution.
                Class: double

R.obj   Optimal objective value.
                Class: double

R.lambda   Lagrangian multipliers.
                Class: double

R.exitflag   An integer value that informs if the result was feasible
                (1), or otherwise (different from 1).
                Class: double

R.how   A string that informs if the result was feasible ('ok'), or
                if any problem appeared through optimization.
                Class: char

SEE ALSO

mpt_solve

440

AUTHOR(s)

MPT_CALL_MPLP

A gateway function to MPLP solver (without errorchecks)

SYNTAX

```
R = mpt_call_mplp(S)
```

DESCRIPTION

The function call to MPLP solver from `Opt` class. Note that this solver is not capable of solving MPLP with the parameterized cost function, i.e. if there is non-zero `pF` term. Using option settings for MPLP solver taken from MPT2.6.

INPUT

S	Object of the `Opt` class Class: `Opt`
S.H	Quadratic part of the objective function. Class: `double` Default: `0`
S.f	Linear part of the objective function. Class: `double`
S.pF	Linear part of the objective function for parameters. Class: `double` Default: `0`
S.A	Linear part of the inequality constraints $Ax \leq b + B\theta$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \leq b + B\theta$. Class: `double`
S.pB	Right hand side of the inequality constraints for parameters $Ax \leq b + B\theta$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e + E\theta$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e + E\theta$. Class: `double` Default: `[]`

`S.pE`	Right hand side of the equality constraints for parameters $A_e x = b_e + E\theta$. Class: `double` Default: `[]`
`S.lb`	Lower bound for the decision variables $x \geq \text{lb}$. Class: `double` Default: `[]`
`S.ub`	Upper bound for the decision variables $x \leq \text{ub}$. Class: `double` Default: `[]`
`S.Ath`	Linear part of the inequality constraints $A_\theta \theta \leq b_\theta$. Class: `double` Default: `[]`
`S.bth`	Right hand side of the inequality constraints $A_\theta \theta \leq b_\theta$. Class: `double` Default: `[]`
`S.M`	Linear matrix involved in LCP. Class: `double` Default: `[]`
`S.q`	Right hand side vector involved in LCP. Class: `double` Default: `[]`
`S.Q`	Linear matrix involved in parametric formulation of LCP. Class: `double` Default: `[]`
`S.n`	Number of decision variables. Class: `double`
`S.d`	Number of parameters. Class: `double`
`S.m`	Number of inequalities in $Ax \leq b + B\theta$. Class: `double`
`S.me`	Number of equalities in $A_e x = b_e + E\theta$. Class: `double`
`S.problem_type`	A string specifying the problem to be solved Class: `char` Default: `[]`

443

`S.varOrder`	Order of variables if the problem was processed by YALMIP first. Class: `double` Default: `[]`
`S.Internal`	Internal property of `Opt` class. Class: `struct` Default: `[]`
`S.recover`	Affine map for MPLP problems if there were any equalities present and have been removed by `eliminateEquations` method. Class: `struct`
`S.recover.Y`	Matrix of the affine map $x = \mathrm{Y}y + \mathrm{th} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in the reduced MPLP $y(\theta) \mapsto x$ to the original MPLP. Class: `double` Default: `[]`
`S.recover.th`	Matrix of the affine map $x = \mathrm{Y}y + \mathrm{th} \begin{pmatrix} \theta \\ 1 \end{pmatrix}$. The map is from the optimization variables involed in the reduced MPLP $y(\theta) \mapsto x$ to the original MPLP. Class: `double` Default: `[]` Default: `[]`

OUTPUT

`R`	result structure Class: `struct`
`R.xopt`	Optimal solution Class: `PolyUnion`
`R.mplpsol`	Structure with the solution as returned by MPLP solver. Class: `struct`
`R.mplpsol.Pn`	Array of polytopes in MPT2 format. Class: `polytope`
`R.mplpsol.Fi`	Cell array of matrices of the control law given as $x = F_i\theta + G_i$. Class: `cell`
`R.mplpsol.Gi`	Cell array of matrices of the control law given as $x = F_i\theta + G_i$.

	Class: `cell`
`R.mplpsol.activeConstraints`	Index set of active constraints Class: `cell`
`R.mplpsol.Phard`	Feasible domain. Class: `polytope`
`R.mplpsol.details`	More details about the solution and the computation. Class: `struct`

SEE ALSO

Opt, mpt_solvemp

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_CALL_CDD

A gateway function to CDD solver (without errorchecks)

SYNTAX

```
R = mpt_call_cdd(S)
```

DESCRIPTION

The function implements call to CDD solver based on formulation from `Opt` class. Only LP problems are supported.
It is assumed that LP entering this function is of the form

$$
\begin{aligned}
\min \quad & f^T x \\
\text{s.t.} \quad & lb \le x \le ub \\
& Ax \le b \\
& A_e x = b_e
\end{aligned}
$$

which is accepted by CDD directly. Two specific routines can be chosen to solve LP: criss-cross method or dual-simplex that are specified in "solver" field. Dual-simplex method is taken by default.

INPUT

S	structure of the Opt class Class: `struct`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \le b$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \le b$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.lb	Lower bound for the variables $x \ge$ lb. Class: `double` Default: `[]`

`S.ub`	Upper bound for the variables $x \leq \text{ub}$. Class: `double` Default: `[]`
`S.n`	Problem dimension (number of variables). Class: `double`
`S.m`	Number of inequalities in $Ax \leq b$. Class: `double`
`S.me`	Number of equalities in $A_e x = b_e$. Class: `double`
`S.problem_type`	A string specifying the problem to be solved. Class: `char`
`S.test`	Call (false) or not to call (true) MPT global settings. Class: `logical` Default: `false`
`S.solver`	Specify string to call "criss-cross" or "dual-simplex" method. By default, the method "dual-simplex" is used. Class: `char`

OUTPUT

`R`	result structure Class: `struct`
`R.xopt`	Optimal solution. Class: `double`
`R.obj`	Objective value. Class: `double`
`R.lambda`	Lagrangian multipliers Class: `double`
`R.exitflag`	An integer value that informs if the result was feasible (1), or otherwise (different from 1). Class: `double`
`R.how`	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization. Class: `char`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_CALL_GUROBI

A gateway function to GUROBI solver (without errorchecks)

SYNTAX

```
R = mpt_call_gurobi(S)
```

DESCRIPTION

The function implements call to GUROBI solver based on formulation from `Opt` class. QP, LP, MILP and MIQP problems are supported.
It is assumed that QP/LP/MIQP/MILP entering this function (for LP/MILP $H = 0$) is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & lb \leq x \leq ub \\
& Ax \leq b \\
& A_e x = b_e \\
& x \in \{C, I, B, N, S\}
\end{aligned}
$$

where the set $\{C, I, B, N, S\}$ represents

- C - continuous variables, $x \in (-\infty, \infty)$

- I - integer variables $x \in (\ldots, -1, 0, 1, \ldots)$

- B - binary variables $x \in \{0, 1\}$

- N - semi-integer variables (possibly bounded above) $x \in [0, 1, \overline{x} \leq \infty)$

- S - semi-continuous variables (possibly bounded above) $x \in [0, \overline{x} \leq \infty)$

which is given by strings in `vartype` field. GUROBI accepts this format directly, the only prerequisite is to transform input data to sparse format.

INPUT

S	Structure of the Opt class. Class: `struct`
S.H	Quadratic part of the objective function. Class: `double` Default: `[]`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \leq b$. Class: `double`

| S.b | Right hand side of the inequality constraints $Ax \leq b$. |
| | Class: `double` |

S.Ae	Linear part of the equality constraints $A_e x = b_e$.
	Class: `double`
	Default: `[]`

S.be	Right hand side of the equality constraints $A_e x = b_e$.
	Class: `double`
	Default: `[]`

S.lb	Lower bound for the variables $x \geq$ lb.
	Class: `double`
	Default: `[]`

S.ub	Upper bound for the variables $x \leq$ ub.
	Class: `double`
	Default: `[]`

| S.n | Problem dimension (number of variables). |
| | Class: `double` |

| S.m | Number of inequalities in $Ax \leq b$. |
| | Class: `double` |

| S.me | Number of equalities in $A_e x = b_e$ |
| | Class: `double` |

| S.problem_type | A string specifying the problem to be solved. |
| | Class: `char` |

| S.vartype | A string specifying the type of variable. Supported characters are C (continuous), I (integer), B (binary), N (semi-integer), S (semi-continuous). Example: First variable from three is binary, the rest is continuous: `S.vartype='BCC';` |
| | Class: `char` |

S.test	Call (false) or not to call (true) MPT global settings.
	Class: `logical`
	Default: `false`

OUTPUT

| R | result structure |
| | Class: `struct` |

| `R.xopt` | optimal solution |
| | Class: `double` |

| `R.obj` | Optimal objective value. |
| | Class: `double` |

| `R.lambda` | Lagrangian multipliers. |
| | Class: `double` |

| `R.exitflag` | An integer value that informs if the result was feasible (1), or otherwise (different from 1). |
| | Class: `double` |

| `R.how` | A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization. |
| | Class: `char` |

SEE ALSO

mpt_solve

LITERATURE

Stephen Boyd and Lieven Vandenberghe: Convex Optimization; Cambridge University Press

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_CALL_QUADPROG

A gateway function to QUADPROG solver (without errorchecks)

SYNTAX

```
R = mpt_call_quadprog(S)
```

DESCRIPTION

The function implements call to QUADPROG solver based on formulation from `Opt` class. Only QP and LP problems are supported.

It is assumed that QP/LP entering this function (for LP $H = 0$) is of the form

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + f^T x \\
\text{s.t.} \quad & \text{lb} \leq x \leq \text{ub} \\
& Ax \leq b \\
& A_e x = b_e
\end{aligned}
$$

which is passed to QUADPROG solver directly. For LP QUADPROG solver passes the data to LINPROG automatically.

INPUT

S	Structure of the `Opt` class Class: `struct`
S.H	Quadratic part of the objective function. Class: `double` Default: `[]`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \leq b$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.lb	Lower bound for the variables $x \geq \text{lb}$.

Class: `double`
Default: `[]`

S.ub   Upper bound for the variables $x \leq$ ub.
                     Class: `double`
                     Default: `[]`

S.n   Problem dimension (number of variables).
                     Class: `double`

S.m   Number of inequalities in $Ax \leq b$.
                     Class: `double`

S.me   Number of equalities in $A_e x = b_e$.
                     Class: `double`

S.problem_type   A string specifying the problem to be solved.
                     Class: `char`

S.test   Call (false) or not to call (true) MPT global settings.
                     Class: `logical`
                     Default: `false`

OUTPUT

R   result structure
                     Class: `struct`

R.xopt   Optimal solution.
                     Class: `double`

R.obj   Optimal objective value.
                     Class: `double`

R.lambda   Lagrangian multipliers.
                     Class: `double`

R.exitflag   An integer value that informs if the result was feasible
                     (1), or otherwise (different from 1).
                     Class: `double`

R.how   A string that informs if the result was feasible ('ok'), or
                     if any problem appeared through optimization.
                     Class: `char`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_CALL_GLPK

A gateway function to GLPK solver (without errorchecks)

SYNTAX

```
R = mpt_call_glpk(S)
```

DESCRIPTION

The function implements call to GLPK solver based on formulation from `Opt` class. Only LP, MILP and problems are supported.
It is assumed that LP/MILP entering this function is of the form

$$
\begin{aligned}
\min \quad & f^T x \\
\text{s.t.} \quad & \text{lb} \leq x \leq \text{ub} \\
& Ax \leq b \\
& A_e x = b_e \\
& x \in \{C, I, B\}
\end{aligned}
$$

where the set $\{C, I, B\}$ represents

- C - continuous variables, $x \in (-\infty, \infty)$

- I - integer variables $x \in (\ldots, -1, 0, 1, \ldots)$

- B - binary variables $x \in \{0, 1\}$

which is given by strings in `vartype` field. GLPK accepts this format directly.

INPUT

S	Structure of the Opt class. Class: `struct`
S.f	Linear part of the objective function. Class: `double`
S.A	Linear part of the inequality constraints $Ax \leq b$. Class: `double`
S.b	Right hand side of the inequality constraints $Ax \leq b$. Class: `double`
S.Ae	Linear part of the equality constraints $A_e x = b_e$. Class: `double` Default: `[]`
S.be	Right hand side of the equality constraints $A_e x = b_e$. Class: `double`

Default: []

S.lb   Lower bound for the variables $x \geq$ lb.
Class: double
Default: []

S.ub   Upper bound for the variables $x \leq$ ub.
Class: double
Default: []

S.n   Problem dimension (number of variables).
Class: double

S.m   Number of inequalities in $Ax \leq b$.
Class: double

S.me   Number of equalities in $A_e x = b_e$.
Class: double

S.problem_type   A string specifying the problem to be solved.
Class: char

S.vartype   A string specifying the type of variable. Supported characters are C (continuous), I (integer), B (binary). Example: First variable from three is binary, the rest is continuous: S.vartype='BCC';

Class: char

S.test   Call (false) or not to call (true) MPT global settings.
Class: logical
Default: false

OUTPUT

R   result structure
Class: struct

R.xopt   optimal solution
Class: double

R.obj   Optimal objective value.
Class: double

R.lambda   Lagrangian multipliers.
Class: double

R.exitflag   An integer value that informs if the result was feasible (1), or otherwise (different from 1).

Class: `double`

R.how   A string that informs if the result was feasible ('ok'), or
if any problem appeared through optimization.
Class: `char`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

ELIMINATEEQUATIONS

Transforms LP/QP/MPLP/MPQP to LPC/PLCP

SYNTAX

`problem.qp2lcp`

`qp2lcp(problem)`

DESCRIPTION

Transformation of LP, QP, MPLP, and MPQP to LCP/PLCP formulation. Consider the following MPQP format that is accepted by `Opt` class:

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + (F\theta + f)^T x \\
\text{s.t.} \quad & Ax \le b + B\theta \\
& A_e x = b_e + E\theta \\
& A_\theta \theta = b_\theta
\end{aligned}
$$

which contains m inequality constrains and m_e equality constraints and constraints on the parameter θ. If there are lower and upper bounds on the variables x present, i.e. `lb` and `ub`, these can be merged to inequalities (150). This format is not appropriate for transformation to LCP form because it contains equality constraints and the inequalities are not nonnegative. To get appropriate LCP representation, the equality constrains of the problem $(149) - (152)$ are removed using `eliminateEquations` method of the `Opt` class. The intermediate form of the optimization problem is given as

$$
\begin{aligned}
\min \quad & \frac{1}{2}{x_{Nc}}^T \tilde{H} x_{Nc} + (\tilde{F}\theta + \tilde{f})^T x_{Nc} \\
\text{s.t.} \quad & \tilde{A} x_{Nc} \le \tilde{b} + \tilde{B}\theta \\
& A_\theta \theta = b_\theta
\end{aligned}
$$

with x_{Nc} as the non-basic variables that map to x affinely.
Problem $(154) - (155)$ can be transformed effectively when considering the rank of the matrix $\tilde{A}$. If the rank of matrix $\tilde{A}$ is less than the number of inequalities in (155), then vector x_N can be expressed as a difference of two positive numbers, i.e.

$$
\begin{aligned}
x_{Nc} &= {x_{Nc}}^+ - {x_{Nc}}^- \\
{x_{Nc}}^+ &\ge 0 \\
{x_{Nc}}^- &\ge 0
\end{aligned}
$$

Using the substitution

$$
v = \begin{pmatrix} {x_{Nc}}^+ \\ {x_{Nc}}^- \end{pmatrix}
$$

and putting back to $(154) - (155)$ we get

$$
\begin{aligned}
\min \quad & \frac{1}{2} v^T \begin{pmatrix} \tilde{H} & -\tilde{H} \\ -\tilde{H} & \tilde{H} \end{pmatrix} v + \left(\begin{pmatrix} \tilde{F} & -\tilde{F} \end{pmatrix} \theta \begin{pmatrix} \tilde{f} & -\tilde{f} \end{pmatrix} \right)^T v \\
\text{s.t.} \quad & \begin{pmatrix} -\tilde{A} & \tilde{A} \end{pmatrix} v \geq -\tilde{b} - \tilde{B}\theta \\
& v \geq 0
\end{aligned}
$$

which is a form suitable for PLCP formulation.
If the rank of matrix $\tilde{A}$ is greater or equal than the number of inequalities in (155), we can factorize the matrix $\tilde{A}$ rowwise

$$
\tilde{A} = \begin{pmatrix} \tilde{A}_B \\ \tilde{A}_N \end{pmatrix}
$$

where B, N are index sets corresponding to rows from which submatrices are built. The factored system can be written as

$$
\begin{aligned}
-\tilde{A}_B x &= -\tilde{b}_B - \tilde{B}\theta + y \\
-\tilde{A}_N x &\geq -\tilde{b}_N - \tilde{B}\theta \\
y &\geq 0
\end{aligned}
$$

where the matrix $\tilde{A}_B$ form by rows in the set B must be invertible. Using this substitution the system $(154)(156)$ can be rewritten in variable y

$$
\begin{aligned}
\min \quad & \frac{1}{2} y^T \hat{H} y + (\hat{F}\theta + \hat{f})^T y \\
\text{s.t.} \quad & \hat{A} \geq \hat{b} + \hat{B}\theta \\
& y \geq 0
\end{aligned}
$$

which is suitable for PLCP formulation where

$$
\begin{aligned}
\hat{H} &= \tilde{A}_B^{-T} \tilde{H} \tilde{A}_B^{-1} \\
\hat{F} &= -(\tilde{A}_B^{-T} \tilde{H} \tilde{A}_B^{-1} \tilde{B}_B - \tilde{A}_B^{-T} \tilde{F} \\
\hat{f} &= -\tilde{A}_B^{-T} \tilde{H} \tilde{A}_B^{-1} b_B - \tilde{A}_B^{-T} \tilde{f} \\
\hat{A} &= \tilde{A}_N \tilde{A}_B^{-1} \\
\hat{b} &= \tilde{A}_N \tilde{A}_B^{-1} b_B - b_N \\
\hat{B} &= \tilde{A}_N \tilde{A}_B^{-1} B_B - b_N - B_N
\end{aligned}
$$

The corresponding PLCP can be written as follows:

$$
\begin{aligned}
w - Mz &= q + Q\theta \\
w &\geq 0 \\
z &\geq 0 \\
w^T z &= 0
\end{aligned}
$$

where the problem data are built from MPQP $(167) - (169)$

$$
M = \begin{pmatrix} \hat{H} & -\hat{A}^T \\ \hat{A} & 0 \end{pmatrix}
$$

$$
q = \begin{pmatrix} \hat{f} \\ -\hat{b} \end{pmatrix}
$$

$$
Q = \begin{pmatrix} \hat{F} \\ -\hat{B} \end{pmatrix}
$$

Original solution to MPQP problem $(149) - (151)$ can be obtained by affine map from the variables $w(\theta)$ and $z(\theta)$ to x. The matrices of the backward map are stored inside `recover` property of the `Opt` class as follows

$$
x = \mathrm{uX} \begin{pmatrix} w \\ z \end{pmatrix} + \mathrm{uTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}
$$

If the problem was formulated using YALMIP, it is possible that some variables are in the different order. The original order of variables is stored in `problem.varOrder.requested_variables` and the map to original variables is given by

$$
x = \mathrm{primalX} \begin{pmatrix} w \\ z \end{pmatrix} + \mathrm{primalTh} \begin{pmatrix} \theta \\ 1 \end{pmatrix}
$$

INPUT

`problem`	LP/QP/MPLP/MPQP optimization problem given as `Opt` class. Class: `Opt`

EXAMPLE(s)

Example 1

Consider MPQP in the following form

$$
\begin{aligned}
\min_{x} \quad & x^2 \\
\mathrm{s.t.} \quad & \theta_1 - \theta_2 - x \le 0.2 \\
& -0.5\theta_1 + \theta_2 - x \le 0.4 \\
& -1 \le x \le 1 \\
& 0 \le \theta_1 \le 1 \\
& 0 \le \theta_2 \le 1
\end{aligned}
$$

with one decision variables x and two parameters θ_1, θ_2.
Construct MPQP optimization problem

```
H = 1;
```

```
A = [-1; -1]; b = [0.2; 0.4]; pB = [-1,1; 0.5,-1];

lb = -1; ub = 1;

Ath = [1 0;-1 0;0 1;0 -1]; bth = [1;1;1;1];

problem = Opt('H',H,'A',A,'b',b,'pB',pB,'lb',lb,'ub',ub,'Ath',Ath,'bth',bth)


--------------------------------------------------
Parametric quadratic program
Num variables:              1
Num inequality constraints: 2
Num equality constraints:   0
Num lower bounds            1
Num upper bounds            1
Num parameters:             2
Solver:                     PLCP
--------------------------------------------------
```

Transform `problem` to PLCP form

```
problem.qp2lcp
```

```
--------------------------------------------------
Parametric linear complementarity problem
Num variables:              7
Num inequality constraints: 0
Num equality constraints:   0
Num parameters:             2
Solver:                     PLCP
--------------------------------------------------
```

Solve the appropriate PLCP

```
solution = problem.solve
```

```
mpt_plcp: 3 regions

solution =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Plot the optimizer, for instance the variables z.

```
solution.xopt.fplot('z')
```
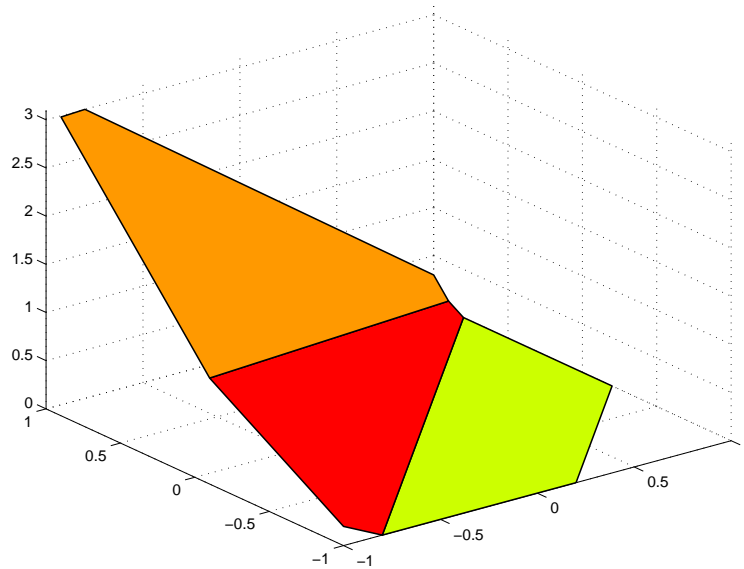
SEE ALSO

solve, mpt_call_lcp

LITERATURE

Stephen Boyd and Lieven Vandenberghe: Convex Optimization; Cambridge University Press
Richard W. Cottle, Jong-Shi Pang, Richard E. Stone: Linear complementarity problem,
Academic Press Inc. 1992

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

ELIMINATEEQUATIONS

Reduce LP/QP/MPLP/MPQP by removing equality constraints

SYNTAX

```
problem.eliminateEquations
```

```
eliminateEquations(problem)
```

DESCRIPTION

Remove equality constraints involved in LP, QP, MPLP, and MPQP of the dimension n to get optimization problem in the dimension $n - m_e$ where m_e stands for the number of equalities. Consider the following MPQP

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + (F\theta + f)^T x \\
\text{s.t.} \quad & Ax \le b + B\theta \\
& A_e x = b_e + E\theta
\end{aligned}
$$

which contains m inequality constrains and m_e equality constraints.

To be able to reduce the optimization problem to a simpler form, it is required that the system of linear equations $A_e x = b_e + E\theta$ is consistent, i.e. no linearly dependent rows are found and the number of equalities m_e is strictly less than number of variables n, i.e. $m_e < n$. The principle is based on factorizing equality constraints $A_e x = b_e + E\theta$ in basic x_{Bc} and non-basic variables x_{Nc}, i.e.

$$
A_e = \begin{pmatrix} A_{e,Bc} & A_{e,Nc} \end{pmatrix}
$$

which gives

$$
A_{e,Bc} x_{Bc} + A_{e,Nc} x_{Nc} = b_e + E\theta
$$

where the index sets `Bc`, `Nc` denote the columns from which factored system is built. The factored submatrix $A_{e,Bc}$ must be invertible in order to express basic variables as a function of non-basic variables, i.e.

$$
x_{Bc} = -A_{e,Bc}^{-1} A_{e,Nc} x_{Nc} + A_{e,Bc}^{-1} b_e + A_{e,Bc}^{-1} E\theta
$$

Using the substitution

$$
C = -A_{e,Bc}^{-1} A_{e,Nc}
$$

and

$$
\begin{aligned}
D_1 &= A_{e,Bc}^{-1} b_e \\
D_2 &= A_{e,Bc}^{-1} E
\end{aligned}
$$

the relation between basic and non-basic variables is simplified to

$$
x_{Bc} = C x_{Nc} + D_1 + D_2\theta \tag{189}
$$

The above MPQP problem $(184) - (186)$ can be expressed only in non-basic variables x_{Nc} as follows:

$$
\begin{aligned}
\min \quad & \frac{1}{2}{x_{Nc}}^T \tilde{H} x_{Nc} + (\tilde{F}\theta + \tilde{f})^T x_{Nc} \\
\text{s.t.} \quad & \tilde{A} x_{Nc} \le \tilde{b} + \tilde{B}\theta
\end{aligned}
$$

where

$$
\begin{aligned}
\tilde{H} &= C^T H_{Bc,Bc} C + C^T H_{Bc,Nc} + H_{Nc,Bc} C + H_{Nc,Nc} \\
\tilde{F} &= 0.5(C^T H_{Bc,Bc} D_2 + C^T {H_{Bc,Bc}}^T D_2 + {H_{Bc,Nc}}^T D_2 + H_{Nc,Bc} D_2) + C^T F_{Bc} + F_{Nc} \\
\tilde{f} &= 0.5(C^T H_{Bc,Bc} D_1 + C^T {H_{Bc,Bc}}^T D_1 + {H_{Bc,Nc}}^T D_1 + H_{Nc,Bc} D_1) + C^T f_{Bc} + f_{Nc} \\
\tilde{A} &= A_{Bc} C + A_{Nc} \\
\tilde{b} &= b - A_{Bc} D_1 \\
\tilde{B} &= B - A_{Bc} D_2
\end{aligned}
$$

Original solution to LP/QP problem $(184) - (186)$ can be obtained via relation (). The matrices of the backward map are stored inside `recover` property of the `Opt` class as follows

$$
x_{Bc} = \mathrm{Y} x_{Nc} + \mathrm{th}\begin{pmatrix} \theta \\ 1 \end{pmatrix}
$$

where the matrix `Y` corresponds to C and the matrix `th` to D in (). Note the the reduced problem $(187) - (188)$ has different cost function as the original problem.

INPUT

problem   LP/QP/MPLP/MPQP optimization problem given as
            `Opt` class.
            Class: `Opt`

EXAMPLE(s)

Example 1

Consider LP optimization problem with the following data

```
f = [-1 1 2];
```

```
A = [1 -1 0.4; 0.5 -9 -2; 0.5 -1 0; -5.1 -1 -3];
```

```
b = [1; 2; 2; 4];
```

```
Ae = [1 0 0]; be = 1;
```

Create the problem

```
problem = Opt('A',A,'b',b,'f',f,'Ae',Ae,'be',be)
```

```
--------------------------------------------------
Linear program
Num variables:               3
Num inequality constraints:  4
Num equality constraints:    1
Solver:                      LCP
--------------------------------------------------
```

The problem has equality constaint $x_1 = 1$ which can be verified by solving the problem.

```
r = problem.solve
```

```
r =

xopt: [3x1 double]
lambda: [1x1 struct]
obj: -6.884
how: 'ok'
exitflag: 1
```

```
r.xopt
```

```
ans =

1
0.548
-3.216
```

We can eliminate the equality constaint by calling

```
problem.eliminateEquations
```

```
--------------------------------------------------
Linear program
Num variables:               2
Num inequality constraints:  4
Num equality constraints:    0
Solver:                      LCP
--------------------------------------------------
```

The solution and the objective value is different

```
rn = problem.solve
```

```
rn =

xopt: [2x1 double]
lambda: [1x1 struct]
obj: -6.884
how: 'ok'
exitflag: 1
```

```
rn.xopt
```

```
ans =

-3.216
0.548
```

The mapping between the variables in the old and new problem are stored inside `recover` property.

```
problem.recover
```

```
ans =

Y: [3x2 double]
th: [3x1 double]
```

To get the original solution, use

```
xold = problem.recover.Y*rn.xopt + problem.recover.th
```

```
xold =

1
0.548
-3.216
```

We can see that the solution is the same

```
r.xopt -xold
```

```
ans =

0
0
0
```

SEE ALSO

solve

LITERATURE

Stephen Boyd and Lieven Vandenberghe: Convex Optimization; Cambridge University Press
Richard W. Cottle, Jong-Shi Pang, Richard E. Stone: Linear complementarity problem,
Academic Press Inc. 1992

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

DISPLAY

Overload display for `Opt` class.

SYNTAX

```
display(problem)
```

```
problem.display()
```

DESCRIPTION

Default display for `Opt` class.

INPUT

problem   Object of the `Opt` class that defines optimization problem.
Class: `Opt`

SEE ALSO

mpt_solve

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
    mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

FEASIBLESET

Computes the feasible set of a given parametric problem.

SYNTAX

```
K = prob.feasibleSet()

K = prob.feasibleSet('method')

K = prob.feasibleSet(regions)
```

DESCRIPTION

Method for computing a feasible set of a given parametric problem. For the following formulation of a parametric problem

$$\min \ J(z, x)$$
$$\text{s.t.} \ Az \le b + Bx$$
$$A_e z = b_e + Ex$$

the feasible set K is the polyhedron

$$K = \{x \mid \exists \, z \text{ s.t. } Az \le b + Bx, A_e z = b_e + Ex\}$$

This method implements two procedures to compute K:

1. if K=prob.feasibleSet('method') is called, the feasible set is calculated by a given 'method' used in projection (can be expensive)

2. if K=prob.feasibleSet(regions) is called with "regions" being the critical regions of the parametric solution, then K is constructed as follows. For each facet of each region do:

   (a) compute the center of the facet

   (b) take a small step accross the facet

   (c) solve the problem for the new point

   (d) if the problem is infeasible, add the facet to the feasible set

INPUT

method	A string describing which method of projection to use for computing the feasible set. For more help of the methods type `help Polyhedron/projection`. Class: `char`
regions	A polyhedron array for which to describe the feasible set. The polyhedra must be in the same dimension. Class: `Polyhedron`

OUTPUT

   K   The computed feasible set for the given parametric optimization problem.
Class: `Polyhedron`

EXAMPLE(s)

Example 1

Create parametric LP with 3 decision variables and 2 parameters
The cost function $(F\theta + f)^T x$

```
f = [1; -1; 0.5]; pF = [1 -1.2; -0.8 -2.1; 3.1 0];
```

The linear inequality constraints $Ax \leq b + B\theta$

```
A = [-11.15 -16.28 -8.09; 6.85 16.17 11.61; 10.37 2.64  5.92; 18.22 -11.28 2.42;
-5.29 -5.6 2.4];
```

```
b = [ 1.43; 3.49; 3.98; 2.2; 2.23];
```

```
B = [ -0.11 0.5; -0.89 -0.83; 0.74 0.2; 1.39 -1.01; 2.47 -0.62];
```

Provide bounds on the parameters $A_\theta \theta \leq b_\theta$

```
Ath = [eye(2); -eye(2)]; bth = [2;2;5;5];
```

Create the parametric optimization problem

```
problem = Opt('f',f,'A',A,'b',b,'pB',B,'pF',pF,'Ath',Ath,'bth',bth)
```

```
-------------------------------------------------
Parametric linear program
Num variables:                3
Num inequality constraints:   5
Num equality constraints:     0
Num parameters:               2
Solver:                       PLCP
-------------------------------------------------
```

We want to compute the feasible set of the parametric optimization problem.

```
K = problem.feasibleSet()
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities   4 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : none
```

The feasible set represents a region where there exist a feasible set of constraints. Now compute the solution by solving the parametric optimization problem:

```
res = problem.solve

mpt_plcp: 8 regions

res =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

The parametric solver returns regions with the optimal solution to the parametric optimization problem. The union of these regions is a subset of the feasible set K because in some parts of the feasible space the cost function can be unbounded and therefore the optimality conditions do not hold.

On the figure below one can see that in this example the union of regions returned from the parametric solver is a subset of the feasible set K.

```
K.plot('wire',true,'linewidth',2,'linestyle','--');
hold on;
res.xopt.plot
```


SEE ALSO

Opt
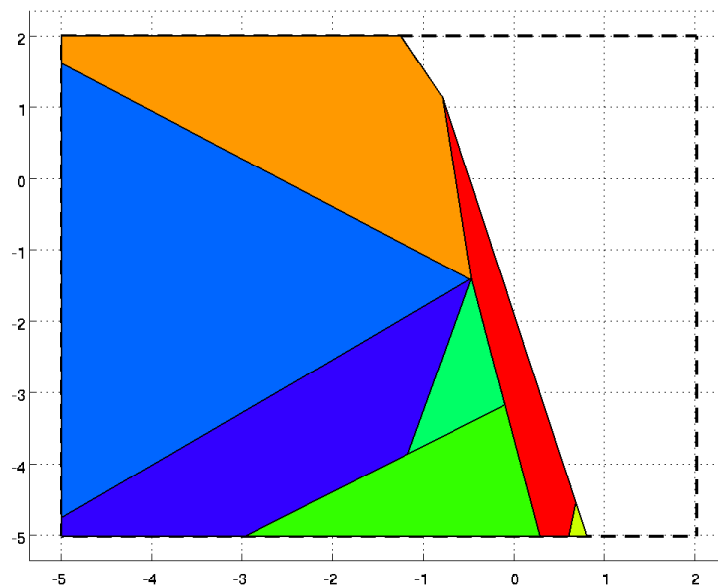
AUTHOR(s)

COPY

Creates a copy of the optimization problem given as `Opt` object.

SYNTAX

```
new_problem = problem.copy
```

```
new_problem = copy(problem)
```

DESCRIPTION

Creates a copy `new_problem` of the given optimization `problem`. The new object is an exact copy of the optimization problem. Changing the original `problem` does not affect the `new_problem`.

INPUT

problem	Object of the `Opt` class that defines optimization problem.
	Class: `Opt`

OUTPUT

new_problem	Object of the `Opt` that is an exact copy of `problem`.
	Class: `Opt`

EXAMPLE(s)

Example 1

Create LP optimization problem as min $x_1 - x_2$, s.t. $x_1 + 2x_2 <= 3, -x_1 - 0.3x_2 <= 1, -3x_1 + x_2 <= 4$.
Get the data of the optimization problem written as matrices.
The cost function $f^T x$

```
f = [1; -1];
```

The linear inequality constraints $Ax <= b$

```
A = [1 2;-1 -0.3;-3 1]; b = [3; 1; 4];
```

Create the optimization problem

```
problem = Opt('f',f,'A',A,'b',b)
```

```
--------------------------------------------------
Linear program
Num variables:            2
Num inequality constraints:  3
```

473

```
Num equality constraints:    0
Solver:                    LCP
-------------------------------------------------
```

We want to have a new copy of `problem`.

`LPproblem = problem.copy`

```
-------------------------------------------------
Linear program
Num variables:             2
Num inequality constraints:   3
Num equality constraints:    0
Solver:                    LCP
-------------------------------------------------
```

The new `LPproblem` is exactly the same as `problem`.
If we transform `problem` to LCP, its representation will change

`problem.qp2lcp`

```
-------------------------------------------------
Linear complementarity problem
Num variables:             3
Num inequality constraints:   0
Num equality constraints:    0
Solver:                    LCP
-------------------------------------------------
```

But we keep the original problem stored as `LPproblem`.

SEE ALSO

Opt

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

OPT

Interface for solving optimization problems

SYNTAX

```
problem = Opt('H',H,'f',f,'pF',F,'A',A,'b',b,'Ae',Ae,...)

problem = Opt('M',M,'q',q,'Q',Q,'Ath',Ath,'bth',bth)

problem = Opt(Matrices)

problem = Opt(constraints, objective, theta, x)
```

DESCRIPTION

Encapsulates data and solutions for LP/QP/MPLP/MPQP/LCP/PLCP problems. The data can be provided directly, or using YALMIP format or MPT2 format.
The following general MPQP format that is considered by `Opt` class:

$$
\begin{aligned}
\min \quad & \frac{1}{2}x^T H x + (F\theta + f)^T x + \theta^T Y\theta + C\theta + c \\
\text{s.t.} \quad & Ax \leq b + B\theta \\
& A_e x = b_e + E\theta \\
& A_\theta \theta = b_\theta
\end{aligned}
$$

which contains n optimization variables x, d number of parameters y, m inequality constrains, m_e equality constraints and constraints on the parameter θ. `Opt` class accepts also PLCP formulation of the form

$$
\begin{aligned}
w - Mz &= q + Q\theta \\
w &\geq 0 \\
z &\geq 0 \\
w^T z &= 0
\end{aligned}
$$

where w and z are the optimization variables and θ is the parameter.
The output format of the optimization problem depends on the input data. For instance, non-parametric LCP can be created by providing M, q data and the parametric LPC by supplying M, q, and Q.
At the time of construction, the problem data are validated and checked for proper dimensions. For all the methods involved in `Opt` class see `methods('Opt')`.

INPUT

H	Quadratic part of the objective function.
	Class: `double`
	Default: `[]`
f	Linear part of the objective function.

Class: `double`

pF   Linear part of the objective function for parameters.
              Class: `double`
              Default: `[]`

Y   Quadratic part of the objective function for parameters.
              Class: `double`
              Default: `0`

C   Linear part of the objective function for parameters.
              Class: `double`
              Default: `0`

c   Constant term in the objective function
              Class: `double`
              Default: `0`

A   Linear part of the inequality constraints $Ax \leq b + B\theta$.
              Class: `double`

b   Right hand side of the inequality constraints $Ax \leq b + B\theta$.
              Class: `double`

pB   Right hand side of the inequality constraints for parameters $Ax \leq b + B\theta$.
              Class: `double`

Ae   Linear part of the equality constraints $A_e x = b_e + E\theta$.
              Class: `double`
              Default: `[]`

be   Right hand side of the equality constraints $A_e x = b_e + E\theta$.
              Class: `double`
              Default: `[]`

pE   Right hand side of the equality constraints for parameters $A_e x = b_e + E\theta$.
              Class: `double`
              Default: `[]`

lb   Lower bound for the decision variables $x \geq$ lb.
              Class: `double`
              Default: `[]`

ub   Upper bound for the decision variables $x \leq$ ub.
              Class: `double`

Default: []

Ath	Linear part of the inequality constraints $A_\theta \theta \leq b_\theta$. Class: double Default: []
bth	Right hand side of the inequality constraints $A_\theta \theta \leq b_\theta$. Class: double Default: []
vartype	A vector of strings determining the type of the variable. Supported characters are C (continuous), I (integer), B (binary), N (semi-integer), S (semi-continuous). Example: First variable from three is binary, the rest is continuous: vartype='BCC'; Class: char Default: ''
solver	S string specifying which solver should be called. Class: char Default: []
M	Linear matrix involved in LCP. Class: double Default: []
q	Right hand side vector involved in LCP. Class: double Default: []
Q	Linear matrix involved in parametric formulation of LCP. Class: double Default: []
Matrices	Structure with the matrices defining MPLP/M-PQP problem as returned by mpt_constructMatrices function. For detailed description, see help mpt_constructMatrices. Class: struct
constraints	Yalmip set of constraints that formulate the given problem. Class: lmi
objective	Yalmip variable that represents objective value of the given problem. Class: sdpvar

477

theta	Specification of parametric variables involved in problem formulated in Yalmip. Class: `sdpvar`
x	Specification of decision variables involved in problem formulated in Yalmip. Class: `sdpvar`

OUTPUT

problem	Object of the `Opt` class that defines the optimization problem. Class: `Opt`

EXAMPLE(s)

Example 1

Formulate LP problem min $f^T x$, s.t. $Ax \leq b$.

```
f = [1 2 3]; A = randn(8,3); b = ones(8,1);
```

Consruct the object

```
problem = Opt('f',f,'A',A,'b',b)
```

```
--------------------------------------------------
Linear program
Num variables:               3
Num inequality constraints:  8
Num equality constraints:    0
Solver:                      LCP
--------------------------------------------------
```

Solve the problem

```
problem.solve
```

```
ans =

xopt: [3x1 double]
lambda: [1x1 struct]
obj: -2.25934252307113
how: 'ok'
exitflag: 1
```

Example 2

Formulate MPQP min $0.5x^T H x + f^T x$, s.t. $Ax \le b + B\theta$.

```
f = [-1 2 -3 4]; A = randn(12,4); b = ones(12,1); B = randn(12,2);
```

Restrict the parameters to a unitbox of the size 5.

```
Ath = [eye(2); -eye(2)]; bth = 5*ones(4,1);
```

Construct the object

```
problem = Opt('H',eye(4),'f',f,'A',A,'b',b,'pB',B,'Ath',Ath,'bth',bth)
```

```
-------------------------------------------------
Parametric quadratic program
Num variables:              4
Num inequality constraints:  12
Num equality constraints:    0
Num parameters:              2
Solver:                     PLCP
-------------------------------------------------
```

Solve the problem

```
solution = problem.solve
```

```
regions:   27,unexplored: 9
mpt_plcp: 43 regions
Fixing the adjacency list...
...done.

solution =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Since we have an explicit solution, we can plot the partition

```
solution.xopt.plot
```
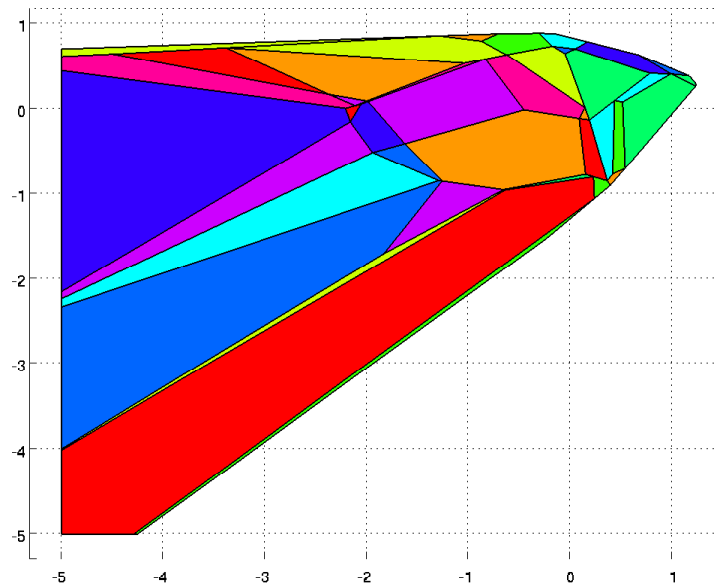
Example 3

Formulate LCP problem

```
H = randn(5); M = sqrt(2)*H'*H; q = randn(5,1);
```

Construct the object

```
problem = Opt('M',M,'q',q);
```

Solve the problem

```
problem.solve
```

```
ans =

xopt: [5x1 double]
lambda: [5x1 double]
obj: []
how: 'ok'
exitflag: 1
```

Example 4

Formulate MPC parametric problem using YALMIP
Define the model $x_{k+1} = Ax_k + Buk$.

```
A = 0.7*randn(2); B = randn(2,1);
```

Define the optimization variables

```
u = sdpvar(repmat(1,1,5),repmat(1,1,5));
```

```
x = sdpvar(repmat(2,1,6),repmat(1,1,6));
```

Define the constraints and the cost function.

```
objective = []; constraints = [];
```

```
for k = 1:5,
objective = objective + norm(eye(2)*x{k},2) + norm(u{k},2);
constraints = [constraints,x{k+1} == A*x{k} + B*u{k}];
constraints = [constraints,-5 <= u{k} <= 5,-10 <= x{k} <= 10];
end
```

Create the object

```
problem = Opt(constraints,objective,x{1},u{1})
```

```
--------------------------------------------------
Parametric linear program
Num variables:                15
Num inequality constraints:   30
Num equality constraints:     10
Num lower bounds              15
Num upper bounds              15
Num parameters:                2
Solver:                      PLCP
--------------------------------------------------
```

Solve the problem explicitly

```
solution = problem.solve
```

```
mpt_plcp: 4 regions
```

```
solution =
```

```
xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Plot the explicit solution

```
solution.xopt.plot
```
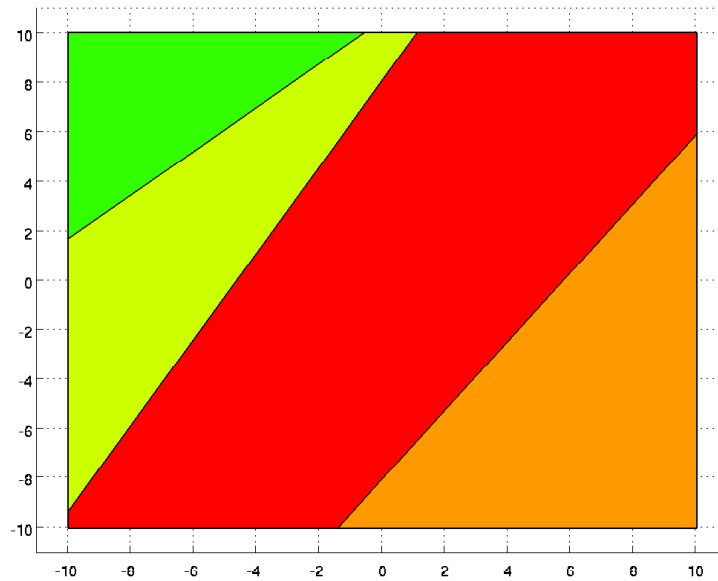
Example 5

Formulate MPC problem using MPT2

```
sysStruct.A = 0.7*randn(2);

sysStruct.B = randn(2,1);

sysStruct.C = [1,-1];

sysStruct.D = 0;

sysStruct.xmin = [-10;-10];

sysStruct.xmax = [10;10];

sysStruct.umin = -5;

sysStruct.umax = 5;

probStruct.Q = 2*eye(2);

probStruct.R = 1;

probStruct.subopt_lev = 0;

probStruct.norm = 2;

probStruct.N = 5;
```

Create problem data to MPQP

```
Matrices = mpt_constructMatrices(sysStruct,probStruct)
```

```
Function mpt_constructMatrices is obsolete and will be removed in a future MPT version.
Iteration 1...
Iteration 2...
Iteration 3...

Matrices =

PbndIncluded: 1
G: [36x5 double]
W: [36x1 double]
E: [36x2 double]
H: [5x5 double]
F: [2x5 double]
Y: [2x2 double]
Cf: [0 0 0 0 0]
Cx: [0 0]
Cc: 0
symmetric: 0
bndA: [4x2 double]
bndb: [4x1 double]
Pinvset: [1x1 polytope]
constraints_reduced: 1
```

Construct object of the `Opt` class.

```
problem = Opt(Matrices)
```

```
--------------------------------------------------
Parametric quadratic program
Num variables:                  5
Num inequality constraints:  36
Num equality constraints:     0
Num parameters:               2
Solver:                    PLCP
--------------------------------------------------
```

Solve the problem

```
solution = problem.solve
```

```
mpt_plcp: 7 regions

solution =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Plot the control law over the explicit solution

```
solution.xopt.fplot('primal')
```
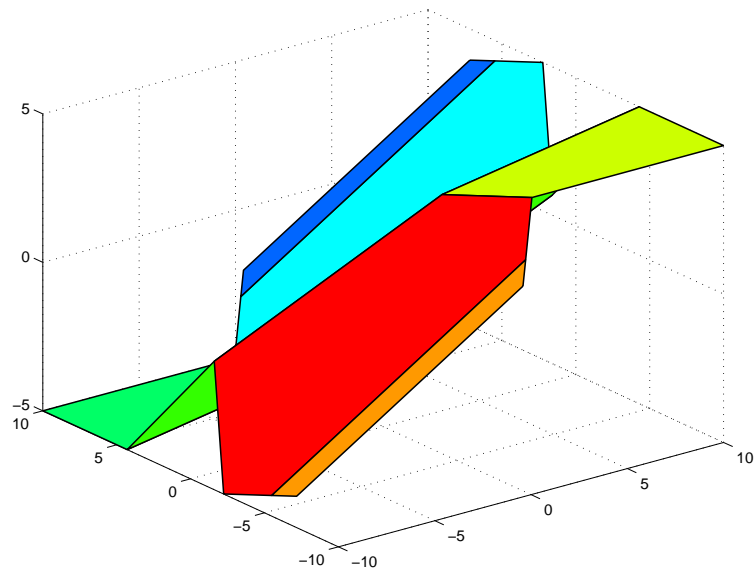

AUTHOR(s)

© **2010-2013** Colin Neil Jones: EPF Lausanne
   mailto:colin.jones@epfl.ch

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

SOLVE

The main routine for solving optimization problems

SYNTAX

```
result = problem.solve
```

```
result = problem.solve(th)
```

```
result = solve(problem)
```

DESCRIPTION

The main routine for solving non-parametric (LP, QP, MILP, MIQP) and parametric problems (MPLP, MPQP, PLCP). The result is returned in the appropriate format depending on the problem.

The `Opt` class serves as general wrapper for preprocessing the data involved in optimization, including necessary error checks. Once the data are valid, then are passed to `mpt_solve` or `mpt_solvemp` function that calls the appropriate solver without any errorchecks.

For parametric problems it is possible to solve the problem for a particular value of the parameters θ if provided as an argument `th`.

INPUT

problem	Object of the `Opt` class that defines the optimization problem to be solved. Class: `Opt`

OUTPUT

result	Structure with the date that represents the solution to given problem. For non-parametric problems the solution is returned with the following fields. Class: `struct`
result.xopt	Optimal solution for primal variables. Class: `double`
result.obj	Objective value. Class: `double`
result.lambda	Lagrangian multipliers Class: `double`
result.exitflag	An integer value that informs if the result was feasible (1), or otherwise (different from 1) Class: `double`

result.how	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization Class: `char`
result	Structure with the date that represents the solution to given problem. For parametric problems the solution is returned with the following fields. Class: `struct`
result.xopt	Optimal solution for variables w, z from PLCP reformulation if the problem was solved using PLCP solver. If the original problem was given as MPLP/MPQP, then this field returns also primal, dual variables, and the objective value for given problem. The solution is given as a collection of polyhedra in the same dimension with certain properties and is given as `PolyUnion` class. The function data associated to each variable is stored under `Function`, in particular in `res.Set(i).Funcj` where the index i corresponds to `i`-th region and index j to `j`-th function. Class: `PolyUnion`
result.exitflag	An integer value that informs if the result was feasible (1), or otherwise (different from 1) Class: `double`
result.how	A string that informs if the result was feasible ('ok'), or if any problem appeared through optimization Class: `char`
result.stats	Other details from the computation that might be of interest, such as the number of pivots, elapsed time, etc. Class: `struct`

EXAMPLE(s)

Example 1

Solve an LP stored in file `sc50b`.
Load the data

```
load sc50b
```

Create the optimization problem using `Opt` class

```
problem = Opt('A',A,'b',b,'Ae',Aeq,'be',beq,'lb',lb,'f',f)
```

```
-------------------------------------------------
Linear program
```

```
Num variables:           48
Num inequality constraints:  30
Num equality constraints:    20
Num lower bounds             48
Solver:                  LCP
--------------------------------------------------
```

Solve the problem

```
result = problem.solve
```

```
result =

xopt: [48x1 double]
lambda: [1x1 struct]
obj: -69.9999999999999
how: 'ok'
exitflag: 1
```

Example 2

Create random PLCP problem and solve it
Generate data

```
H = randn(5); M = sqrt(2)*H'*H; q=randn(5,1); Q = randn(5,2);
```

Provide bounds on the parameters

```
Ath = randn(8,2); bth = 5*ones(8,1);
```

Construct the problem

```
problem = Opt('M',M,'q',q,'Q',Q,'Ath',Ath,'bth',bth);
```

Solve the problem

```
result = problem.solve
```

```
mpt_plcp: 10 regions
```

```
result =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

We can plot the solution

```
result.xopt.plot
```
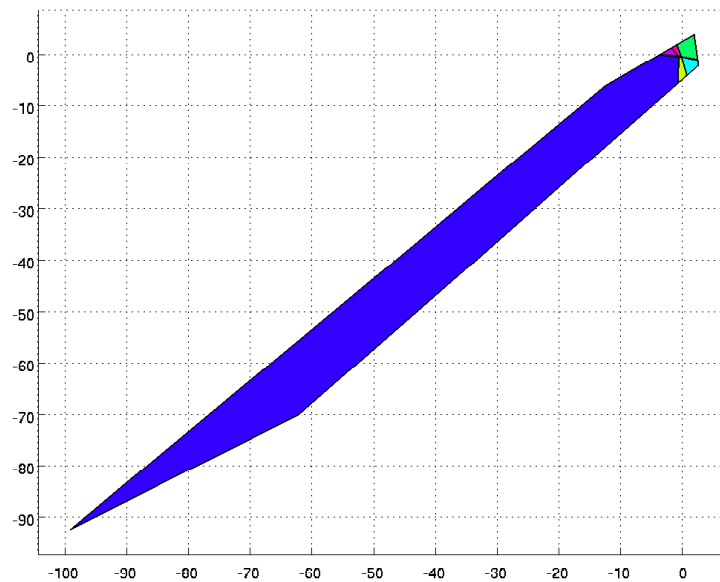

Example 3

Formulate linear parametric problem $\min x_1 - 2x_2\theta + \theta \text{ s.t. } x_1 - x_2 \geq \theta, x_1 \geq 0, x_2 \geq 0, -1 \leq \theta \leq 1$ with the help of YALMIP.

```
sdpvar x1 x2 th
```

Objective function:

```
obj = x1-2*x2*th+th;
```

Constraints:

```
F = [ x1-x2 >= th; x1>=0; x2>=0; -1<= th <= 1];
```

Construct an instance of `Opt` class that represents parametric linear program

```
problem=Opt(F,obj,th,[x1;x2])
```

```
--------------------------------------------------
Parametric linear program
Num variables:              2
Num inequality constraints: 5
Num equality constraints:   0
Num lower bounds:           2
Num upper bounds:           2
Num parameters:             1
Solver:                     PLCP
--------------------------------------------------
```

Solve the above problem quickly for the point `th=0.5` without generating explicit solution

```
problem.solve(0.5)
```

```
ans =

xopt: [2x1 double]
lambda: [1x1 struct]
obj: 1
how: 'ok'
exitflag: 1
```

Verify this result by solving the problem parametrically

```
res=problem.solve;
```

```
mpt_plcp: 2 regions
```

Plot the optimal cost function

```
res.xopt.fplot('obj','linewidth',3);
```
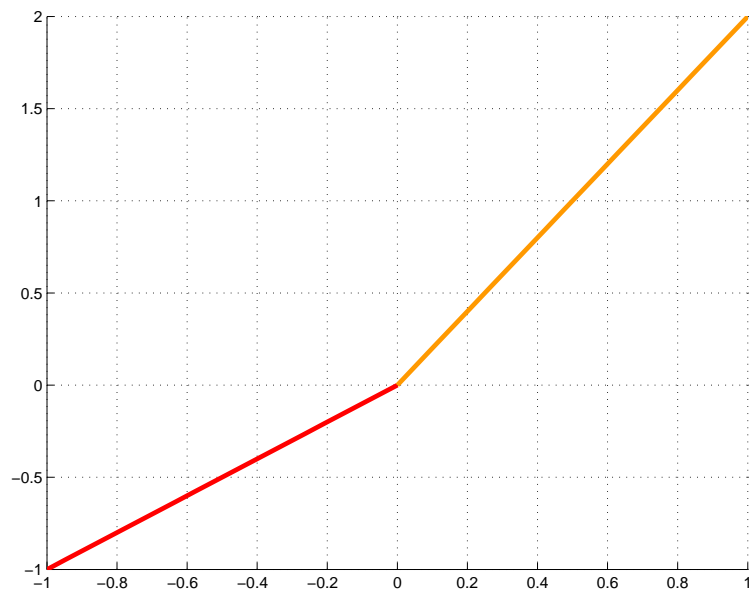

Evaluate the explicit solution for the point `th=0.5`

```
res.xopt.feval(0.5,'obj')
```

```
ans =

1
```

SEE ALSO

Opt, mpt_solve, mpt_solvemp

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_IMPORT

Converts sysStruct and probStruct into an MPT3 prediction model

SYNTAX

```
model = mpt_import ( sysStruct , probStruct )
```

DESCRIPTION

`model = mpt_import(sysStruct, probStruct)` converts MPT2-styled control problem, described by `sysStruct` and `probStruct` structures, into an MPT3 prediction model. The model can then be used for synthesis of MPC controllers via `controller = MPCController(model)`.

The following list shows which fields of `sysStruct` and `probStruct` can be automatically converted and what are their respective equivalents in MPT3:

- `sysStruct.xmin: model.x.min`

- `sysStruct.xmax: model.x.max`

- `sysStruct.umin: model.u.min`

- `sysStruct.umax: model.u.max`

- `sysStruct.ymin: model.y.min`

- `sysStruct.ymax: model.y.max`

- `probStruct.Q: model.x.penalty`

- `probStruct.R: model.u.penalty`

- `probStruct.Qy: model.y.penalty`

- `probStruct.P_N: model.x.terminalPenalty`

- `probStruct.Tset: model.x.terminalSet`

- `probStruct.xref: model.x.reference`

- `probStruct.uref: model.u.reference`

- `probStruct.yref: model.y.reference`

- `probStruct.tracking:` currently only `tracking=0` is supported

- `probStruct.Nc: model.u.block`

- `probStruct.sxmax: model.x.softMin` and `model.x.softMax`

- `probStruct.sumax: model.u.softMin` and `model.u.softMax`

- `probStruct.symax: model.y.softMin` and `model.y.softMax`

- `probStruct.subopt_lev:` currently only `subopt_lev=0` is supported. Use `ctrl = EMinTimeController(model)` to obtain a minimum-time controller

INPUT

sysStruct   System structure
             Class: struct

probStruct   Problem structure
             Class: struct

OUTPUT

model   Prediction model

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

EVALUATE

Returns the optimal control action

SYNTAX

```
u = controller.evaluate(x0)

[u, feasible] = controller.evaluate(x0)

[U, feasible, openloop] = controller.evaluate(x0)
```

DESCRIPTION

`u = controller.evaluate(x0)` solves the MPC optimization problem using `x0` as the initial condition and returns the first element of the optimal sequence of control inputs. If the problem is infeasible, `u` will be `NaN`.

`[u, feasible] = controller.evaluate(x0)` also returns the boolean feasibility flag.

`[u, feasible, openloop] = controller.evaluate(x0)` also returns the open-loop predictions of states, inputs and outputs in `openloop.X`, `openloop.Y`, and `openloop.Y`, respectively. Value of the optimal cost is returned in `openloop.cost`.

INPUT

x0   Initial state of the MPC optimization problem.
      Class: `double`

OUTPUT

u   Optimal control action.
          Class: `double`

feasible True if the optimization problem was feasible, false otherwise.
          Class: `logical`

openloop Structure containing open-loop predictions and value of the optimal cost.
          Class: `struct`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

493

MINTIMECONTROLLER

Implicit minimum-time MPC controller

SYNTAX

```
ctrl = MinTimeController(model)
```

DESCRIPTION

Constructs an object representing an implicit solution to the minimum-time MPC problem:

$$\min N$$
$$\text{s.t.} x(k+1) = f(x_k, u_k)$$
$$x(N) \in T_{set}$$
$$x(k) \in X$$
$$u(k) \in U$$

where $f(x_k, u_k)$ represents the prediction model's state-update equation, T_{set} is a given terminal set, and X and U denote, respectively, the state and input constraints.

LTI, PWA, and MLD models are accepted for predictions. State and input constraint sets X, U are automatically created using the model's state and input constraints (`model.x.min`, `model.x.max`, `model.u.min`, `model.u.max`). The terminal set T_{set} is either taken from `model.x.terminalSet` or, if the field is empty, by calling `model.stabizilingController()` (for LTI systems only).

Once computed, minimum-time controllers inherit the behavior of the `MPCController` class in terms of evaluation and post-processing.

INPUT

model   Any MPT3 system (`LTISystem`, `PWASystem`, `MLDSystem`)
       Class: `AbstractSystem`

OUTPUT

ctrl   Instance of the `MinTimeController` class.

EXAMPLE(s)

Example 1

Create a 1D LTI system

```
sys = LTISystem('A',1,'B',1,'C',1,'D',0)
```

`LTISystem with 1 state,1 input,1 output`

Define constraints

`sys.x.min = -1; sys.x.max = 1;`

`sys.u.min = -1; sys.u.max = 1;`

Define penalties (we use squared two-norm with unity weights here)

`sys.x.penalty = QuadFunction(1);`

`sys.u.penalty = QuadFunction(1);`

Construct the minimum-time controller

`ctrl = MinTimeController(sys)`

`Minimum-time controller (horizon: 1)`

Convert the controller to its explicit form

`expctrl = ctrl.toExplicit()`

```
Computing stabilizing terminal controller...
Iteration 1...
Iterating...
New horizon: 1
mpt_plcp: 1 regions
Explicit minimum-time controller (horizon: 1,regions: 2,partitions: 2)
```

SEE ALSO

EMinTimeControllerr, MPCController

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

EVALUATE

Returns the optimal control action

SYNTAX

```
u = controller.evaluate(x0)

[u, feasible] = controller.evaluate(x0)

[U, feasible, openloop] = controller.evaluate(x0)
```

DESCRIPTION

`u = controller.evaluate(x0)` solves the MPC optimization problem using `x0` as the initial condition and returns the first element of the optimal sequence of control inputs. If the problem is infeasible, `u` will be `NaN`.

`[u, feasible] = controller.evaluate(x0)` also returns the boolean feasibility flag.

`[u, feasible, openloop] = controller.evaluate(x0)` also returns the open-loop predictions of states, inputs and outputs in `openloop.X`, `openloop.Y`, and `openloop.Y`, respectively. Value of the optimal cost is returned in `openloop.cost`.

INPUT

x0   Initial state of the MPC optimization problem.
    Class: `double`

OUTPUT

u   Optimal control action.
       Class: `double`

feasible   True if the optimization problem was feasible, false otherwise.
       Class: `logical`

openloop   Structure containing open-loop predictions and value of the optimal cost.
       Class: `struct`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

TOEXPLICIT

Computes the explicit solution to the MPC optimization problem

SYNTAX

```
expctrl = controller.toExplicit()
```

DESCRIPTION

`expctrl = controller.toExplicit()` computes the explicit solution to the MPC optimization problem defined by the `controller` object and returns an instance of a class representing explicit MPC controllers.

The generated explicit controller inherits all methods and properties of the input object.

SEE ALSO

EMPCController

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

FROMYALMIP

Converts a custom YALMIP setup to a controller

SYNTAX

```
controller.fromYALMIP(yalmipdata)
```

DESCRIPTION

This function convers a custom YALMIP optimization problem to an MPT3 controller object.

INPUT

yalmipdata	Structure generated by `toYALMIP` Class: `struct`

SEE ALSO

toYALMIP

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

TOYALMIP

Converts an MPC problem into YALMIP's constraints and objective

SYNTAX

```
yalmipdata = controller.toYALMIP()
```

DESCRIPTION

This function convers an MPC optimization problem into YALMIP. The output structure contains following fields:

- `constraints`: contains constraints of the MPC problem as YALMIP's `lmi` object.

- `objective`: scalar variable of type `sdpvar` which defines the optimization objective.

- `variables`: structure containing variables of the optimization problem.

This method is MPT3 replacement of MPT2's `ownmpc` mechanism. In short, `toYALMIP` allows to modify the MPC problem by adding new constraints and/or by modifying the objective function.

OUTPUT

yalmipdata	Structure containing constraints, objective, and variables of the MPC problem. Class: `struct`

EXAMPLE(s)

Example 1

Create a 1D LTI system $x^+ = 0.9x + u$.

```
sys = LTISystem('A',0.9,'B',1);
```

Define an MPC controller.

```
ctrl = MPCController(sys);
```

Specify constraints, penalties, and the prediction horizon.

```
ctrl.model.x.min = -5; ctrl.model.x.max = 5;
ctrl.model.u.min = -5; ctrl.model.u.max = 5;
ctrl.model.x.penalty = QuadFunction(1);
ctrl.model.u.penalty = QuadFunction(1);
ctrl.N = 3;
```

Obtain the optimal sequence of inputs over the whole prediction horizon for the initial condition `x0=3`.

```
[~,~,openloop] = ctrl.evaluate(3); openloop.U
```

```
ans =
```

```
-1.57733887733888       -0.505197505197505       5.55111512312578e-17
```

Convert the problem to YALMIP and add custom state constraints on the first two predicted inputs via YALMIP.

```
Y = ctrl.toYALMIP()
Y.constraints = Y.constraints + [ -0.5 <= Y.variables.u(:,1) <= 0.5];
Y.constraints = Y.constraints + [ -0.8 <= Y.variables.u(:,2) <= 0.8];
ctrl.fromYALMIP(Y);
x0 = 3;
[~,~,openloop] = ctrl.evaluate(x0); openloop.U
```

```
Y =
```

```
constraints: [4 lmi]
objective: [1x1 sdpvar]
variables: [1x1 struct]
internal: [1x1 struct]
```

```
ans =
```

```
-0.5                    -0.8       -6.418622835482e-17
```

Notice that the updated controller respects constraints added via YALMIP.

SEE ALSO

fromYALMIP

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

MPCCONTROLLER

MPC controller with on-line optimization

SYNTAX

```
ctrl = MPCController(model)

ctrl = MPCController(model, horizon)
```

DESCRIPTION

`ctrl = MPCController(model)` constructs an MPC optimization problem by using the object `model` as the prediction model.

The basic optimization problem takes the following form:

$$
\min_{U} J(x_N) + \sum_{k=0}^{N-1} J(x_k, u_k, y_k)
$$
$$
\text{s.t.} \, x_{k+1} = f(x_k, u_k)
$$
$$
y_k = g(x_k, u_k)
$$
$$
x_{min} \leq x_k \leq x_{max}
$$
$$
u_{min} \leq u_k \leq u_{max}
$$
$$
y_{min} \leq y_k \leq y_{max}
$$

where N is the prediction horizon, x_k, u_k, y_k are the state, input, and output predictions, respectively, $J(x_N)$ denotes the terminal cost, $J(x_k, u_k, y_k)$ is the stage cost, $f(x_k, u_k)$ represents the state-update equation, and $g(x_k, u_k)$ is the output equation.

The terminal cost is given by $J(x_N) = \|Qx_N\|_p$, where Q is the weighting matrix and p is the type of the norm (if $p = 2$, then $J(x_N) = x_N^T Q x_N$, see "`help Function`" for more information). The type of the norm as well as the weighting matrix are specified by the `terminalPenalty` property of the state signal (see "`help SystemSignal/filter_terminalPenalty`").

The stage cost is represented by $J(x_k, u_k, y_k) = \|Q_x x_k\|_{p_x} + \|Q_u u_k\|_{p_u} + \|Q_y y_k\|_{p_y}$, where Q_i are weighting matrices and p_i represent types of the norms. Properties of the state penalization are taken from `model.x.penalty` (see "`help SystemSignal/filter_penalty`"). Similarly, properties of input and output penalizations are extracted from `model.u.penalty` and `model.y.penalty`, respectively. Note that different norms/weights can be assigned to various signals. Hence you can have a quadratic penalization of the state, but use a 1-norm penalty on the inputs.

If a certain signal (state, input, and/or output) has the `reference` filter enabled (see "`help SystemSignal/filter_reference`"), the terminal and stage costs become $J(x_N) = \|Q(x_N - x_{ref})\|_p$ and $J(x_k, u_k, y_k) = \|Q_x(x_k - x_{ref}\|_{p_x} + \|Q_u(u_k - u_{ref})\|_{p_u} + \|Q_y(y_k - y_{ref})\|_{p_y}$. If some signal does not have the `reference` property enabled, we assume a zero reference instead.

By default, min/max bounds on state, input, and output variables are added to the MPC setup. The bounds stored in `model.x.min`, `model.x.max`, `model.u.min`, `model.u.max`, `model.y.min`, and `model.y.max` are used. Note that other types of constraints can be added. These include polyhedral constraints (see "`help SystemSignal/filter_setConstraint`"), soft constraints (see "`help SystemSignal/filter_softMax`" and "`help SystemSignal/filter_softMin`"), or move blocking (see "`help SystemSignal/filter_block`").

The basic way to create an MPC controller is to call

```
controller = MPCController(system)
```

where `system` represents the dynamical system to be used for predictions (can be an instance of the `LTISystem`, `PWASystem`, or `MLDSystem` classes). Then you can further fine-tune the prediction model by adding constraints and/or modifying penalties. This can be done by accessing the `model` property of the controller object:

```
controller.model.x.min = [-5; -5]
```

You can also specify the prediction horizon directly from the command line:

```
controller.N = 3
```

Once the controller is fully specified, you can use the `evaluate` method to obtain the optimal sequence of inputs, (see "`help MPCController/evaluate`"), or compute the explicit form of the controller by calling the `toExplicit` method (see "`help MPCController/toExplicit`").

INPUT

model	Any MPT3 system (`LTISystem`, `PWASystem`, `MLDSystem`) Class: `AbstractSystem`
N	Prediction horizon Class: `Double`

OUTPUT

controller   Instance of the `MPCController` class.

EXAMPLE(s)

Example 1

Create a 1D LTI system $x^+ = 0.9x + u$

```
sys = LTISystem('A',0.9,'B',1)
```

LTISystem with 1 state,1 input,0 outputs

Construct the MPC controller

```
ctrl = MPCController(sys)
```

MPC controller (no prediction horizon defined)

Add constraints

```
ctrl.model.x.min = -1; ctrl.model.x.max = 1;
ctrl.model.u.min = -1; ctrl.model.u.max = 1;
```

Add penalties (we use squared two-norm with unity weights here)

```
ctrl.model.x.penalty = QuadFunction(1);
ctrl.model.u.penalty = QuadFunction(1);
```

Specify the prediction horizon

```
ctrl.N = 4;
```

Compute the optimal control action for the initial state x0=0.5

```
x0 = 0.5;
u = ctrl.evaluate(x0)
```

u =

-0.268049612898345

We can also ask for the full open-loop predictions:

```
[u,feasible,openloop] = ctrl.evaluate(x0)
```

u =

-0.268049612898345

feasible =

1

openloop =

cost: 0.370622325804255
U: [-0.268049612898345 -0.0956658064407663 -0.0306402938778255 -3.46944695195361e-18]
X: [0.5 0.181950387101655 0.0680895419507233 0.0306402938778255 0.0275762644900429]
Y: [0x4 double]

Convert the controller to an explicit form

expctrl = ctrl.toExplicit()

mpt_plcp: 1 regions
Explicit MPC controller (horizon: 4,regions: 1)

SEE ALSO

EMPCController

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

MPT_MPSOL2PU

Converts a parametric solution generated by solvemp to a PolyUnion object

SYNTAX

```
pu = mpt_mpsol2pu(sol)
```

DESCRIPTION

`pu = mpt_mpsol2pu(sol)` converts a solution obtained by YALMIP's `solvemp()` function to a `PolyUnion` object. This allows to easily convert `solvemp` solutions into explicit MPC controllers by `ctrl = EMPCController(mpt_mpsol2pu(sol))`.

This function produces an instance of the `PolyUnion` class whose polyhedral partition is taken from `sol.Pn`. The primal optimizer (given by `sol.Fi` and `sol.Gi`) is assigned to the `primal` function of the polyunion, while the cost (represented by `sol.Ai`, `sol.Bi`, and `sol.Ci`) is converted to the `obj` function of the union.

Note that, if the input solution contains overlapping partitions, then an array of polyunions will be returned. Each element of such an array then represents the corresponding overlapping partition of the input solution. If the objective function is piecewise affine, you can subsequently remove the overlaps by calling `pu = pu.min('obj')`.

INPUT

  sol   Solution obtained by `solvemp`
        Class: `cell`

OUTPUT

  pu   Parametric optimizer represented as an instance of the
        `PolyUnion` class
        Class: `PolyUnion`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

DELTAMAX

Upper bound on the increment of a signal

SYNTAX

DESCRIPTION

This filter introduces the upper bound ub on the difference of a signal $\Delta s_k \leq$ ub. The signal difference is defined as $\Delta s_k = s_k - s_{k-1}$ where s is the system signal and k is the time instant. The filter is activated by calling `s.with('deltaMax')`.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

SETCONSTRAINT

Adds a polyhedral constraint

SYNTAX

DESCRIPTION

This filter adds polyhedral constraints $z_k \in P$ for all $k = 0, \ldots, N-1$ to the MPC setup. The filter can be applied to arbitrary signals, be it state, input, or output variables.

Note, however, that adding this filter to a state variable will NOT add the constraint on the final predicted state (only quantities on prediction steps $k = 0, \ldots, N-1$ are affected by this filter). To add a polyhedral terminal state constraint, use the `terminalSet` filter (see "`help SystemSignal/filter_terminalSet`").

To enable the filter, first use `model.x.with('setConstraint')` (you can replace the `x` signal by any other signal of the prediction model).

The polyhedron P of the new constraint $x_k \in P$ can then be specified in the `setConstraint` property of the signal:

`model.x.setConstraint = Polyhedron(...)`

To remove this filter, call `model.x.without('setConstraint')`.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

TERMINALSET

Adds a polyhedral constraint on the final predicted state

SYNTAX

DESCRIPTION

This filter adds a polyhedral constraint on the final predicted state, i.e., $x_N \in P$, where x_N is the final predicted state and P is a polyhedron of suitable dimension.

To enable the filter, first call `model.x.with('terminalSet')`. Then you can provide the polyhedron in the `model.x.terminalSet` property.

To remove this filter, call `model.x.without('terminalSet')`.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

SOFTMIN

Soft lower bound constraint

SYNTAX

DESCRIPTION

This filter will soften the lower bound constraint on a selected signal in the MPC setup. Without this filter, lower bounds are hard, i.e., the signal has to satisfy $z_{min} \leq z_k$. With this filter added, the lower bound is soft and can be violated by some positive margin d_k, i.e., the new constraint becomes $z_{min} - d_k \leq z_k$. The slack variables d_k are then penalized in the MPC cost function by adding the term $\|Qd_k\|_p$.

To enable this filter, first call `model.signal.with('softMin')`, where you replace `signal` by the actual system's signal (typically by `x` for state variables, `u` for inputs, and `y` for outputs).

With the filter enabled, you can set the maximal allowed violation of the constraint in `model.signal.softMin.maximalViolation`, and specify penalization of the slack variables by setting `model.signal.softMin.penalty`.

To remove this filter, call `model.signal.without('softMin')`.

EXAMPLE(s)

Example 1

Specify an LTI system $x^+ = 0.9x + u$, $y = x$

```
sys = LTISystem('A',0.9,'B',1,'C',1);
```

Add unity quadratic penalties on states and inputs (but not on the terminal state)

```
sys.x.penalty = QuadFunction(1);
sys.u.penalty = QuadFunction(1);
```

Define hard state constraints

```
sys.x.min = -1; sys.x.max = 1;
```

Make the lower bound soft with maximal violation of 3

```
sys.x.with('softMin');
sys.x.softMin.maximalViolation = 3;
M = MPCController(sys,4);
x0 = -3.5;
[u,feasible,openloop] = M.evaluate(x0)
```

```
u =

2.15


feasible =

1


openloop =

cost: 25018.3457016632
U: [2.15 0.525779625779547 0.168399168399664 7.04103442217274e-13]
X: [-3.5 -0.999999999999999 -0.374220374220451 -0.168399168398741 -0.151559251558165]
Y: [-3.5 -1 -0.37422037422045 -0.168399168398741]
```

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

SYSTEMSIGNAL

Class representing variables of a dynamical system

SYNTAX

```
signal = SystemSignal(n)
```

DESCRIPTION

The `SystemSignal` class represents prediction of variables (e.g., system states, inputs, or outputs) of a dynamical system in an MPC setup.

Signals are created by `signal = SystemSignal(n)`, where `n` is the dimension of the variable. Once constructed, signals can be assigned names by `signal.name = 'myname'`. Each signal has following properties available for read/write access by default:

- `signal.min`: lower bound on the variable in MPC problems

- `signal.max`: upper bound on the variable in MPC problems

- `signal.penalty`: penalization of the variable in the MPC cost function given as `Function` object

Many additional properties can be set by the concept of *filters*. In short, filters are dynamical properties which can be added to a signal on demand. A filter is added by calling `signal.with('filter_name')`. List of available filters can be obtained by calling `methods SystemSignal` and looking for methods prefixed by the `filter_` string. Filters can be removed by calling `signal.without('filter_name')`. To list filters added to a particular signal, use the `signal.listFilters()` method.

INPUT

n   Dimension of the variable
      Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

MIN

Lower bound on a signal

SYNTAX

DESCRIPTION

This filter introduces the lower bound lb on the signal s, i.e. $\text{lb} \leq s_k$.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

REFERENCE

Penalizes difference of a signal from a given reference level

SYNTAX

DESCRIPTION

Adding this filter to an MPC setup will modify the objective function in such a way that the difference between actual signal and a prescribed reference signal is minimized.

To enable this filter, call `model.x.with('reference')`, which will enable tracking of state references by minimizing $\|Q(x - x_{ref})\|_p$. You can also add tracking to input and/or output signals by using `model.u.with('reference')` and `model.y.with('reference')`, respectively.

Once the filter is enabled, the reference trajectory can be specified in the signal's `reference` property:

```
model.x.reference = [0.5; -1]
```

The reference signal can also be time-varying. In such a case the k-th column of the reference is interpreted as the reference to be used at the k-th step of the prediction:

```
model.u.reference = [-1 -2 0 0 1]
```

The filter can be removed by calling `model.x.without('reference')`.

EXAMPLE(s)

Example 1

Create a 1D LTI system $x^+ = 0.9x + u$.

```
model = LTISystem('A',0.9,'B',1);
```

Enable tracking of state and input references.

```
model.x.with('reference');
model.u.with('reference');
model.x.reference = 0.4;
model.u.reference = 1;
model.x.penalty = QuadFunction(1);
model.u.penalty = QuadFunction(1);
```

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

BINARY

Constraints variable to be binary (0/1)

SYNTAX

DESCRIPTION

Adding this filter will constraint some (or all) elements of a given variable to take only binary values. To enable the filter, use `model.signal.with('binary')`.

To impose binary on all elements of a given variable (say, `model.u`), use `model.u.binary = true`. To add binary only to elements indexed by `idx`, call `model.u.binary = idx`. To mark all elements of `model.u` as real variables, use `model.u.binary = []`.

To remove this filter, call `model.signal.without('binary')`, in which case all elements of `signal` will be considered as real-valued variables.

EXAMPLE(s)

Example 1

Create a 1D LTI system $x^+ = 0.9x + u$.

```
model = LTISystem('A',0.9,'B',1);
```

Add constraints and penalties

```
model.x.min = -4; model.x.max = 4;
model.x.penalty = OneNormFunction(1);
model.u.penalty = OneNormFunction(0.1);
```

Create an MPC controller with prediction horizon 3

```
M = MPCController(model,3);
```

Constraint the control action to be binary and compute the open-loop optimizer for x0=-4

```
M.model.u.with('binary');
M.model.u.binary = true;
[~,~,openloop] = M.evaluate(-4);
openloop.U
```

```
ans =

1     1     0
```

AUTHOR(s)

BLOCK

Adds a move blocking constraint

SYNTAX

DESCRIPTION

Adding this filter to an MPC setup will modify the constraints in such a way that the differences between several consecutive optimization variables are equal to zero.

In the most common scenario, adding this filter to a signal representing control inputs will add a move-blocking constraint, which is equivalent to setting a control horizon.

To enable this filter, call `model.u.with('block')` (note that you can add this constrain to any type of signals, e.g., to state and output signals as well).

Once the filter is enabled, parameters of the blocking scheme can be specified in the `model.u.block.from` and `model.u.block.to` parameters. Setting these values to non-zero integers will add the constraint $u_{from} = u_{from+1} = \cdots = u_{to-1} = u_{to}$.

The filter can be removed by calling `model.x.without('block')`.

EXAMPLE(s)

Example 1

Create a 1D LTI system $x^+ = 0.9x + u$.

```
model = LTISystem('A',0.9,'B',1);
```

Add constraints and penalties

```
model.x.min = -4; model.x.max = 4;
model.u.min = -1; model.u.max = 1;
model.x.penalty = QuadFunction(1);
model.u.penalty = QuadFunction(1);
```

Create an MPC controller with prediction horizon 10

```
M = MPCController(model,5);
```

Tell the controller that only the first two control moves are free, the rest are to fixed

```
M.model.u.with('block');
M.model.u.block.from = 2;
M.model.u.block.to = 5;
x0 = 4;
[~,~,openloop] = M.evaluate(x0);
openloop.U
```

```
ans =

-1        -0.719434753448955        -0.719434753448955        -0.719434753448955        -0.719434753448955
```

AUTHOR(s)

MAX

Upper bound on a signal

SYNTAX

DESCRIPTION

This filter introduces the upper bound ub on the signal s, i.e. $s_k \leq$ ub.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

DELTAPENALTY

Penalizes the increment of a signal

SYNTAX

DESCRIPTION

This filter penalizes the difference of a signal defined as $\Delta s_k = s_k - s_{k-1}$ where s is the system signal and k is the time instant. The filter is activated by calling `s.with('deltaPenalty')`.

Any signal can have this filter, e.g. we can without problems enable slew-rate constraints on states, outputs, or even on binary variables. Non-state signals, however, require that the previous value is specified when calling `MPCController/evaluate`, e.g. `ctrl.evaluate(x0, 'u.previous')`.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

DELTAMIN

Lower bound on the increment of a signal

SYNTAX

DESCRIPTION

This filter introduces the lower bound lb on the difference of a signal $lb \leq \Delta s_k$. The signal difference is defined as $\Delta s_k = s_k - s_{k-1}$ where s is the system signal and k is the time instant. The filter is activated by calling `s.with('deltaMin')`.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

PENALTY

Penalizes the signal in the MPC cost function

SYNTAX

DESCRIPTION

This filter, which is enabled by default, allows to penalize a particular signal in the MPC cost function. The penalty function can be specified by setting the `model.signal.penalty` property to an instance of the `Function` class (see "`help Function`" for more information). Penalization of state variables is achieved by setting `model.x.penalty`, while `model.u.penalty` and `model.y.penalty` specify penalization of input and output variables, respectively.

Note that this filter only adds penalties to signals predicted at steps $k = 0, \ldots, N - 1$ of the prediction horizon. Therefore setting `model.x.penalty` will NOT penalize the final predicted state (i.e., x_N). To add a terminal state penalty, use the `terminalPenalty` filter (see "`help SystemSignal/filter_terminalPenalty`").

To disable penalization of a signal, set its `penalty` property to an empty matrix.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
      mailto:michal.kvasnica@stuba.sk

SOFTMAX

Soft upper bound constraint

SYNTAX

DESCRIPTION

This filter will soften the upper bound constraint on a selected signal in the MPC setup. Without this filter, upper bounds are hard, i.e., the signal has to satisfy $z_k \leq z_{max}$. With this filter added, the upper bound is soft and can be violated by some positive margin d_k, i.e., the new constraint becomes $z_k \leq z_{max} + d_k$. The slack variables d_k are then penalized in the MPC cost function by adding the term $\|Qd_k\|_p$.

To enable this filter, first call `model.signal.with('softMax')`, where you replace `signal` by the actual system's signal (typically by `x` for state variables, `u` for inputs, and `y` for outputs).

With the filter enabled, you can set the maximal allowed violation of the constraint in `model.signal.softMax.maximalViolation`, and specify penalization of the slack variables by setting `model.signal.softMax.penalty`.

To remove this filter, call `model.signal.without('softMax')`.

EXAMPLE(s)

Example 1

Specify an LTI system $x^+ = 0.9x + u$, $y = x$

```
sys = LTISystem('A',0.9,'B',1,'C',1);
```

Add unity quadratic penalties on states and inputs (but not on the terminal state)

```
sys.x.penalty = Penalty(1,2);
sys.u.penalty = Penalty(1,2);
```

```
============================================================
The Penalty object is deprecated.
Use obj.penalty = QuadFunction(Q) to define z'*Q*z penalty.
============================================================
============================================================
The Penalty object is deprecated.
Use obj.penalty = QuadFunction(Q) to define z'*Q*z penalty.
============================================================
```

Define hard state constraints

```
sys.x.min = -1; sys.x.max = 1;
```

Make the upper bound soft with maximal violation of 3

```
sys.x.with('softMax');
sys.x.softMax.maximalViolation = 3;
M = MPCController(sys,4);
x0 = 3.5;
[u,feasible,openloop] = M.evaluate(x0)



u =

-2.15


feasible =

1


openloop =

cost: 25018.3457016632
U: [-2.15 -0.525779625779547 -0.168399168399664 -7.04103442217274e-13]
X: [3.5 0.999999999999999 0.374220374220451 0.168399168398741 0.151559251558165]
Y: [3.5 1 0.37422037422045 0.168399168398741]
```

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

TERMINALPENALTY

Penalizes the final predicted state in the MPC problem

SYNTAX

DESCRIPTION

This filter adds the term $\|Qx_N\|_p$ to the MPC cost function. Properties of the penalty (i.e., the weighting matrix Q and the norm p) are provided by objects derived from the `Function` class.

Note that if the state signal has the `reference` filter enabled, the terminal penalty becomes $\|Q(x_N - x_{ref})\|_p$, where x_{ref} is a user-specified reference signal, taken from `model.x.reference`.

To add this filter, call `model.x.with('terminalPenalty')`. Then you can specify parameters of the penalty function by setting the `model.x.terminalPenalty` property to an instance of the `Function` class (see "`help Function`").

To remove this filter, call `model.x.without('terminalPenalty')`.

EXAMPLE(s)

Example 1

Specify an LTI system $x^+ = 0.9x + u$

```
sys = LTISystem('A',0.9,'B',1);
```

Add unity quadratic penalties on states and inputs (but not on the terminal state)

```
sys.x.penalty = QuadFunction(1);
sys.u.penalty = QuadFunction(1);
```

Compute an LQR terminal penalty and use it as a terminal state cost

```
sys.x.with('terminalPenalty');
sys.x.terminalPenalty = sys.LQRPenalty();
M = MPCController(sys,4);
x0 = 10;
u = M.evaluate(x0)
```

```
u =

-5.37666558531833
```

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

GETSTATES

Returns the internal state of the system

SYNTAX

```
x = system.getStates()
```

DESCRIPTION

This function returns the system's internal state.

OUTPUT

x   Values of the internal states
Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

UPDATE

Updates the internal state using the state-update equation

SYNTAX

```
system.update(u)

xn = system.update(u)

[xn, y] = system.update(u)
```

DESCRIPTION

This function evaluates the system's state-update equation and updates the internal state of the system.

By calling `system.update(u)` this function updates the internal state of `system` by evaluating the state-update equation. The updated state can then be retrieved by calling `x = system.getStates()`.

By calling `[xn, y] = system.update(u)` this function also returns the updated state as the first output, and the result of the output equation as the second output. Note that the internal system's state is still updated as described above.

INPUT

u   Vector of system's inputs
     Class: `double`

OUTPUT

xn   Updated state vector
     Class: `double`

y   Vector of outputs
     Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
     mailto:michal.kvasnica@stuba.sk

SIMULATE

Simulates evolution of the system

SYNTAX

```
data = system.simulate(U)
```

DESCRIPTION

`data = system.simulate(U)` computes evolution of system's states and outputs, starting from the system's internal state and using a sequence of inputs `U`.

Each column of `U` is interpreted as the control action to use at the m-th step of the simulation. The total number of simulation steps is given by the number of columns of `U`. To simulate an autonomous system over `M` steps, you need to define `U=zeros(0, M)`.

This function returns a structure `data`, which contains the simulated evolution of system's states (in `data.X`) and the outputs (in `data.Y`), respectively.

Note that you should always run `system.initialize(x0)` to set the initial condition prior to running the simulation.

Also note that the `simulate` method updates the internal system's state. Therefore once the function completes, the internal state will be set to the final value obtained at the end of the simulation.

INPUT

U   Matrix of control inputs stored column-wise for each
     simulation step.
     Class: `double`

OUTPUT

data   Structure with simulated state and output profiles.
        Class: `struct`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
     mailto:michal.kvasnica@stuba.sk

OUTPUT

Returns value of the output variables

SYNTAX

```
y = system.output()
```

```
y = system.output(u)
```

DESCRIPTION

This function evaluates the system's output equation and returns the calculated output. For systems with direct feed-through, it is necessary to provide the vector of inputs as the first input argument.

INPUT

u   Vector of system's inputs
Class: `double`

OUTPUT

y   Vector of outputs
Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

INITIALIZE

Sets the internal state of the system

SYNTAX

```
system.initialize(x0)
```

DESCRIPTION

`system.initialize(x0)` sets the initial state of the system to value `x0`.
The internal state can be retrieved by calling `system.getStates()`.

INPUT

x0   Initial state
       Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
     mailto:michal.kvasnica@stuba.sk

GETSTATES

Returns the internal state of the system

SYNTAX

```
x = system.getStates()
```

DESCRIPTION

This function returns the system's internal state.

OUTPUT

x   Values of the internal states
Class: `double`

EXAMPLE(s)

Example 1

Define a 1D LTI system $x^+ = x + u$.

```
sys = LTISystem('A',1,'B',1)
```

```
LTISystem with 1 state,1 input,0 outputs
```

Set the system's internal state to 2.5.

```
sys.initialize(2.5);
```

Retrieve the internal state.

```
x = sys.getStates()
```

```
x =

2.5
```

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

LQRPENALTY

Linear-quadratic regulator design for LTI systems

SYNTAX

```
P = system.LQRPenalty()
```

DESCRIPTION

`P = system.LQRPenalty()` calculates the solution of the discre-time Riccati equation.

Note that this function requires that penalties on states and inputs are defined in `system.x.penalty` and `system.u.penalty` (see "`help SystemSignal`" and "`help Function`").

OUTPUT

P   Solution of the discrete-time Riccati equation
     Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

LQRGAIN

Linear-quadratic regulator design for LTI systems

SYNTAX

```
K = system.LQRGain()
```

DESCRIPTION

`K = system.LQRGain()` calculates the optimal gain matrix `K` such that the state-feedback law $u = Kx$ minimizes the LQR cost function.

Note that this function requires that penalties on states and inputs are defined in `system.x.penalty` and `system.u.penalty` (see "`help SystemSignal`" and "`help Penalty`").

OUTPUT

K   Optimal gain matrix of the state-feedback law $u = Kx$
     Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

SETDOMAIN

Sets the domain of an LTI system

SYNTAX

```
sys.setDomain('x', P)
```

```
sys.setDomain('u', P)
```

```
sys.setDomain('xu', P)
```

DESCRIPTION

Sets the region of the state-input space where a given LTI system is valid.

This function sets the `domain` property of `LTISystem` objects to the polyhedron `P`, provided as the second input.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

UPDATE

Updates the internal state using the state-update equation

SYNTAX

```
system.update(u)

xn = system.update(u)

[xn, y] = system.update(u)
```

DESCRIPTION

This function evaluates the system's state-update equation and updates the internal state of the system.

By calling `system.update(u)` this function updates the internal state of `system` by evaluating the state-update equation. The updated state can then be retrieved by calling `x = system.getStates()`.

By calling `[xn, y] = system.update(u)` this function also returns the updated state as the first output, and the result of the output equation as the second output. Note that the internal system's state is still updated as described above.

INPUT

u   Vector of system's inputs
     Class: `double`

OUTPUT

xn   Updated state vector
     Class: `double`

y   Vector of outputs
     Class: `double`

EXAMPLE(s)

Example 1

Define a 1D LTI system $x^+ = 0.9x + u$, $y = 0.5x$

```
sys = LTISystem('A',0.9,'B',1,'C',0.5)
```

```
LTISystem with 1 state,1 input,1 output
```

Set the system's internal state to 1.

```
sys.initialize(1);
```

Update the system's state using $u = 0.2$.

```
sys.update(0.2);
```

Retrieve the internal state (should be equal to 1.1).

```
sys.getStates()
```

```
ans =

1.1
```

Update the system's state using $u = 0$.

```
sys.update(0)
```

```
ans =

0.99
```

Retrieve the internal state (should be equal to 0.99).

```
sys.getStates()
```

```
ans =

0.99
```

Update the internal state using $u = -1$ and retrieve the updated state and the associated output

```
[xn,y] = sys.update(-1)
```

```
xn =

-0.109
```

```
y =

0.495
```

AUTHOR(s)

SIMULATE

Simulates evolution of the system

SYNTAX

```
data = system.simulate(U)
```

DESCRIPTION

`data = system.simulate(U)` computes evolution of system's states and outputs, starting from the system's internal state and using a sequence of inputs `U`.

Each column of `U` is interpreted as the control action to use at the m-th step of the simulation. The total number of simulation steps is given by the number of columns of `U`. To simulate an autonomous system over M steps, you need to define `U=zeros(0, M)`.

This function returns a structure `data`, which contains the simulated evolution of system's states (in `data.X`) and the outputs (in `data.Y`), respectively.

Note that you should always run `system.initialize(x0)` to set the initial condition prior to running the simulation.

Also note that the `simulate` method updates the internal system's state. Therefore once the function completes, the internal state will be set to the final value obtained at the end of the simulation.

INPUT

U   Matrix of control inputs stored column-wise for each
    simulation step.
    Class: `double`

OUTPUT

data   Structure with simulated state and output profiles.
       Class: `struct`

EXAMPLE(s)

Example 1

Define a 1D LTI system $x^+ = 0.9x + u$.

```
sys = LTISystem('A',0.9,'B',1)
```

LTISystem with 1 state,1 input,0 outputs

Set the system's internal state to 2.5.

```
sys.initialize(2.5);
```

Simulate evolution of the system using the sequence of inputs U = [-0.5, 0.1, 2, 0, 0, 1].

```
U = [-0.5,0.1,2,0,0,1];
```

```
data = sys.simulate(U)
```

```
data =

X: [2.5 1.75 1.675 3.5075 3.15675 2.841075 3.5569675]
U: [-0.5 0.1 2 0 0 1]
Y: [0x6 double]
```

Plot the simulated evolution of system's states.

```
plot(data.X)
```
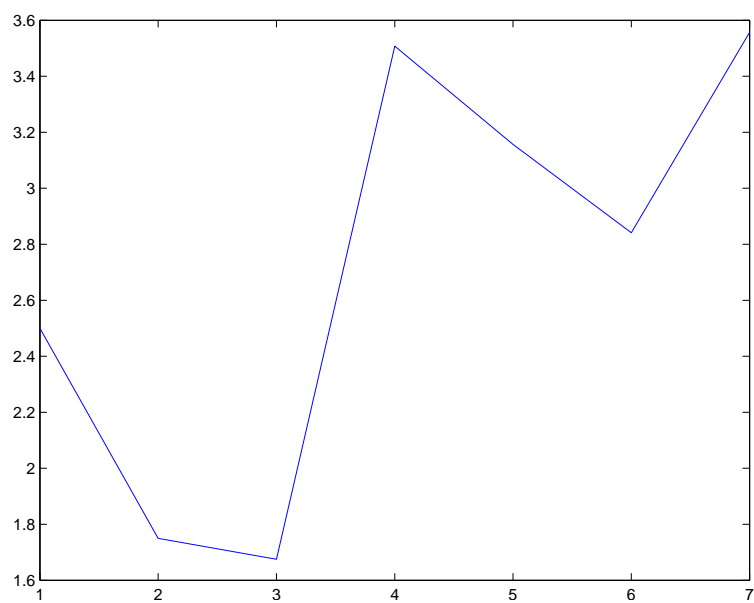

Note that at the end of simulation, the system's internal state is set to the final state value:

```
sys.getStates()
```

```
ans =

3.5569675
```

538

AUTHOR(s)

LTISYSTEM

Represents linear time-invariant systems

SYNTAX

```
sys = LTISystem('A', A, 'B', B, 'C', C, 'D', D, 'T_s', Ts)
```

DESCRIPTION

This class represents linear time-invariant systems of the form

$$x(t + Ts) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where $x \in \mathbb{R}^{n_x}$ is the state vector, $u \in \mathbb{R}^{n_u}$ is the vector of inputs, $y \in \mathbb{R}^{n_y}$ is the vector of outputs, and T_s represents the sampling time.

Each LTI system defines following properties:

- `A`, `B`: matrices of the state-update equation (read-only)

- `C`, `D`: matrices of the output equation (read-only)

- `Ts`: sampling time (read-only)

- `nx`, `nu`, `ny`: number of states, inputs and outputs (automatically determined, read-only)

- `x`: specifications of system's states (see `help SystemSignal`)

- `u`: specifications of system's inputs (see `help SystemSignal`)

- `y`: specifications of system's outputs (see `help SystemSignal`)

To define an LTI system, provide the list of system's matrices to the constructor:

```
sys = LTISystem('A', A, 'B', B, 'C', C, 'D', D, 'Ts', Ts)
```
All inputs, except of the `A` matrix, can be omitted. In such a case they are set to empty matrices of corresponding dimension. As a consequence, one can easily define autonomous systems $x(t + T_s) = Ax(t) + f$ by calling `sys = LTISystem('A', A, 'f', f, 'Ts', Ts)`. Similarly, to define an LTI system without outputs, call `sys = LTISystem('A', A, 'B', B, 'Ts', Ts)`. If the sampling time is omitted, it is set to `Ts=1`.

Another option to define an LTI system is to import the dynamics from Control toolbox' discre-time state-space objects:

```
sys = LTISystem(ss(A, B, C, D, Ts))
```
Important to remember is that LTI systems carry with them the value of the state vector. The initial value can be set by the `sys.initialize(x0)` method (see "`help LTISystem/initialize`"). Value of the internal state can be retrieved by the `sys.getStates()` method (see "`help LTISystem/getStates`"). To update the internal state using the system's state-update equation, use the `sys.update(u)` function (see "`help LTISystem/update`").

EXAMPLE(s)

Example 1

Create a 1D LTI system $x^+ = 2x + u$, $y = 3x$

```
sys = LTISystem('A',2,'B',1,'C',3)
```

LTISystem with 1 state,1 input,1 output

SEE ALSO

PWASystem

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

541

REACHABLESET

Computes forward or backwards reachable sets

SYNTAX

```
[S, SN] = system.reachableSet('X', X, 'U', U, 'N', N, '
    direction', ['forward'|'backwards'])
```

DESCRIPTION

`S = system.reachableSet('X', X, 'U', U, 'direction', 'forward')` computes the forward reachable set for an LTI system, i.e.,

$$S = \{Ax + Bu \mid x \in X, u \in U\}$$

`S = system.reachableSet('X', X, 'U', U, 'direction', 'backward')` computes the backward reachable set, i.e.,

$$S = \{x \mid Ax + Bu \in X, u \in U\}$$

If `X` and/or `U` are not given, the constraints are extracted from state and input constraints. If `direction` is not specified, `direction='forward'` is assumed.

This function supports autonomous systems as well.

INPUT

X	Polyhedron defining state constraints (optional) Class: `polyhedron`
U	Polyhedron defining input constraints (optional) Class: `polyhedron`
direction	Flag to switch between forward and backwards reachability. Class: `char`
N	Number of steps (defaults to 1). Class: `double`

OUTPUT

S	Set of states reachable in N steps. Class: `Polyhedron`
SN	Set of states reachable at each step. Class: `cell`

EXAMPLE(s)

Example 1

Define a 2D LTI system $x^+ = Ax + Bu$.

```
sys = LTISystem('A',[1 1; 0 1],'B',[1; 0.5]);
```

Specify domain of states and inputs

```
X = Polyhedron('lb',[-1; -1],'ub',[1; 1]);
U = Polyhedron('lb',-1,'ub',1);
```

Compute forward reachable set.

```
S = sys.reachableSet('X',X,'U',U,'direction','forward');
```

```
S.plot()
```
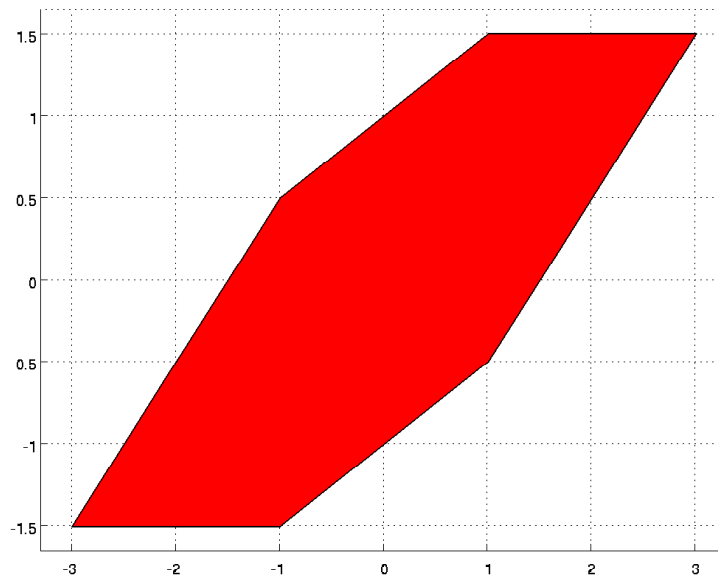

Example 2

Define a 2D LTI system $x^+ = Ax + Bu$.

```
sys = LTISystem('A',[1 1; 0 1],'B',[1; 0.5]);
```

Specify domain of states and inputs

```
X = Polyhedron('lb',[-1; -1],'ub',[1; 1]);
U = Polyhedron('lb',-1,'ub',1);
```

Compute the backward reachable set.

```
S = sys.reachableSet('X',X,'U',U,'direction','backward');

S.plot()
```


AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk
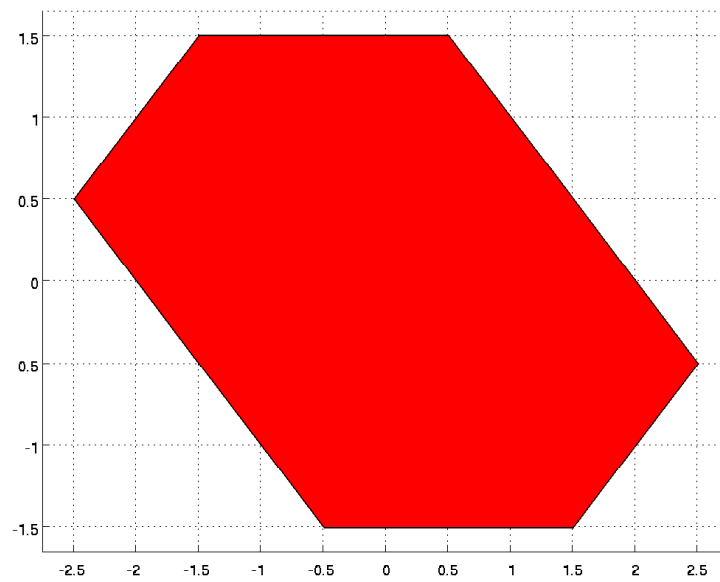
LQRSET

Computes an invariant subset of an LQR controller

SYNTAX

```
S = system.LQRSet()
```

DESCRIPTION

This function computes the set of states where an LQR controller $u = Kx$ provides recursive satisfaction of state and input constraints.

The set is computed by running the set recursion

$$S_{k+1} = \{x \mid (A + BK)x \in S_k, \ u_{\min} \le Kx \le u_{max}\},$$

initialized by $S_0 = \{x \mid x_{min} \le x \le x_{max}\}$ and terminating once $S_{k+1} = S_k$. The LQR feedback gain is computed automatically by calling `K = system.LQRGain()`.

Note that this function requires that input and state constraints are defined in `system.u.min`, `system.u.max`, `system.x.min`, and `system.x.max` (see "help SystemSignal").

OUTPUT

S   Invariant set of the LQR controller
     Class: `Polyhedron`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

INVARIANTSET

Computation of invariant sets for linear systems

SYNTAX

```
S = system.invariantSet()
```

```
S = system.invariantSet('X', X, 'U', U)
```

DESCRIPTION

For an autonomous LTI system $x^+ = Ax + f$ this function computes the set of states for which recursive satisfaction of state constraints can be shown.

The set is computed by running the set recursion

$$S_{k+1} = \{x \mid Ax + f \in S_k\},$$

initialized by $S_0 = X$ and terminating once $S_{k+1} = S_k$. If X is not provided, $X = \{x \mid x_{min} \leq x \leq x_{max}\}$ is assumed.

For an LTI system $x^+ = Ax + Bu$, which is subject to polyhedral state constraints $x \in X$ and input constraints $u \in U$ this function calculates the maximal control-invariant set

$$C = \{x \mid \exists u(k) \in U, \text{ s.t. } x(k) \in X, \ \forall k \geq 0\}.$$

Note that this function requires that state constraints defined in `system.x.min` and `system.x.max` (see "`help SystemSignal`").

INPUT

X	Polyhedron defining state constraints (optional) Class: `polyhedron`
U	Polyhedron defining input constraints (optional) Class: `polyhedron`
maxIterations	Maximal number of iterations (optional) Class: `double`

OUTPUT

S	Invariant set Class: `Polyhedron`

AUTHOR(s)

OUTPUT

Returns value of the output variables

SYNTAX

```
y = system.output()
```

```
y = system.output(u)
```

DESCRIPTION

This function evaluates the system's output equation and returns the calculated output. For systems with direct feed-through, it is necessary to provide the vector of inputs as the first input argument.

INPUT

u Vector of system's inputs
   Class: **double**

OUTPUT

y Vector of outputs
   Class: **double**

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

INITIALIZE

Sets the internal state of the system

SYNTAX

```
system.initialize(x0)
```

DESCRIPTION

system.initialize(x0) sets the initial state of the system to value x0.

The internal state can be retrieved by calling system.getStates().

INPUT

  x0   Initial state
        Class: `double`

EXAMPLE(s)

Example 1

Define a 1D LTI system $x^+ = x + u$.

```
sys = LTISystem('A',1,'B',1)
```

```
LTISystem with 1 state,1 input,0 outputs
```

Set the system's internal state to 2.5.

```
sys.initialize(2.5);
```

Retrieve the internal state.

```
x = sys.getStates()
```

```
x =

2.5
```

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

GETSTATES

Returns the internal state of the system

SYNTAX

```
x = system.getStates()
```

DESCRIPTION

This function returns the system's internal state.

OUTPUT

x   Values of the internal states
     Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
  mailto:michal.kvasnica@stuba.sk

PWASYSTEM

Represents discrete-time piecewise affine systems

SYNTAX

```
sys = PWASystem([lti1, lti2, ..., ltiM])
```

DESCRIPTION

This class represents PWA systems, which are composed of a finite number of local affine dynamics, each valid in a corresponding polyhedral region of the state-input space:

$$x(t + Ts) = A_i x(t) + B_i u(t) + f_i \text{ if } (x, u) \in R_i$$
$$y(t) = C_i x(t) + D_i u(t) + g_i$$

where $x \in \mathbb{R}^{n_x}$ is the state vector, $u \in \mathbb{R}^{n_u}$ is the vector of inputs, $y \in \mathbb{R}^{n_y}$ is the vector of outputs, T_s represents the sampling time, and R_i are the polyhedral regions of validity of the i-th local dynamics.

Each PWA system defines following properties:

- A, B, f: matrices of the state-update equation, stored as cell arrays (read-only)

- C, D, g: matrices of the output equation, stored as cell arrays (read-only)

- Ts: sampling time (read-only)

- domain: array of polyhedra denoting domain of the i-th local model (read-only)

- nx, nu, ny: number of states, inputs and outputs (automatically determined, read-only)

- ndyn: number of local models (read-only)

- x: specifications of system's states (see `help SystemSignal`)

- u: specifications of system's inputs (see `help SystemSignal`)

- y: specifications of system's outputs (see `help SystemSignal`)

- d: specifications of the binary dynamics selector signal (see `help SystemSignal`)

The preferred way to define a PWA system consisting of a finite number of local affine models is to provide the list of LTI models to the `PWASystem` constructor:

```
pwasys = PWASystem([ltisys1, ltisys2, ..., ltisysM])
```

Here, each LTI model must have its domain defined by the `ltisys.setDomain()` method (see "`help LTISystem/setDomain`").

EXAMPLE(s)

Example 1

We create a PWA system, consisting of two local affine models.
The first model is given by $x^+ = x + u$, and is valid for $x \geq 0$.

```
sys1 = LTISystem('A',1,'B',1);
```

```
sys1.setDomain('x',Polyhedron('lb',0));
```

The second model has dynamics $x^+ = -2x + 0.5u$, and is valid if $x \leq 0$.

```
sys2 = LTISystem('A',-2,'B',0.5);
```

```
sys2.setDomain('x',Polyhedron('ub',0));
```

Create the PWA system using `sys1` and `sys2`:

```
pwa = PWASystem([sys1,sys2])
```

```
State/input/output constraints not imported,set them manually afterwards.
PWASystem with 1 state,1 input,0 outputs,2 modes
```

SEE ALSO

LTISystem

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

TOLTI

Converts the i-th mode of a PWA system to an LTI system

SYNTAX

```
ltisys = pwasys.toLTI(index)
```

DESCRIPTION

For a PWA system consisting of M local affine models, this function converts the `index`-th model to an instance of the `LTISystem` class.

INPUT

index	Index of the local model to extract.
	Class: `double`

OUTPUT

ltisys	LTI representation of the i-th local model.
	Class: `LTISystem`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

UPDATE

Updates the internal state using the state-update equation

SYNTAX

```
system.update(u)
```

```
xn = system.update(u)
```

```
[xn, y] = system.update(u)
```

DESCRIPTION

This function evaluates the system's state-update equation and updates the internal state of the system.

By calling `system.update(u)` this function updates the internal state of `system` by evaluating the state-update equation. The updated state can then be retrieved by calling `x = system.getStates()`.

By calling `[xn, y] = system.update(u)` this function also returns the updated state as the first output, and the result of the output equation as the second output. Note that the internal system's state is still updated as described above.

INPUT

u   Vector of system's inputs
     Class: `double`

OUTPUT

xn   Updated state vector
      Class: `double`

y   Vector of outputs
      Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

SIMULATE

Simulates evolution of the system

SYNTAX

```
data = system.simulate(U)
```

DESCRIPTION

`data = system.simulate(U)` computes evolution of system's states and outputs, starting from the system's internal state and using a sequence of inputs `U`.

Each column of `U` is interpreted as the control action to use at the m-th step of the simulation. The total number of simulation steps is given by the number of columns of `U`. To simulate an autonomous system over `M` steps, you need to define `U=zeros(0, M)`.

This function returns a structure `data`, which contains the simulated evolution of system's states (in `data.X`) and the outputs (in `data.Y`), respectively.

Note that you should always run `system.initialize(x0)` to set the initial condition prior to running the simulation.

Also note that the `simulate` method updates the internal system's state. Therefore once the function completes, the internal state will be set to the final value obtained at the end of the simulation.

INPUT

U   Matrix of control inputs stored column-wise for each simulation step.
Class: `double`

OUTPUT

data   Structure with simulated state and output profiles.
Class: `struct`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

REACHABLESET

Computes forward or backwards reachable sets

SYNTAX

```
[S, SN] = system.reachableSet('X', X, 'U', U, 'N', N, '
    direction', ['forward'|'backwards'])
```

DESCRIPTION

`S = system.reachableSet('X', X, 'U', U, 'direction', 'forward')` computes the forward reachable set for a PWA system, i.e.,

$$S = \{f_P WA(x, u) \mid x \in X, u \in U\}$$

`S = system.reachableSet('X', X, 'U', U, 'direction', 'backward')` computes the backward reachable set, i.e.,

$$S = \{x \mid f_P WA(x, u) \in X, u \in U\}$$

If `X` and/or `U` are not given, the constraints are extracted from state and input constraints. If `direction` is not specified, `direction='forward'` is assumed.

This function supports autonomous systems as well.

INPUT

X	Polyhedron defining state constraints (optional) Class: `polyhedron`
U	Polyhedron defining input constraints (optional) Class: `polyhedron`
direction	Flag to switch between forward and backwards reachability. Class: `char`
N	Number of steps (defaults to 1). Class: `double`

OUTPUT

S	Set of states reachable in N steps. Class: `Polyhedron`
SN	Set of states reachable at each step. Class: `cell`

AUTHOR(s)

INVARIANTSET

Computation of invariant sets for PWA systems

SYNTAX

```
S = system.invariantSet()

S = system.invariantSet('X', X, 'U', U)
```

DESCRIPTION

For an autonomous PWA system $x^+ = f_{P}WA(x)$ this function computes the set of states for which recursive satisfaction of state constraints can be shown.

The set is computed by running the set recursion

$$S_{k+1} = \{x \mid f_{P}WA(x) \in S_k\},$$

initialized by $S_0 = X$ and terminating once $S_{k+1} = S_k$. If X is not provided, $X = \{x \mid x_{min} \leq x \leq x_{max}\}$ is assumed.

For a PWA system $x^+ = f_{P}WA(x, u)$, which is subject to polyhedral state constraints $x \in X$ and input constraints $u \in U$ this function calculates the maximal control-invariant set

$$C = \{x \mid \exists u(k) \in U, \text{ s.t. } x(k) \in X, \ \forall k \geq 0\}.$$

Note that this function requires that state constraints defined in `system.x.min` and `system.x.max` (see "help SystemSignal").

INPUT

X	Polyhedron defining state constraints (optional) Class: `polyhedron`
U	Polyhedron defining input constraints (optional) Class: `polyhedron`
maxIterations	Maximal number of iterations (optional) Class: `double`

OUTPUT

S	Invariant set Class: `Polyhedron`

AUTHOR(s)

OUTPUT

Returns value of the output variables

SYNTAX

```
y = system.output()
```

```
y = system.output(u)
```

DESCRIPTION

This function evaluates the system's output equation and returns the calculated output.

For systems with direct feed-through, it is necessary to provide the vector of inputs as the first input argument.

INPUT

u   Vector of system's inputs
Class: `double`

OUTPUT

y   Vector of outputs
Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

INITIALIZE

Sets the internal state of the system

SYNTAX

```
system.initialize(x0)
```

DESCRIPTION

system.initialize(x0) sets the initial state of the system to value x0.

The internal state can be retrieved by calling system.getStates().

INPUT

x0   Initial state
       Class: double

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

EXPORTTOC

Export the explicit controller to C-code

SYNTAX

```
controller.exportToC
```

```
controller.exportToC('filename')
```

```
controller.exportToC('filename','dirname')
```

DESCRIPTION

The routine `exportToC` generates C-code from the explicit `controller`. A new directory is always created with the generated code. If no name of the directory is provided in the second argument, the default name for the directory is `mpt_explicit_controller`. If no file name is specified, the default name for the files to to be generated is `mpt_getInput`. The directory contains three files:

- `mpt_getInput.c` - The routine for evaluation of PWA control law.

- `mpt_getInput_mex.c` - The mex interface for evaluation using Matlab.

- `mpt_getInput_sfunc.c` - The Simulink interface for evaluation using the `mpt_sim` library.

The generated code can be used for fast evaluation of PWA control law. The generated mex interface file can be compiled in Matlab (assuming that a C-compiler is installed and recognized in Matlab)

```
mex mpt_getInput_mex.c
```

and evaluated for a particular value of the parameters.
The Simulink interface is given by `mpt_getInput_sfunc.c` file. To compile the file, type in Matlab

```
mex mpt_getInput_sfunc.c.
```

After succesful compilation, the generated mex-file can be evaluated within Simulink. For this purpose using one can deploy the controller block from MPT Simulink library `mpt_sim`.

INPUT

filename	Name of the header file to be generated. Class: `char`
dirname	Name of the directory to be generated. Class: `char`

562

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

EVALUATE

Returns the optimal control action

SYNTAX

```
u = controller.evaluate(x0)

[u, feasible] = controller.evaluate(x0)

[U, feasible, openloop] = controller.evaluate(x0)
```

DESCRIPTION

`u = controller.evaluate(x0)` evaluates the explicit MPC solution for the initial condition `x0` and returns the first element of the optimal sequence of control inputs. If `x0` is outside of the controller's domain, `u` will be `NaN`.

`[u, feasible] = controller.evaluate(x0)` also returns the boolean flag indicating whether `x0` is inside of the controller's domain.

`[u, feasible, openloop] = controller.evaluate(x0)` also returns the open-loop predictions of states, inputs and outputs in `openloop.X`, `openloop.Y`, and `openloop.Y`, respectively. Value of the optimal cost is returned in `openloop.cost`.

INPUT

x0   Initial state.
      Class: `double`

OUTPUT

u   Optimal control action.
           Class: `double`

feasible   True if a feasible control actions exists for `x0`, false otherwise.
           Class: `logical`

openloop   Structure containing open-loop predictions and value of the optimal cost.
           Class: `struct`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

EMPCCONTROLLER

Explicit MPC controller

SYNTAX

```
ctrl = EMPCController(model, horizon)
```

```
ctrl = EMPCController(MPCController)
```

DESCRIPTION

Constructs the explicit form of an MPC controller.
The particular type of the optimization problem to be solved parametrically is determined by the type of the prediction model and by its parameters. For a more detailed information, see "`help MPCController`".

Instances of the `EMPCController` class expose following properties:

- `model`: the prediction model used in the MPC setup;

- `N`: the prediction horizon

- `optimizer`: the explicit optimizer as an instance of the `PolyUnion` class;

- `partition`: the polyhedral partition of the explicit feedback law as an instance of the `Polyhedron` class;

- `feedback`: the explicit representation of the feedback law as an instance of the `PolyUnion` class;

- `cost`: the explicit representation of the optimal cost function as an instance of the `PolyUnion` class.

The `optimizer` property is available for read/write access. This allows, for instance, to remove overlaps from multiple overlapping partitions by `ctrl.optimizer = ctrl.optimizer.merge()`.

INPUT

model	Any MPT3 system (LTISystem, PWASystem, MLDSystem)
	Class: `AbstractSystem`

OUTPUT

ctrl	Explicit MPC controller

EXAMPLE(s)

Example 1

Create a 2D LTI system

```
A = [1 1; 0 1]; B = [1; 0.5]; C = [1 0]; D = 0;
sys = LTISystem('A',A,'B',B,'C',C,'D',D)
```

LTISystem with 2 states,1 input,1 output

Define constraints

```
sys.x.min = [-5; -5]; sys.x.max = [5; 5];
sys.u.min = -1; sys.u.max = 1;
sys.y.min = -5; sys.y.max = 1;
```

Define penalties (we use squared two-norm with unity weights here)

```
sys.x.penalty = QuadFunction(eye(2));
sys.u.penalty = QuadFunction(1);
```

Construct the explicit MPC controller

```
horizon = 3;
ctrl = MPCController(sys,horizon).toExplicit()
```

mpt_plcp: 7 regions
Explicit MPC controller (horizon: 3,regions: 7)

Plot the optimal feedback law

```
ctrl.feedback.fplot()
```
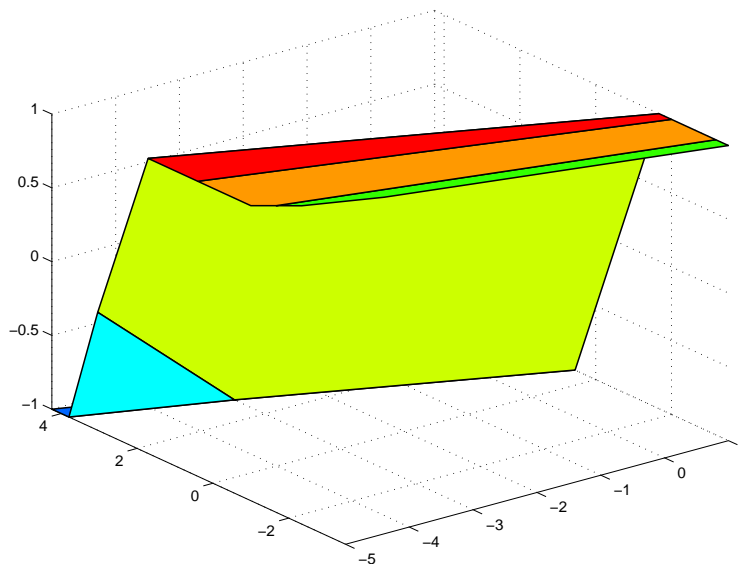

Plot the optimal cost function

```
ctrl.cost.fplot()
```
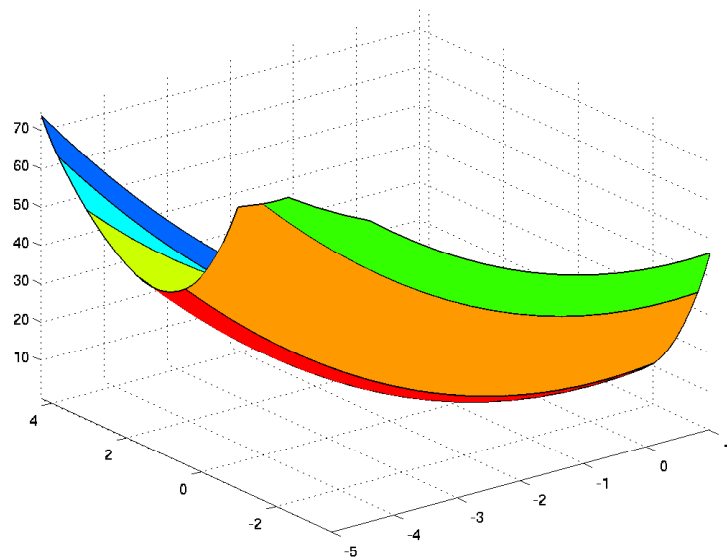

Plot the polyhedral partition

```
ctrl.partition.plot()
```
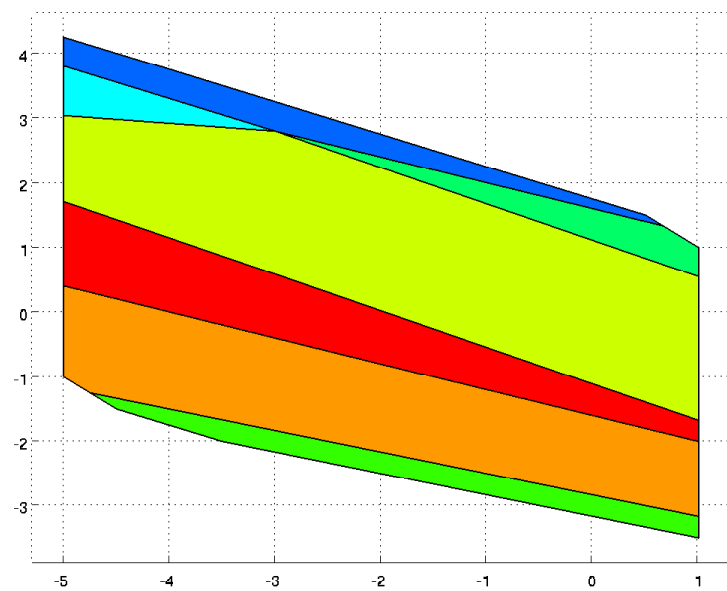

SEE ALSO

MPCController

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

MLDSYSTEM

Represents mixed logical dynamical systems

SYNTAX

```
sys = MLDSystem(source_file)
```

DESCRIPTION

This class represents mixed logical dynamical (MLD) systems of the form

$$x^+ = Ax + B_1u + B_2d + B_3z + B_5$$
$$y = Cx + D_1u + D_2d + D_3z + D_5$$
$$E_2d + E_3z \leq E_1u + E_4x + E_5$$

where x is the state vector, u is the vector of inputs, y is the vector of outputs, d is the vector of auxiliary binary variables, and z is the vector of auxiliary real variables.

Each MLD system defines following properties:

- `A`, `B1`, `B2`, `B3`, `B5`: matrices of the state-update equation (read-only)

- `C`, `D1`, `D2`, `D3`, `D5`: matrices of the output equation (read-only)

- `E1`, `E2`, `E3`, `E4`, `E5`: matrices of the constraints (read-only)

- `nx`, `nu`, `ny`, `nd`, `nz`: number of states, inputs, outputs, binaries, and aux reals (read-only)

- `x`: specifications of system's states (see `help SystemSignal`)

- `u`: specifications of system's inputs (see `help SystemSignal`)

- `y`: specifications of system's outputs (see `help SystemSignal`)

- `d`: specifications of binary variables (see `help SystemSignal`)

- `d`: specifications of auxiliary real variables (see `help SystemSignal`)

MLD models are automatically created from a corresponding HYSDEL source file upon calling the constructor as follows:

```
mldsys = MLDSystem('model.hys')
```

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

GETSTATES

Returns the internal state of the system

SYNTAX

```
x = system.getStates()
```

DESCRIPTION

This function returns the system's internal state.

OUTPUT

x   Values of the internal states
Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

TOPWA

Converts the MLD model into an equivalent PWA form

SYNTAX

```
pwasys = mldsys.toPWA()
```

DESCRIPTION

For a PWA system consisting of M local affine models, this function converts the `index`-th model to an instance of the `LTISystem` class.

OUTPUT

pwasys   PWA representation of the MLD system.
        Class: `PWASystem`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
     mailto:michal.kvasnica@stuba.sk

UPDATE

Updates the internal state using the state-update equation

SYNTAX

```
system.update(u)

xn = system.update(u)

[xn, y] = system.update(u)
```

DESCRIPTION

This function evaluates the system's state-update equation and updates the internal state of the system.

By calling `system.update(u)` this function updates the internal state of `system` by evaluating the state-update equation. The updated state can then be retrieved by calling `x = system.getStates()`.

By calling `[xn, y] = system.update(u)` this function also returns the updated state as the first output, and the result of the output equation as the second output. Note that the internal system's state is still updated as described above.

INPUT

u   Vector of system's inputs
Class: `double`

OUTPUT

xn   Updated state vector
Class: `double`

y   Vector of outputs
Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

SIMULATE

Simulates evolution of the system

SYNTAX

```
data = system.simulate(U)
```

DESCRIPTION

`data = system.simulate(U)` computes evolution of system's states and outputs, starting from the system's internal state and using a sequence of inputs `U`.

Each column of `U` is interpreted as the control action to use at the m-th step of the simulation. The total number of simulation steps is given by the number of columns of `U`. To simulate an autonomous system over `M` steps, you need to define `U=zeros(0, M)`.

This function returns a structure `data`, which contains the simulated evolution of system's states (in `data.X`) and the outputs (in `data.Y`), respectively.

Note that you should always run `system.initialize(x0)` to set the initial condition prior to running the simulation.

Also note that the `simulate` method updates the internal system's state. Therefore once the function completes, the internal state will be set to the final value obtained at the end of the simulation.

INPUT

U   Matrix of control inputs stored column-wise for each
    simulation step.
    Class: `double`

OUTPUT

data   Structure with simulated state and output profiles.
       Class: `struct`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

OUTPUT

Returns value of the output variables

SYNTAX

```
y = system.output()
```

```
y = system.output(u)
```

DESCRIPTION

This function evaluates the system's output equation and returns the calculated output.

For systems with direct feed-through, it is necessary to provide the vector of inputs as the first input argument.

INPUT

u Vector of system's inputs
  Class: `double`

OUTPUT

y Vector of outputs
  Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
  mailto:michal.kvasnica@stuba.sk

INITIALIZE

Sets the internal state of the system

SYNTAX

```
system.initialize(x0)
```

DESCRIPTION

`system.initialize(x0)` sets the initial state of the system to value `x0`.

The internal state can be retrieved by calling `system.getStates()`.

INPUT

x0   Initial state
     Class: `double`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

FROMYALMIP

Converts a custom YALMIP setup to a controller

SYNTAX

```
controller.fromYALMIP(yalmipdata)
```

DESCRIPTION

This function convers a custom YALMIP optimization problem to an MPT3 controller object.

INPUT

yalmipdata	Structure generated by `toYALMIP` Class: `struct`

SEE ALSO

toYALMIP

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

TOYALMIP

Converts an MPC problem into YALMIP's constraints and objective

SYNTAX

```
yalmipdata = controller.toYALMIP()
```

DESCRIPTION

This function convers an MPC optimization problem into YALMIP. The output structure contains following fields:

- `constraints`: contains constraints of the MPC problem as YALMIP's `lmi` object.

- `objective`: scalar variable of type `sdpvar` which defines the optimization objective.

- `variables`: structure containing variables of the optimization problem.

This method is MPT3 replacement of MPT2's `ownmpc` mechanism. In short, `toYALMIP` allows to modify the MPC problem by adding new constraints and/or by modifying the objective function.

OUTPUT

`yalmipdata`	Structure containing constraints, objective, and variables of the MPC problem. Class: `struct`

EXAMPLE(s)

Example 1

Create a 1D LTI system $x^+ = 0.9x + u$.

```
sys = LTISystem('A',0.9,'B',1);
```

Define an MPC controller.

```
ctrl = MPCController(sys);
```

Specify constraints, penalties, and the prediction horizon.

```
ctrl.model.x.min = -5; ctrl.model.x.max = 5;
ctrl.model.u.min = -5; ctrl.model.u.max = 5;
ctrl.model.x.penalty = QuadFunction(1);
ctrl.model.u.penalty = QuadFunction(1);
ctrl.N = 3;
```

Obtain the optimal sequence of inputs over the whole prediction horizon for the initial condition `x0=3`.

```
[~,~,openloop] = ctrl.evaluate(3); openloop.U


ans =

-1.57733887733888        -0.505197505197505       5.55111512312578e-17
```

Convert the problem to YALMIP and add custom state constraints on the first two predicted inputs via YALMIP.

```
Y = ctrl.toYALMIP()
Y.constraints = Y.constraints + [ -0.5 <= Y.variables.u(:,1) <= 0.5];
Y.constraints = Y.constraints + [ -0.8 <= Y.variables.u(:,2) <= 0.8];
ctrl.fromYALMIP(Y);
x0 = 3;
[~,~,openloop] = ctrl.evaluate(x0); openloop.U



Y =

constraints: [4 lmi]
objective: [1x1 sdpvar]
variables: [1x1 struct]
internal: [1x1 struct]


ans =

-0.5                   -0.8      -6.418622835482e-17
```

Notice that the updated controller respects constraints added via YALMIP.

SEE ALSO

fromYALMIP

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

TOSYSTEM

Converts the closed-loop object to an autonomous dynamical system

SYNTAX

```
sys = loop.toSystem()
```

DESCRIPTION

This function converts the closed-loop object to an autonomous dynamical system by using the state feedback $u = f(x)$, where $f(x)$ represents the explicit form of the controller.

If the system is an LTI system (i.e., $x^+ = Ax + Bu$), and the controller is defined over a single region (i.e., $f(x) = Kx + k$), this function produces an instance of the `LTISystem` class with the dynamic $x^+ = (A + BK)x + Bk$.

If the system is PWA, or if the controller is defined over multiple regions, the `toSystem()` method produces an autonomous system as an instance of the `PWASystem` class. The number of dynamics is equal to the number of regions, over which the controller is defined.

In both cases the returned objects represent autonomous systems, i.e. their `B` and `D` matrices are empty.

Note: this function requires that the closed-loop's controller is available in its explicit form.

OUTPUT

`sys`	Autonomous dynamical system representing the closed-loop system.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

CLOSEDLOOP

Create a closed-loop system.

SYNTAX

```
loop = ClosedLoop ( controller , system )
```

DESCRIPTION

Creates a closed-loop system composed of a controller and a system using a state feedback. The object contains two read-only properties:

- `controller`: the controller

- `system`: the system

Note: the `controller` and `system` properties are linked to original objects using references. It means that if you change the system and/or the controller in your MATLAB workspace, the change will automatically propagate to the closed-loop object without the need to re-create it manually.

INPUT

controller	Any MPT3 controller (`MPCController`, `EMPCController`, `LQRController`, ...) Class: `AbstractController`
system	Any MPT3 system (`LTISystem`, `PWASystem`, `MLDSystem`) Class: `AbstractSystem`

OUTPUT

loop	Instance of the `ClosedLoop` class.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

SIMULATE

Simulates the closed-loop system.

SYNTAX

```
data = loop.simulate(x0, M)
```

DESCRIPTION

Simulates the closed-loop system over `M` time steps, starting from the initial condition `x0`.

INPUT

x0   Initial condition.
       Class: `double`

N_steps   Number of simulation steps.
       Class: `double`

OUTPUT

data   Structure containing closed-loop profiles of state, input, and output variables.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

INVARIANTSET

Computes the invariant subset of the closed-loop system

SYNTAX

```
invset = loop.invariantSet()
```

DESCRIPTION

For a closed-loop system, consisting of a system and a controller, this function computes the set of states of the system for which the controller provides recursive satisfaction of system's state and input constraints.

OUTPUT

invset   Invariant subset of the closed-loop system as an instance of the `Polyhedron` class.

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

EVALUATE

Returns the optimal control action

SYNTAX

```
u = controller.evaluate(x0)

[u, feasible] = controller.evaluate(x0)

[U, feasible, openloop] = controller.evaluate(x0)
```

DESCRIPTION

`u = controller.evaluate(x0)` evaluates the explicit MPC solution for the initial condition `x0` and returns the first element of the optimal sequence of control inputs. If `x0` is outside of the controller's domain, `u` will be `NaN`.

`[u, feasible] = controller.evaluate(x0)` also returns the boolean flag indicating whether `x0` is inside of the controller's domain.

`[u, feasible, openloop] = controller.evaluate(x0)` also returns the open-loop predictions of states, inputs and outputs in `openloop.X`, `openloop.Y`, and `openloop.Y`, respectively. Value of the optimal cost is returned in `openloop.cost`.

INPUT

x0   Initial state.
       Class: `double`

OUTPUT

u   Optimal control action.
          Class: `double`

feasible True if a feasible control actions exists for `x0`, false otherwise.
          Class: `logical`

openloop Structure containing open-loop predictions and value of the optimal cost.
          Class: `struct`

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

EMINTIMECONTROLLER

Explicit minimum-time MPC controller

SYNTAX

```
ctrl = EMinTimeController(model)
```

DESCRIPTION

Constructs the explicit form of a minimum-time MPC controller by solving the following optimization problem parametrically:

$$\min N$$
$$\text{s.t.} x(k+1) = f(x_k, u_k)$$
$$x(N) \in T_{set}$$
$$x(k) \in X$$
$$u(k) \in U$$

where $f(x_k, u_k)$ represents the prediction model's state-update equation, T_{set} is a given terminal set, and X and U denote, respectively, the state and input constraints.

LTI, PWA, and MLD models are accepted for predictions. State and input constraint sets X, U are automatically created using the model's state and input constraints (`model.x.min`, `model.x.max`, `model.u.min`, `model.u.max`). The terminal set T_{set} is either taken from `model.x.terminalSet` or, if the field is empty, by calling `model.stabizilingController()` (for LTI systems only).

Once computed, explicit minimum-time controllers inherit the behavior of the `EMPCController` class in terms of internal structure, evaluation and post-processing. See "`help EMPCController`" for more details.

INPUT

model   Prediction model
        Class: `AbstractSystem`

OUTPUT

ctrl   Explicit minimum-time controller.
        Class: `EMinTimeController`

EXAMPLE(s)

Example 1

Create a 1D LTI system

```
sys = LTISystem('A',1,'B',1,'C',1,'D',0)
```

```
LTISystem with 1 state,1 input,1 output
```

Define constraints and quadratic cost function with unity weights

```
sys.x.min = -1; sys.x.max = 1;
sys.u.min = -1; sys.u.max = 1;
sys.x.penalty = QuadFunction(1);
sys.u.penalty = QuadFunction(1);
```

Define a terminal set $T_{set} = \{x \mid -0.1 \le x \le 0.1\}$ and construct the explicit minimum-time controller

```
sys.x.with('terminalSet');
sys.x.terminalSet = Polyhedron('lb',-0.1,'ub',0.1);
ctrl = EMinTimeController(sys)
```

```
Using provided terminal state set...
Iterating...
New horizon: 1
mpt_plcp: 3 regions
New horizon: 2
mpt_plcp: 1 regions
Explicit minimum-time controller (horizon: 1,regions: 4,partitions: 2)
```

SEE ALSO

MinTimeControllerr, EMPCController

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

MPT_DEMO_LTI4

Construction of explicit controller for LTI system

SYNTAX

```
mpt_demo_lti4
```

DESCRIPTION

Construction of explicit controller for LTI system

EXAMPLE(s)

Example 1

Define an LTI prediction model $x^+ = Ax + Bu$

```
A = [1 1; 0 1]; B = [1; 0.5];
```

```
lti = LTISystem('A',A,'B',B);
```

Define an MPC controller

```
horizon = 5;
ctrl = MPCController(lti);
ctrl.N = horizon;
ctrl.model.x.min = [-5; -5];
ctrl.model.x.max = [5; 5];
ctrl.model.u.min = -1;
ctrl.model.u.max = 1;
ctrl.model.x.penalty = QuadFunction(eye(ctrl.model.nx));
ctrl.model.u.penalty = QuadFunction(eye(ctrl.model.nu));
```

Compute the explicit solution

```
exp_ctrl = ctrl.toExplicit()
```

```
mpt_plcp: 21 regions
Explicit MPC controller (horizon: 5,regions: 21)
```

Plot the explicit feedback law

```
exp_ctrl.feedback.fplot()
```
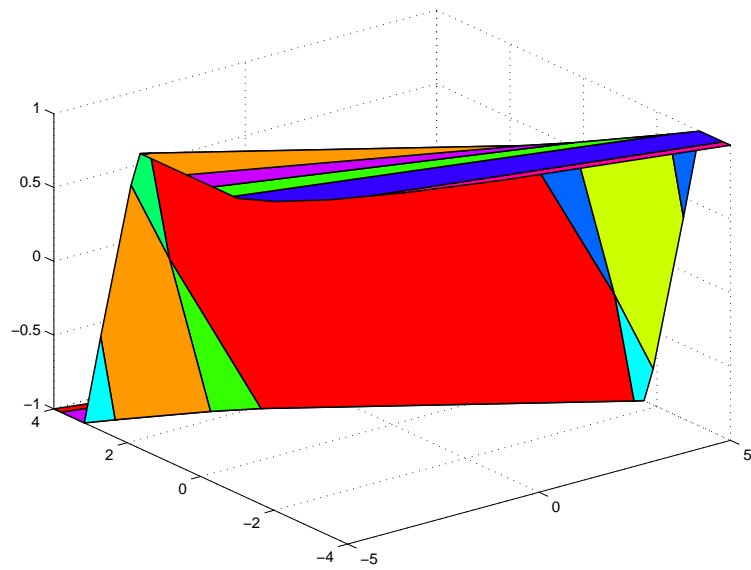
Plot the explicit value function

```
exp_ctrl.cost.fplot()
```
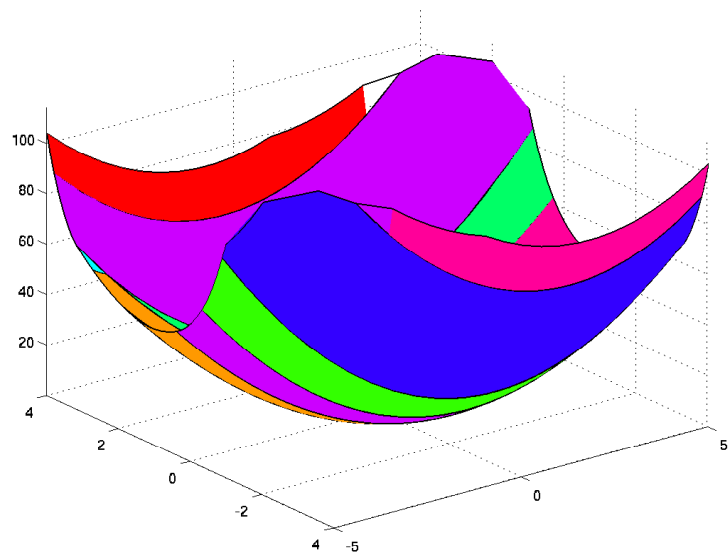

Plot the polyhedral partition

```
exp_ctrl.partition.plot()
```
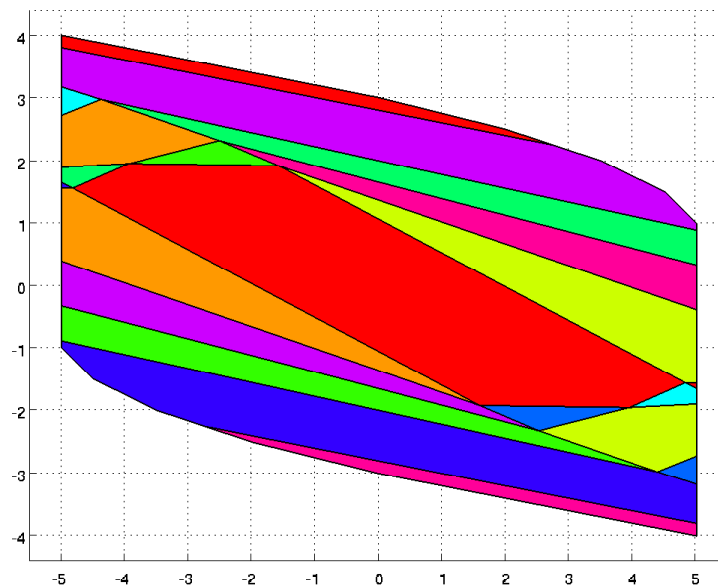
Compare the optimal control inputs

```
x0 = [-4; 0];
Uonl = ctrl.evaluate(x0),
Uexp = exp_ctrl.evaluate(x0)
```

Uonl =

1

Uexp =

1

SEE ALSO

mpt_demo_lti1, mpt_demo_lti2, mpt_demo_lti3, mpt_demo_lti5

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

MPT_DEMO_PWA1

Demonstration for modeling PWA systems

SYNTAX

`mpt_demo_pwa1`

DESCRIPTION

Demonstration for modeling PWA system comprising of two linear time-invariant models:

$$
\begin{aligned}
x^+ &= \begin{cases} A_1 x + Bu & \text{if } x_1 \le 0 \\ A_2 x + Bu & \text{if } x_1 \ge 0 \end{cases} \\
y &= Cx + Du
\end{aligned}
$$

EXAMPLE(s)

Example 1

PWA systems are created by defining each dynamics as an LTI system:

`B = [0; 1]; C = [1 0]; D = 0;`

First dynamics:

`alpha = -pi/3;`

`A1 = 0.8*[cos(alpha) -sin(alpha); sin(alpha) cos(alpha)];`

`dyn1 = LTISystem('A',A1,'B',B,'C',C,'D',D);`

We need to tell that dynamics 1 should be active if $x_1 \le 0$:

`dyn1.setDomain('x',Polyhedron([1 0],1));`

Second dynamics:

`alpha = pi/3;`

`A2 = 0.8*[cos(alpha) -sin(alpha); sin(alpha) cos(alpha)];`

`dyn2 = LTISystem('A',A2,'B',B,'C',C,'D',D);`

Region of validity of the dynamics ($x_1 \ge 0$):

`dyn2.setDomain('x',Polyhedron([-1 0],0));`

Create the PWA description using an array of LTI systems:

`pwa = PWASystem([dyn1 dyn2]);`

`State/input/output constraints not imported,set them manually afterwards.`

Optionally we can set constraints:

```
pwa.x.min = [-10; -10];
```

```
pwa.x.max = [10; 10];
```

```
pwa.y.min = -10;
```

```
pwa.y.max = 10;
```

```
pwa.u.min = -1;
```

```
pwa.u.max = 1;
```

Define an on-line MPC controller for such a system

```
horizon = 2;
```

```
onl_ctrl = MPCController(pwa,horizon);
```

Set one-norm penalties used in the cost function:

```
onl_ctrl.model.x.penalty = OneNormFunction(10*eye(2));
```

```
onl_ctrl.model.u.penalty = OneNormFunction(1);
```

Construct the explicit solution

```
exp_ctrl = onl_ctrl.toExplicit();
```

```
mpt_plcp: 6 regions
mpt_plcp: 7 regions
mpt_plcp: 6 regions
mpt_plcp: 13 regions
->Generated 4 partitions.
```

Obtain the closed-loop optimizers for a particular initial condition

```
x0 = [-4; 0];
```

```
Uonl = onl_ctrl.evaluate(x0)
```

```
Uonl =

-1
```

```
Uexp = exp_ctrl.evaluate(x0)
```

```
Uexp =

-1
```

AUTHOR(s)

MPT_DEMO_SETS2

Construction and basic properties of sets created in Yalmip

SYNTAX

```
mpt_demo_sets2
```

DESCRIPTION

Slideshow on how to construct and work with `YSet` objects. In particular, it is demonstrated how to:

- Create Yalmip `YSet` objects.

- Access internal information stored in the `YSet` object.

- Query basic properties of the `YSet` object.

SEE ALSO

mpt_demo_sets1, mpt_demo_sets3

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_DEMO_UNIONS2

Demo that illustrates working with unions of polyhedra

SYNTAX

```
mpt_demo_unions2
```

DESCRIPTION

Demo that illustrates working with unions of polyhedra. It will be shown how to:

- construct PolyUnion objects,
- query basic properties from the union, and
- perform geometric operations with unions of polyhedra.

SEE ALSO

mpt_demo_unions1

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_DEMO1

Demonstration of basic usage of the geometric library

SYNTAX

`mpt_demo1`

DESCRIPTION

Basic usage of the new interface to geometric library

EXAMPLE(s)

Example 1

Create polytope:

`P = Polyhedron('V',randn(10,2));`

Plot the polytope

`P.plot('color','b','alpha',0.3,'linewidth',1,'linestyle','--'); axis off;`
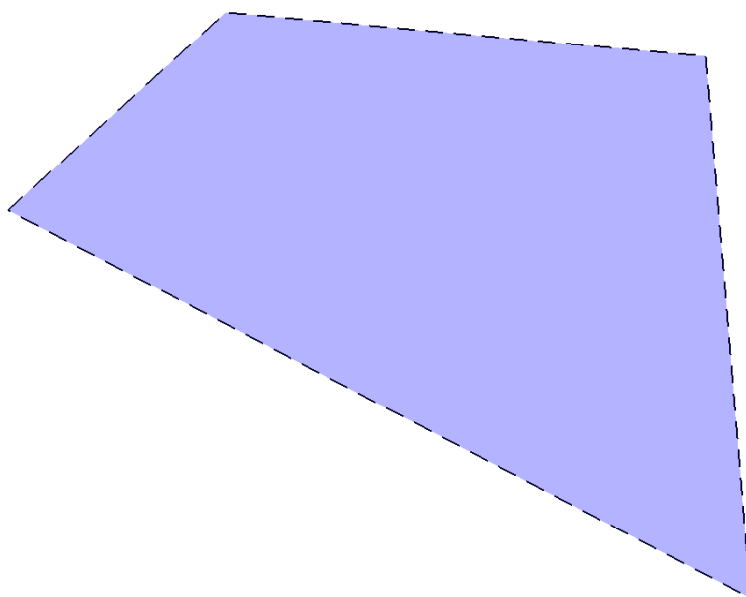
Double description created and stored automatically

`P`

```
Polyhedron in R^2 with representations:
H-rep (irredundant) : Inequalities   4 | Equalities   0
V-rep (irredundant) : Vertices   4 | Rays   0
Functions : none
```

Example 2

Create Polyhedron

```
P = Polyhedron('V',randn(10,2),'R',randn(1,2));
```

```
P.plot;
```
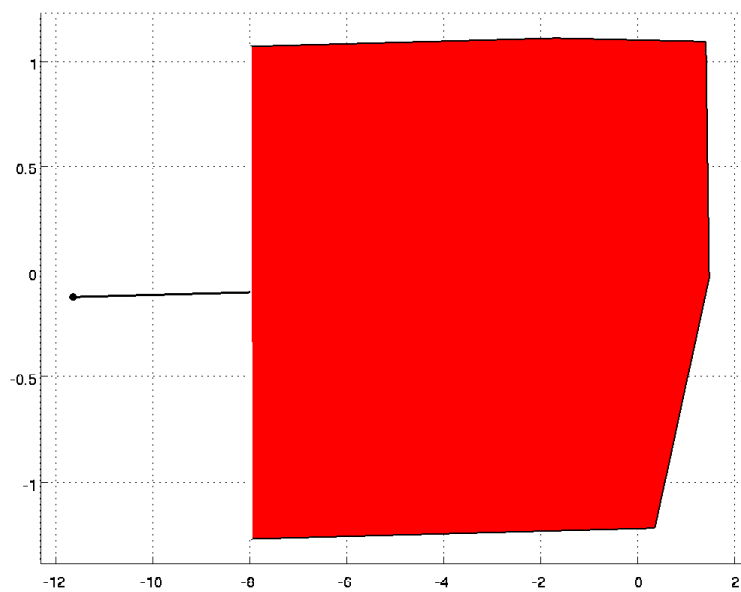

Create another Polyhedron

```
P = Polyhedron('V',randn(50,3),'R',randn(1,3));
```

```
P.plot;  axis vis3d; figure(1);
```
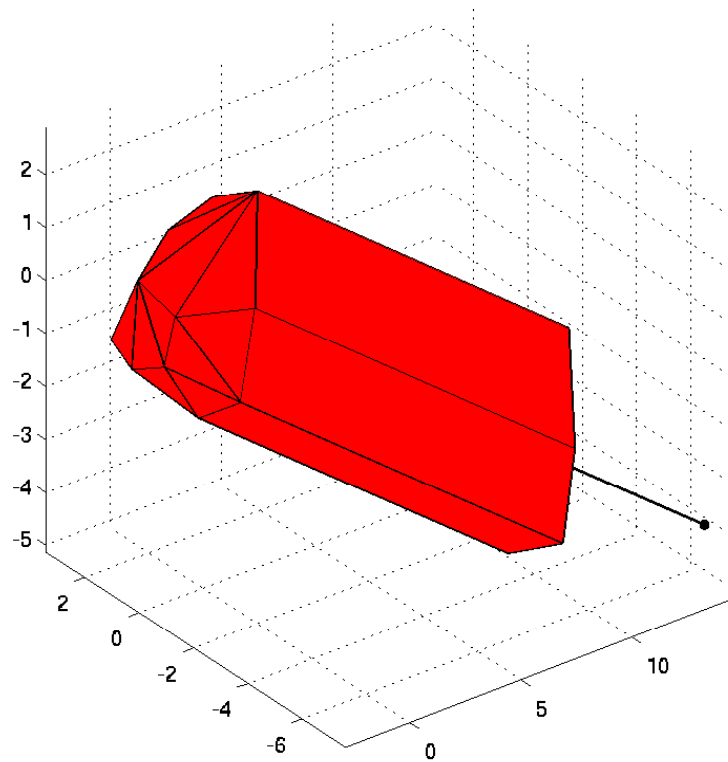
Example 3

Lower-dimensional polyhedra

```
P = Polyhedron('H',[randn(30,3) ones(30,1)],'He',[randn(1,3) 0]);

P.plot('alpha',0.3);
```
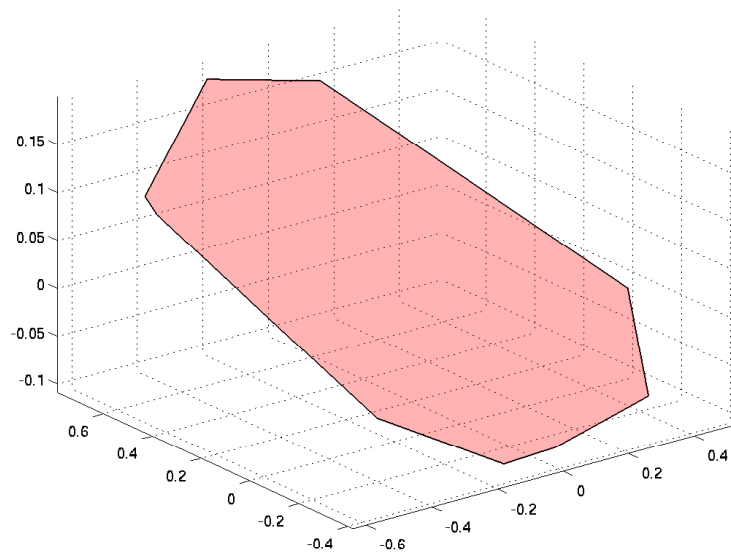
Example 4

Boxes

`P = Polyhedron('lb',-rand(3,1),'ub',rand(3,1),'H',[randn(10,3) 1.5*ones(10,1)]);`

`P.plot('alpha',0.3);`
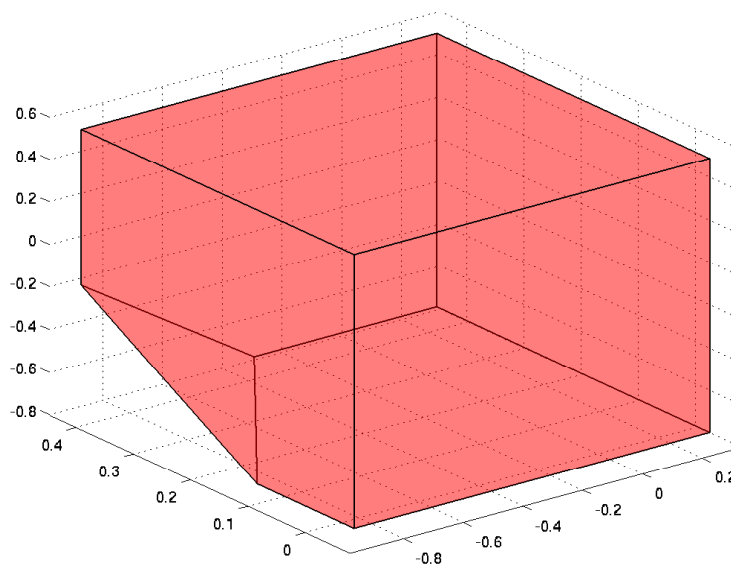
Example 5

Polyhedron queries
Project onto polyhedron:

```
P = Polyhedron(randn(10,2),ones(10,1));

P.plot('color','g');
hold on;
axis square;
x = 5*randn(2,1);
sol = P.project(x);
pplot([x sol.x]','bo');
sep = P.separate(x);
v = axis;
s = Polyhedron('He',sep,'lb',[v(1);v(3)],'ub',[v(2);v(4)]);
s.plot;
sol = P.interiorPoint;
pplot(sol.x,'ro');
```
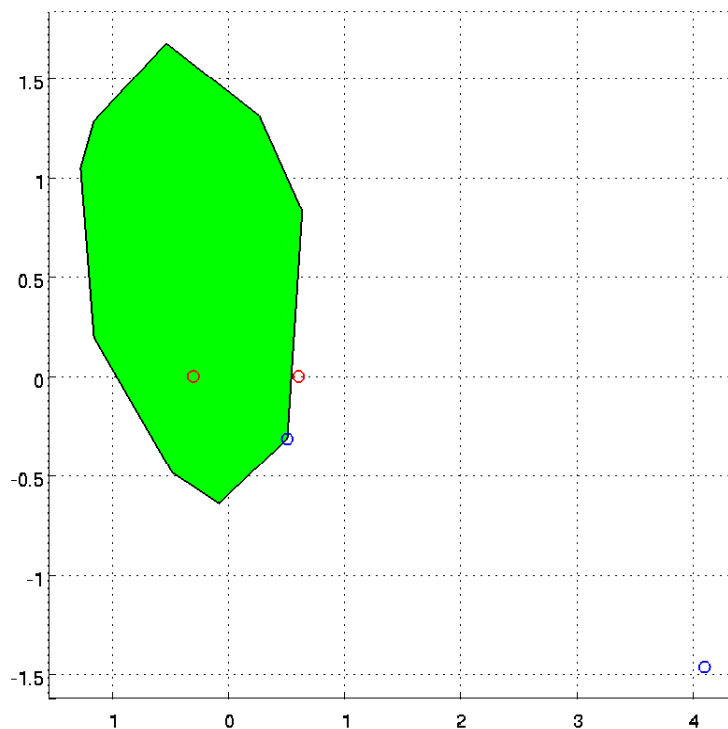

Addition

```
P = Polyhedron(3*randn(10,2));
```

```
Q = Polyhedron('H',[randn(10,2) ones(10,1)]);
```

```
P.plot; hold on; Q.plot;
```
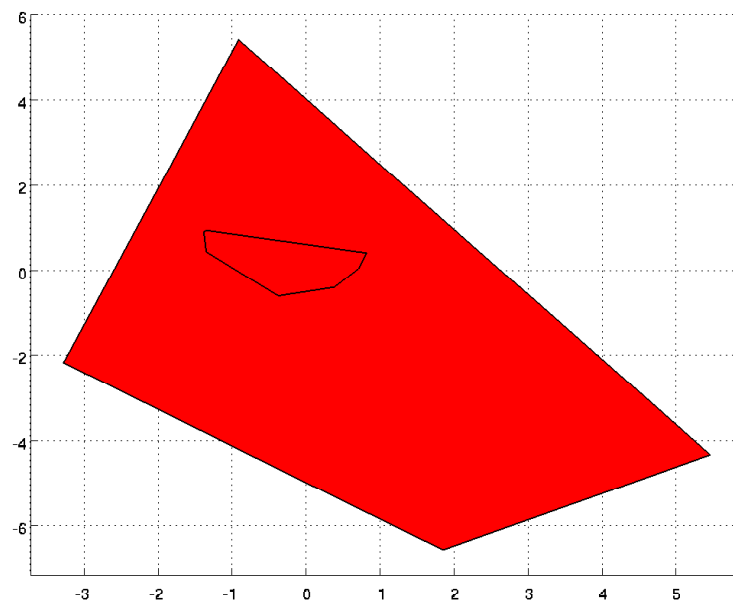
```
PQ = P+Q;

PQ.plot('alpha',0.1,'color','r');
```
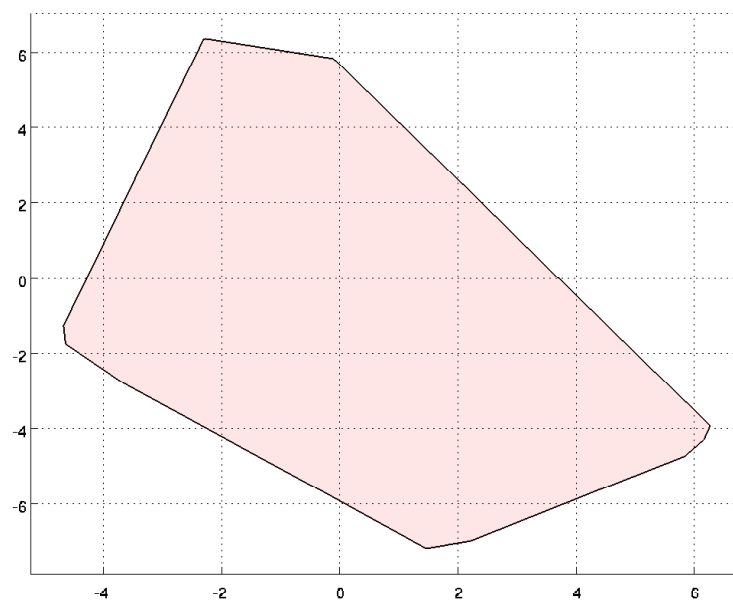

Lower-dimensional addition

```
for i=1:5
P(i) = Polyhedron('V',[0 0 0;randn(1,3)]);
```

```
end


plot(P,'linewidth',2); hold on;
Q = Polyhedron;
for i=1:5
Q = Q + P(i);
end
Q.plot('alpha',0.1,'color','b');
axis vis3d
```
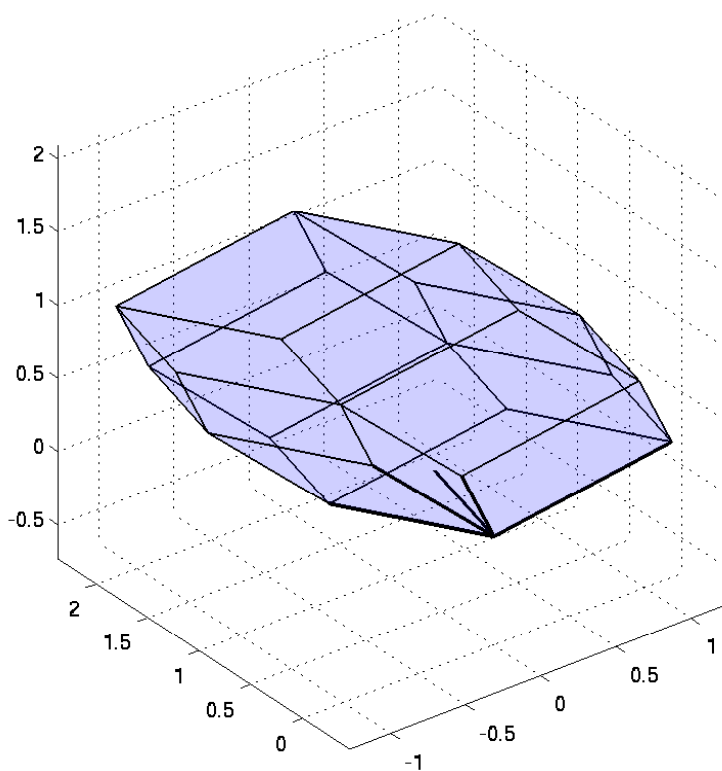

Example 6

Operations with convex sets

```
x = sdpvar(2,1);

T = eye(2);

F = [x <= 0.1 ; x'*T'*T*x <= 1];

Y = YSet(x,F);
```

```
Y.plot('alpha',0.3,'color','b','grid',50);
hold on;
x = [-2;-2];
sol = Y.project(x);
pplot([x sol.x]','bo');
sol = Y.separate(x);
v = axis;
s = Polyhedron('He',sol,'lb',[v(1);v(3)],'ub',[v(2);v(4)]);
s.plot('alpha',0.2);
O = Y.outerApprox;
O.plot('alpha',0.1,'color','b'); axis(axis*1.1);
```

Plotting...
35 of 50
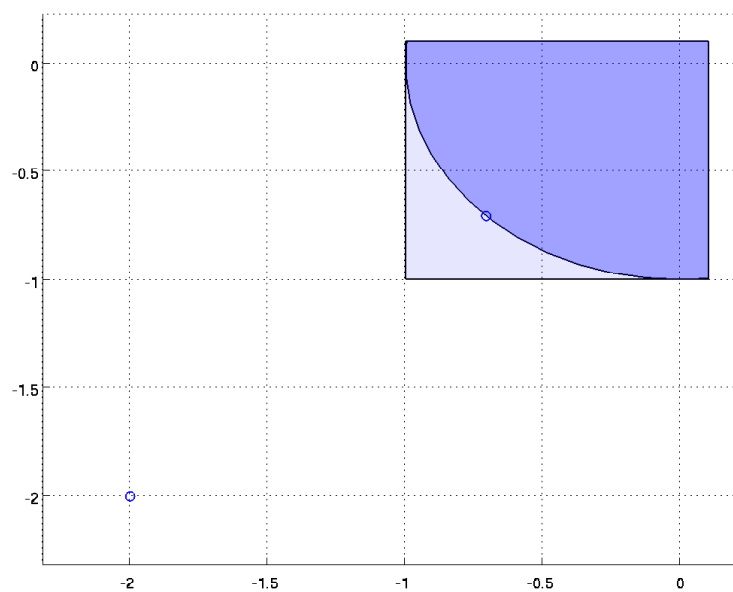
Example 7

Create set of polytopes

```
Ps = PolyUnion;


for i=1:5
Ps.add(Polyhedron(randn(10,2)) + 5*randn(2,1));
end
```

601

```
for i=1:5
Ps.add(Polyhedron('H',[randn(10,2) ones(10,1)],'He',[randn(1,2) 0]) + 5*randn(2,1));
end
```

```
for i=1:5
Z = Polyhedron;
for j=1:3,
Z = Z + Polyhedron([0 0;randn(1,2)]);
end
Ps.add(Z+5*randn(2,1));
end
```

```
Ps.plot;
```
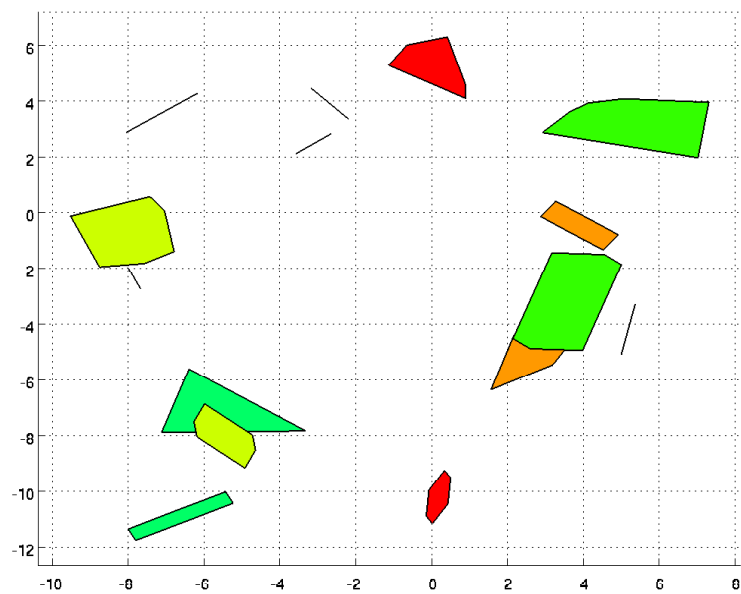

Create complex

```
T = Polyhedron('H',[randn(30,2) ones(30,1)]);
```

```
T = triangulate(T);
```

```
T.plot;
```
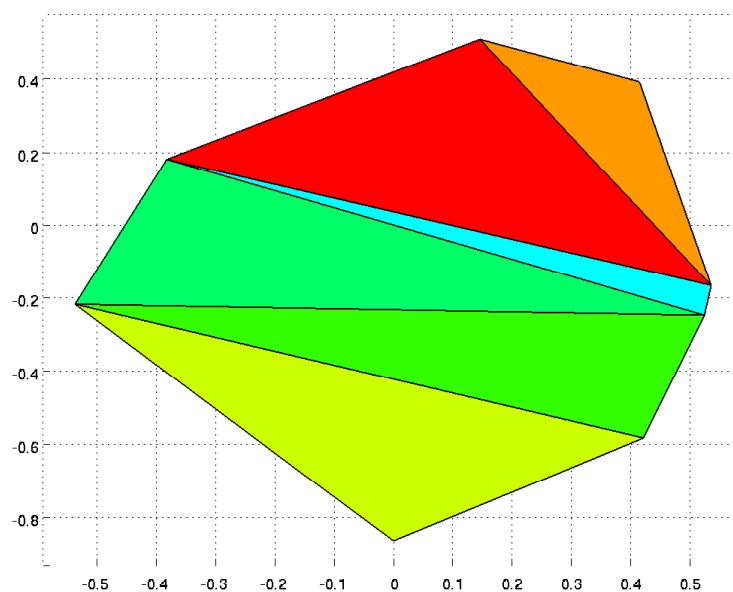
Example 8

Polyhedral functions

```
F = Polyhedron('V',2*randn(10,2))


Polyhedron in R^2 with representations:
H-rep               : Unknown (call computeHRep() to compute)
V-rep (redundant)   : Vertices  10 | Rays   0
Functions : none



F.addFunction(Function(@(x) sin(x(1))*cos(x(2))),'func1');

F.fplot;
```
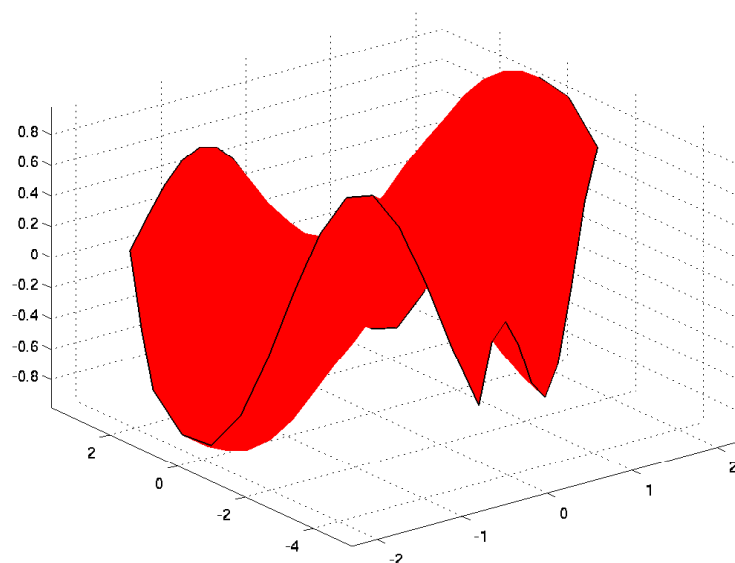
PolyUnion

```
func = @(x) sin(x(1))*cos(x(2));
```

```
for i=1:5
F(i) = Polyhedron('V',2*randn(10,2)) + 5*randn(2,1);
end
F.addFunction(Function(@(x) sin(x(1))*cos(x(2))),'wave');
```

```
Ps = PolyUnion(F);
```

Plot the function over the set

```
Ps.fplot;
```
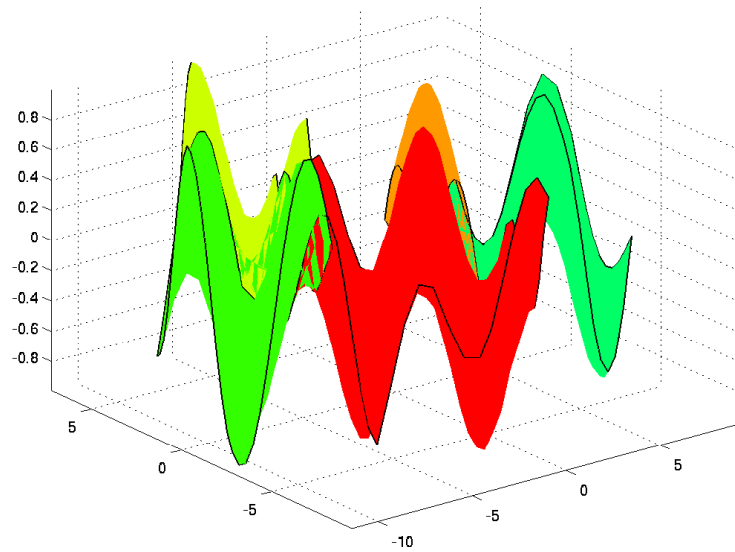
Example 9

Parametric solutions
Formulate MPC problem with the following dimensions

`n = 2; m = 1; N = 5;`

Setup the process model and constraints

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

`B = [1; 0.5];`

`xlb = -10*ones(n,1);`

`xub = 10*ones(n,1);`

`ulb = -5*ones(m,1);`

`uub = 5*ones(m,1);`

`R = 2*eye(m);`

`Q = 0.2*eye(n);`

Formulate MPQP using Yalmip

`x = sdpvar(n,N,'full');`

`u = sdpvar(m,N-1,'full');`

`cost = 0;`

`F = [];`

```
for i=1:N-1
F = F + (x(:,i+1) == A*x(:,i) + B*u(:,i));
F = F + (xlb <= x(:,i) <= xub);
F = F + (ulb <= u(:,i) <= uub);
if i >1
cost = cost + x(:,i)'*Q*x(:,i);
end
cost = cost + u(:,i)'*R*u(:,i);
end
```

```
F = F + [xlb <= x(:,end) <= xub];
```

```
cost = cost + x(:,end)'*Q*x(:,end);
```

Solve using MPQP solver
Change globally the parametric QP solver

```
mptopt('pqpsolver','MPQP');
```

Construct the problem

```
problem1 = Opt(F,cost,x(:,1),u(:));
```

Solve

```
res1 = problem1.solve;
```

```
Calling mpt_mpqp_26 with default options...
mpt_mpqp: 13 regions
```

Solve using PLCP solver
Change globally the parametric QP solver

```
mptopt('pqpsolver','PLCP');
```

Call problem constructor

```
problem2 = Opt(F,cost,x(:,1),u(:));
```

Solve

```
res2 = problem2.solve;
```

```
mpt_plcp: 13 regions
```

Plot the partitions

```
plot(res1.xopt);  title('PLCP'); axis tight;
```

606
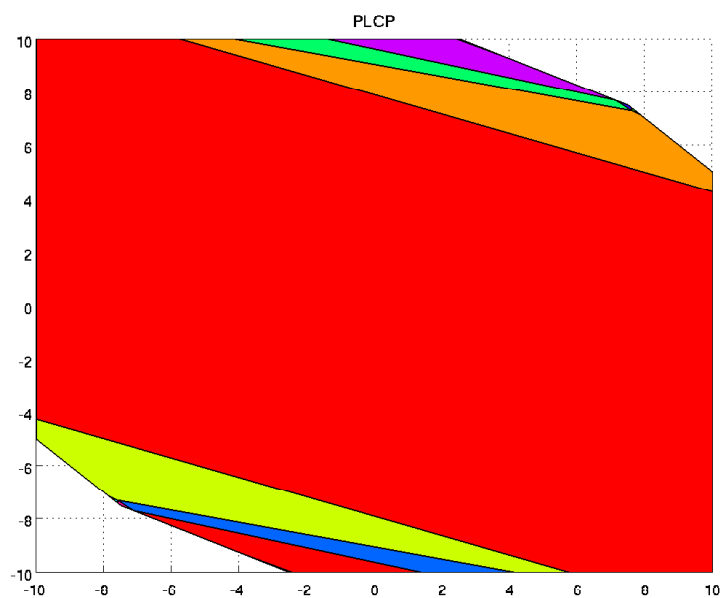
```
plot(res2.xopt);   title('MPQP'); axis tight;
```
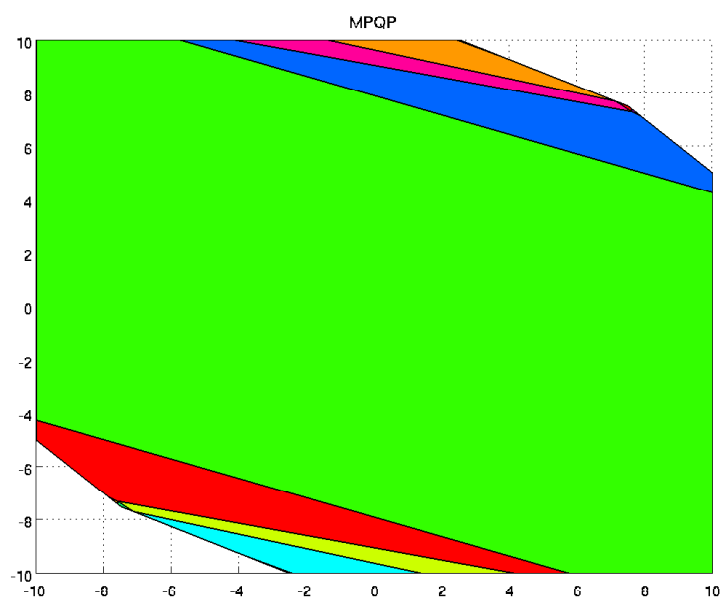

Formulate MPLP problem in YALMIP

```
cost = 0;
```

```
F = [];
```

```
for i=1:N-1
F = F + (x(:,i+1) == A*x(:,i) + B*u(:,i));
F = F + (xlb <= x(:,i) <= xub);
F = F + (ulb <= u(:,i) <= uub);
if i >1
cost = cost + norm(Q*x(:,i),1);
end
cost = cost + norm(R*u(:,i),1);
end


F = F + [xlb <= x(:,end) <= xub];

cost = cost + norm(Q*x(:,end),1);
```

Solve using MPLP solver

```
mptopt('plpsolver','MPLP');

problem3 = Opt(F,cost,x(:,1),u(:));

res3 = problem3.solve;


Calling mpt_mplp_26 with default options...
mpt_mplp: 28 regions
```

Solve using PLCP solver

```
mptopt('plpsolver','PLCP');

problem4 = Opt(F,cost,x(:,1),u(:));

res4 = problem4.solve;


regions:   21,unexplored: 5
mpt_plcp: 28 regions
```

Plot the partitions

```
plot(res3.xopt);  title('MPLP'); axis tight;
```
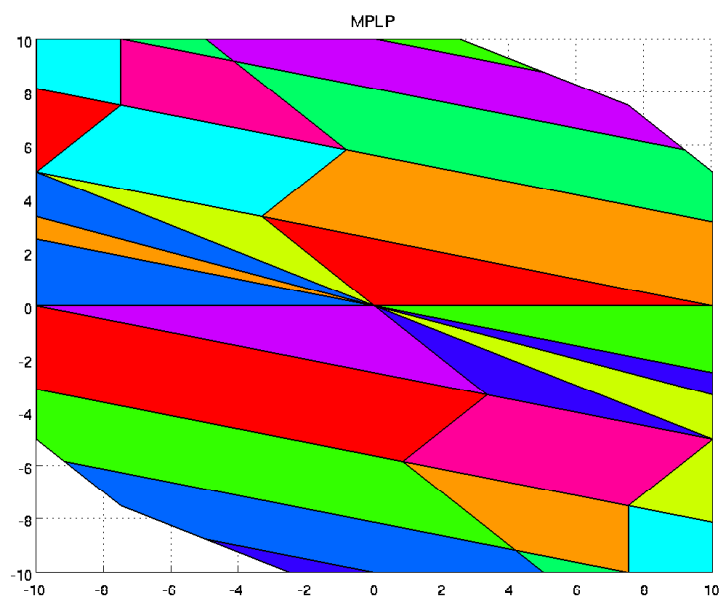
```
plot(res4.xopt);  title('PLCP'); axis tight;
```
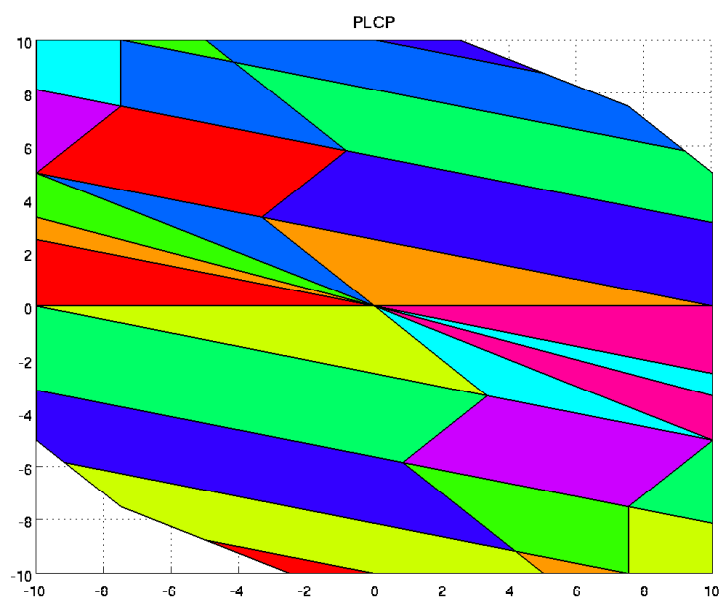

SEE ALSO

mpt_demo_sets1, mpt_demo_functions1, mpt_demo_unions1

609

AUTHOR(s)

MPT_DEMO_DEPLOYMENT_EXPLICITMPCTRACKING

Application of tracking explicit MPC controller with the help of Simulink interface

SYNTAX

```
mpt_demo_deployment_explicitMPCtracking
```

DESCRIPTION

Demonstration of real-time control using explicit MPC controller designed for tracking of a time-varying reference. The objective is to minimize one-norm cost function
$$\min_u \sum_{i=0}^{N-1} \|y - y_{\mathrm{ref}}\|_1 + \|\Delta u\|_1$$
over the horizon $N = 4$ subject to input/output constraints and linear prediction model.

The generated code can be applied for testing in Real-Time Workshop. The code generation and compilation for real-time has been tested under Real-Time Windows target on Windows 32-bit and 64-bit platforms.

Deployment steps:

1. Generate the explicit controller that has the desired properties.

2. Export the explicit controller to C-code using `mpt_exportToC` function.

3. Create a Simulink scheme with the S-Function block that represents generated controller.

4. In the Simulink scheme, choose code generation options and pick **rtwin.tlc** as the system target file (e.g. for Real-Time Windows target).

5. In the code generation options, go to "Custom Code" tab and put in "Source files" the absolute path to generated files.

6. Press "CTRL+B" that executes the code generation and compiles the code.

7. In the Simulink scheme choose "Simulation-¿External" option and press "Connect To Target".

8. Start the simulation to verify the controller.

SEE ALSO

mpt_demo_deployment_explicitMPC

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_DEMO_FUNCTIONS1

Demonstration of functions associated to sets

SYNTAX

```
mpt_demo_functions1
```

DESCRIPTION

Demonstration of functions associated to sets.

EXAMPLE(s)

Example 1

Constructing general functions
Create Function object $f(x) = x$.

```
F1 = Function(@(x)x)
```

Function: @(x)x

Create Function object $f(x) = x_1 - x_2^3$.

```
F2 = Function(@(x) x_(1)-x_(2)^3)
```

Function: @(x)x_(1)-x_(2)^3

Create Function with parameter K, $f(x) = -\log(Kx)$.
Since the parameter value may change, we first create the object with the parameter K.

```
F3 = Function([],struct('K',eye(2)))
```

Empty Function

Once the object has been created, we can assign the handle and refer to already stored parameter K.

```
F3.setHandle(@(x) -log(F3.Data.K*x))
```

Function: @(x)-log(F3.Data.K*x)

We can change the value of the parameter any time later.

```
F3.Data.K = 2*eye(2)
```

Function: @(x)-log(F3.Data.K*x)

Example 2

Constructing linear and affine functions
Affine map $f(x) = 6x + 1$

```
L1 = AffFunction(6,1)
```

Affine Function: R^1 ->R^1

Affine map $f(x) = -x_1 + x_2 + 1$

```
L2 = AffFunction([-1,1],1)
```

Affine Function: R^2 ->R^1

Vector function $f(x) = I_2 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

```
L3 = AffFunction(eye(2),[1;2])
```

Affine Function: R^2 ->R^2

Linear function $f(x) = I_2 x$

```
L4 = AffFunction(eye(5))
```

Affine Function: R^5 ->R^5

Example 3

Constructing quadratic functions
Quadratic map $f(x) = 0.5x^2 + 1$.

```
Q1 = QuadFunction(1,1)
```

Quadratic Function: R^1 ->R^1

Quadratic map $f(x) = x^2 - 4$.

```
Q2 = QuadFunction(2,-4)
```

Quadratic Function: R^1 ->R^1

Quadratic map $f(x) = x_1{}^2 + x_2{}^2 + 1$

```
Q3 = QuadFunction(eye(2),[0,0],1)
```

Quadratic Function: R^2 ->R^1

Example 4

Assign function to a set
Construct the polyhedron first.

```
P1 = Polyhedron('lb',-1,'ub',1)
```

```
Polyhedron in R^1 with representations:
H-rep (redundant)  : Inequalities   2 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : none
```

Assign the function to a set under under the name "a".

```
P1.addFunction(QuadFunction(4,-1),'a')
```

```
Polyhedron in R^1 with representations:
H-rep (redundant)  : Inequalities   2 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : 1 attached "a"
```

Another polyhedron

```
P2 = Polyhedron('V',[-1 1;1 1; -1 -1])
```

```
Polyhedron in R^2 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices   3 | Rays   0
Functions : none
```

Assign the function under the name "b".

```
P2.addFunction(AffFunction(-eye(2),[-1;2]),'b')
```

```
Polyhedron in R^2 with representations:
H-rep              : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices   3 | Rays   0
Functions : 1 attached "b"
```

You can assign names to functions handles.

```
P3 = Polyhedron('lb',[-1;-1],'ub',[1;1])
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)  : Inequalities   4 | Equalities   0
V-rep              : Unknown (call computeVRep() to compute)
Functions : none
```

```
P3.addFunction(Function(@(x)x.^2-x.^3+1),'gain')
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities   4 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : 1 attached "gain"
```

```
P3.addFunction(AffFunction(randn(2)),'power')
```

```
Polyhedron in R^2 with representations:
H-rep (redundant)   : Inequalities   4 | Equalities   0
V-rep               : Unknown (call computeVRep() to compute)
Functions : 2 attached "gain","power"
```

Multiple functions can be assign only at separate calls.

```
P4 = Polyhedron('V',randn(6,2));
```

```
L(1) = AffFunction(randn(2),randn(2,1));
```

```
L(2) = AffFunction(randn(2),randn(2,1));
```

```
P4.addFunction(L(1),'a')
```

```
Polyhedron in R^2 with representations:
H-rep               : Unknown (call computeHRep() to compute)
V-rep (redundant)   : Vertices   6 | Rays   0
Functions : 1 attached "a"
```

```
P4.addFunction(L(2),'b')
```

```
Polyhedron in R^2 with representations:
H-rep               : Unknown (call computeHRep() to compute)
V-rep (redundant)   : Vertices   6 | Rays   0
Functions : 2 attached "a","b"
```
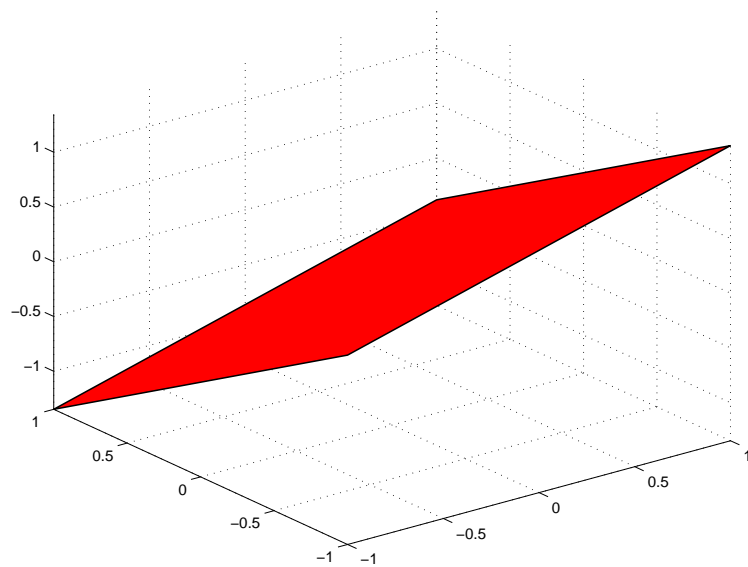
Example 5

Plotting of functions
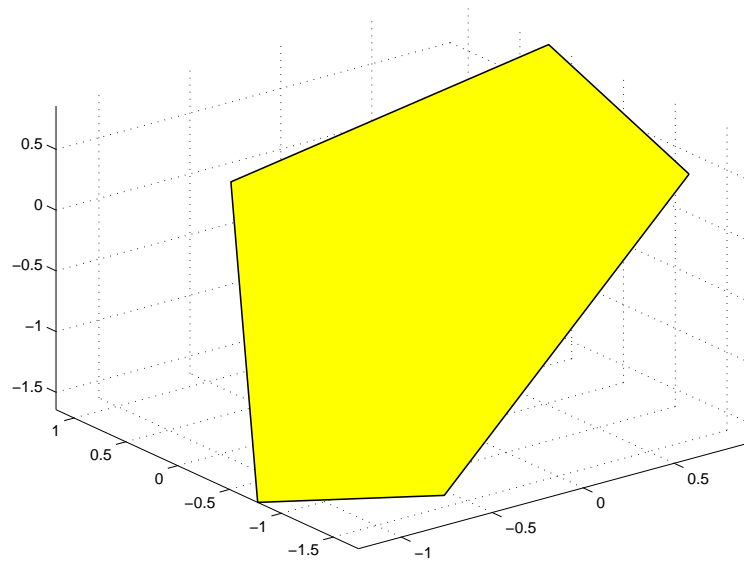Plot the function over the set based on the name
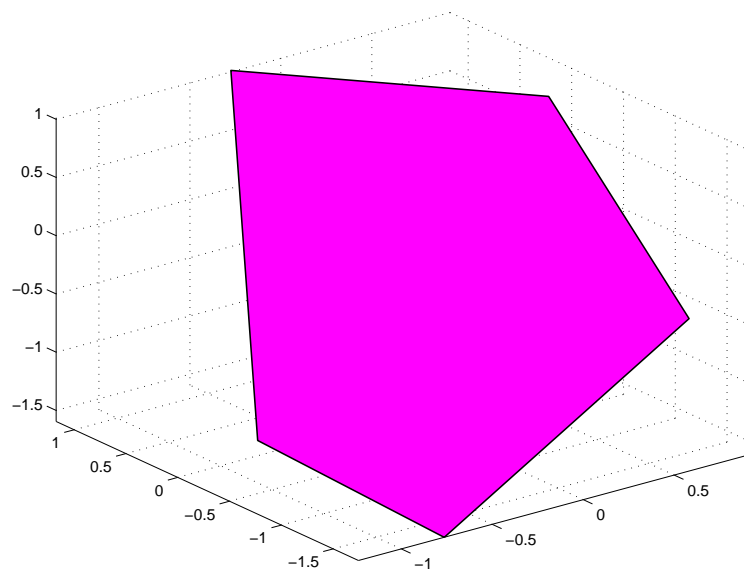
```
P3.fplot('gain')
```
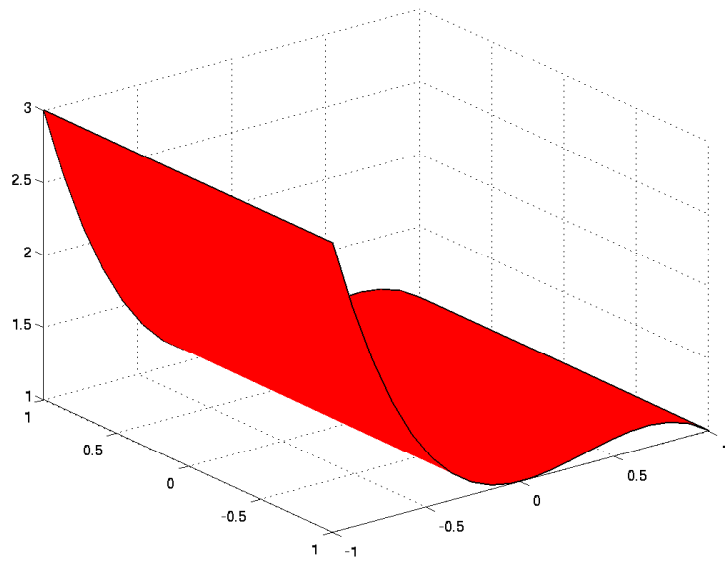
P3.fplot('power')

Plot the first element of the vector valued function "a"

P4.fplot('a','position',1,'color','y')

Plot the second element of the vector valued function "b" based on the index.

```
P4.fplot('b','position',2,'color','m')
```


Example 6

Evaluation of functions
Evaluate the function based on the name

```
P3.feval([1;0],'power')
```

617

```
ans =

0.482747657739079
0.373550607694242
```

Evaluate the function "gain"

```
P3.feval([-1;0],'gain')
```

```
ans =

3
1
```

SEE ALSO

mpt_demo_functions2

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_DEMO_LTI3

Demonstrates simulation of the closed-loop system

SYNTAX

```
mpt_demo_lti3
```

DESCRIPTION

Demonstrates simulation of the closed-loop system.

EXAMPLE(s)

Example 1

Define the prediction model $x^+ = Ax + Bu$, $y = Cx$

```
A = [1 1; 0 1]; B = [1; 0.5]; C = [1 0];
```

```
lti = LTISystem('A',A,'B',B,'C',C);
```

Define the MPC controller with horizon 5.

```
ctrl = MPCController(lti,5);
```

Define the closed-loop system

```
cl = ClosedLoop(ctrl,lti);
```

Simulate the closed loop from a given initial condition

```
data = cl.simulate([-4;0],10)
```

```
data =

X: [2x11 double]
U: [1000000 -1000000 -1000000 1000000 1000000 -1000000 -1000000 1000000 1000000 -1000000]
Y: [-4 999996 499996 -500004 -4 999996 499996 -500004 -4 999996]
cost: [0 0 0 0 0 0 0 0 0 0]
```

SEE ALSO

mpt_demo_lti1, mpt_demo_lti2, mpt_demo_lti4, mpt_demo_lti5

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

MPT_DEMO_UNIONS1

Demo that illustrates working with unions of convex sets

SYNTAX

`mpt_demo_unions1`

DESCRIPTION

Demo that illustrates working with unions of convex sets. It will be shown how to

- construct Union objects and
- add and remove sets from the Union object.

SEE ALSO

mpt_demo_unions2

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

MPT_DEMO_FUNCTIONS2

Demonstration of functions over unions of polyhedra

SYNTAX

```
mpt_demo_functions2
```

DESCRIPTION

Demonstration of functions over unions of polyhedra.

EXAMPLE(s)

Example 1

Constructing union of triangular polyhedra
Create random polyhedron

```
P = 10*ExamplePoly.randVrep
```

```
Polyhedron in R^2 with representations:
H-rep            : Unknown (call computeHRep() to compute)
V-rep (redundant)  : Vertices  10 | Rays   0
Functions : none
```

Triangulate the polyhedron to get a complex.

```
T = P.triangulate
```

```
Array of 4 polyhedra.
```

For each of the polyhedron, assign affine function

```
for i=1:numel(T)
T(i).addFunction(AffFunction(eye(2),[-1;1]),'phi');
end
```

Construct the polyunion object U.

```
U = PolyUnion('Set',T,'FullDim',true,'Bounded',true,'Overlaps',false,'Convex',true)
```
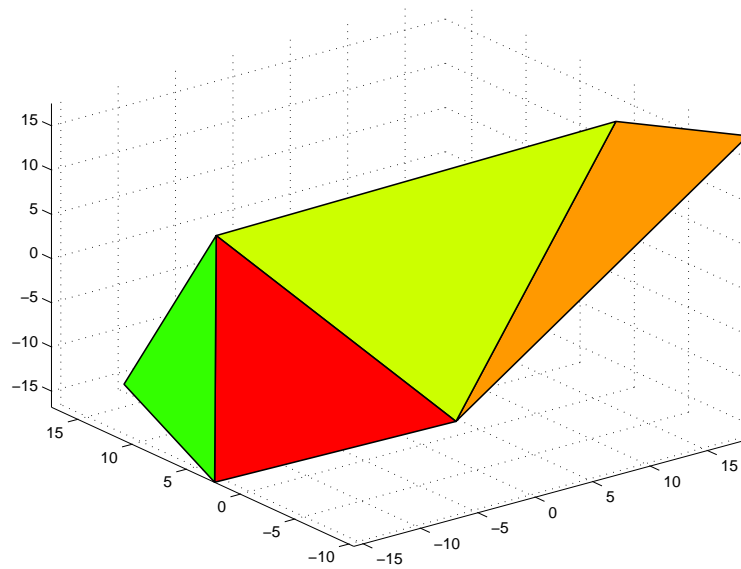
```
PolyUnion in the dimension 2 with 4 polyhedra.
Properties of the union:
Convex: 1
Overlaps: 0
Connected: 1
Bounded: 1
FullDim: 1
Functions : 1 attached "phi"
```

Plot the function over the polyhedra

```
U.fplot
```


Example 2

Construct overlapping union
Create 3 random polyhedra.

```
for i=1:3
Q(i) = ExamplePoly.randVrep+5*rand(2,1);
end
```

Assign two quadratic functions to each of the polyhedra.

```
for i=1:3
Q(i).addFunction(QuadFunction(eye(2),randn(1,2),randn(1)),'alpha');
Q(i).addFunction(QuadFunction(eye(2),randn(1,2),randn(1)),'beta');
end
```
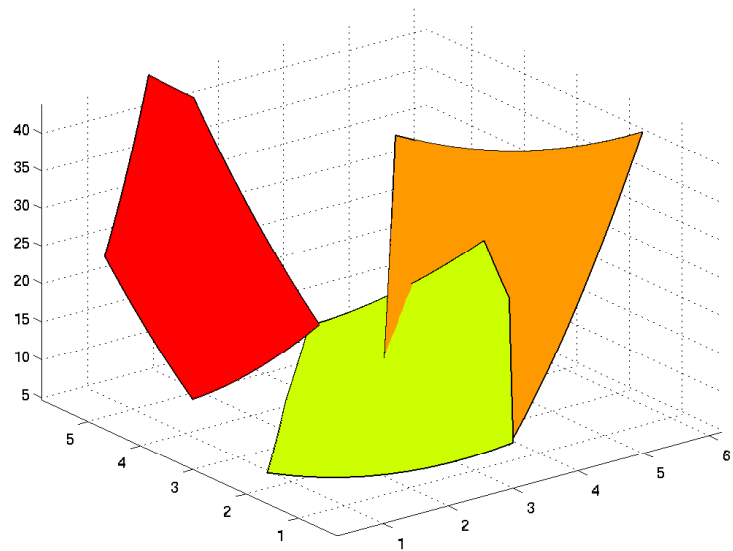
Create union without specifying any properties.

```
PU = PolyUnion(Q)
```

```
PolyUnion in the dimension 2 with 3 polyhedra.
Functions : 2 attached "alpha","beta"
```
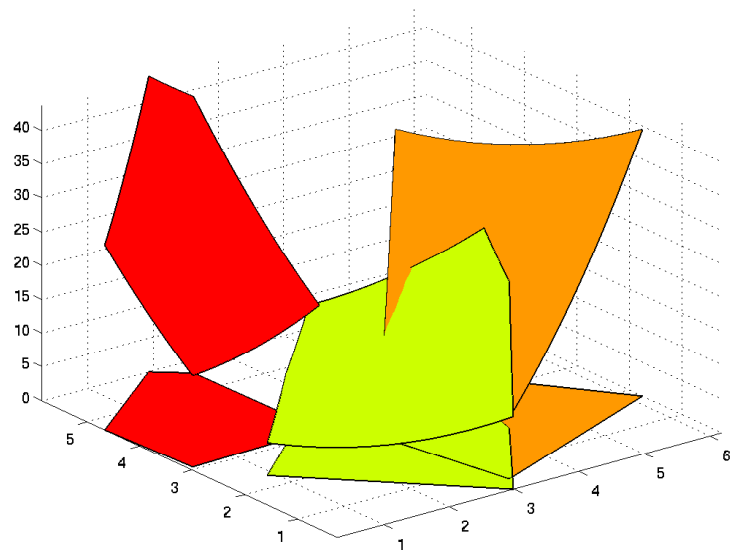
Plot the functions over polyhedra.

PU.fplot('beta')

Plot the functions over polyhedra based with some properties

PU.fplot('beta','show_set',true)

SEE ALSO

mpt_demo_functions1

AUTHOR(s)

MPT_DEMO_DEPLOYMENT_ONLINEMPC

Application of online MPC controller with the help of Simulink interface

SYNTAX

```
mpt_demo_deployment_onlineMPC
```

DESCRIPTION

Demonstration of real-time control using online MPC controller. The demo relies on standalone LCP solver that solves the optimization problem given as linear-complementarity problem (LCP). The LCP solver must be present in the installation path including the extended version **lcprtw** that contains pre-compiled libraries for linking with OpenWatcom compiler. The demo has been tested under Real-Time Windows target and XPC target on Windows 32-bit platform.

Note that this demo can be compiled and run on Windows 32-bit platform for Matlab r2012a!

Deployment steps:

1. Generate the online controller that has the desired properties.

2. Export the online controller to YALMIP using `toYalmip` method of `MPCController` class.

3. Formulate a parametric optimization problem and specify the feedback variables that are used as parameters.

4. Create an instance of `Opt` class and transform the optimization problem to LCP using `qp2lcp` method.

5. Create a Simulink scheme with the S-Function block that links to LCP solver.

6. In the Simulink scheme, choose code generation options and pick **rtwin.tlc** as the system target file that corresponds to Real-Time Windows target or **xpctarget.tlc** for XPC target.

7. Press "CTRL+B" that executes the code generation and compiles the code.

8. In the Simulink scheme choose "Simulation->External" option and press "Connect To Target".

9. Start the simulation to verify the controller in real-time.

SEE ALSO

mpt_demo_deployment_explicitMPC

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_DEMO_SETS1

Demonstrates construction and basic properties of the Polyhedron object

SYNTAX

```
mpt_demo_sets1
```

DESCRIPTION

This demo presents the basic usage for the `Polyhedron` object. In particular it is shown how to:

- create `Polyhedron` objects
- access data stored in the `Polyhedron` object
- query basic properties of the `Polyhedron` set

SEE ALSO

mpt_demo_sets2, mpt_demo_sets3

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_DEMO_LTI5

Demostration of problem formulation using additional properties.

SYNTAX

```
mpt_demo_lti5
```

DESCRIPTION

Formulation of MPC problem using terminal cost, terminal set constraints, and move blocking constraint.

EXAMPLE(s)

Example 1

Define an LTI prediction model

```
lti = LTISystem('A',[1 1; 0 1],'B',[1; 0.5]);
```

Define the MPC controller

```
horizon = 5;
```

```
ctrl = MPCController(lti,horizon);
```

Define quadratic penalties

```
ctrl.model.x.penalty = QuadFunction(eye(2));
```

```
ctrl.model.u.penalty = QuadFunction(1);
```

```
onl_ctrl.model.x.terminalPenalty = QuadFunction(10*eye(2));
```

Add a terminal set constraint (see `help SystemSignal/filter_terminalSet`)

```
ctrl.model.x.with('terminalSet');
```

```
ctrl.model.x.terminalSet = Polyhedron('lb',[-1; -1],'ub',[1; 1]);
```

Add an LQR terminal penalty (see `help SystemSignal/filter_terminalPenalty`)

```
lqr_penalty = ctrl.model.LQRPenalty();
```

```
ctrl.model.x.with('terminalPenalty');
```

```
ctrl.model.x.terminalPenalty = lqr_penalty;
```

Add a move-blocking constraint (the last 3 moves are to be constant)

```
ctrl.model.u.with('block');
```

```
ctrl.model.u.block.from = ctrl.N-2;
```

```
ctrl.model.u.block.to = ctrl.N;
```

Obtain the optimal control input

```
x0 = [-4; 0];
[Uonl,feasible] = ctrl.evaluate(x0)


Uonl =

2.07136913674103


feasible =

1
```

We can also ask for full open-loop predictions:
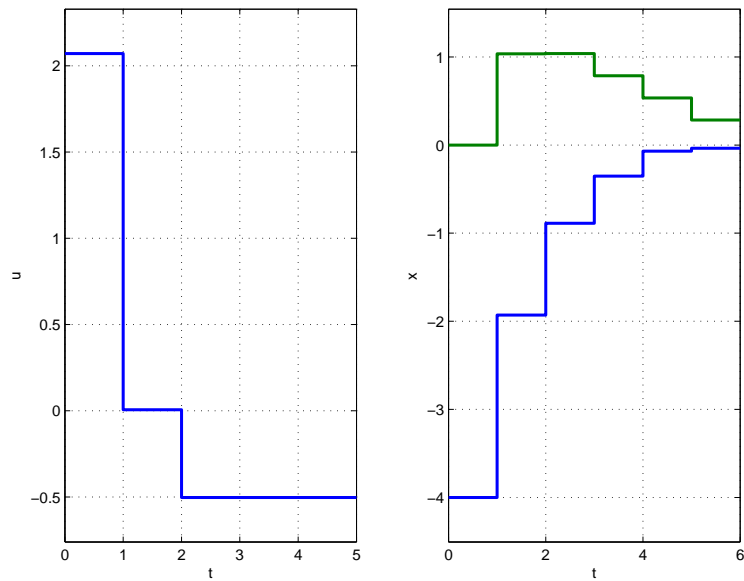
```
[~,~,openloop] = ctrl.evaluate(x0)


openloop =

cost: 28.9534609997889
U: [2.07136913674103 0.00571336432319767 -0.503441481063561 -0.503441481063562 -0.503441481063562]
X: [2x6 double]
Y: [0x5 double]
```

Plot the open-loop predictions

```
ctrl.model.plot
```

SEE ALSO

mpt_demo_lti1, mpt_demo_lti2, mpt_demo_lti3, mpt_demo_lti4

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
    mailto:michal.kvasnica@stuba.sk

MPT_DEMO2

Tour through visualization capabilities of the toolbox

SYNTAX

```
mpt_demo2
```

DESCRIPTION

Demonstration how to plot basic sets and explanation of various plotting options.

SEE ALSO

mpt_demo1, mpt_demo_sets1, mpt_demo_sets2

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
   mailto:michal.kvasnica@stuba.sk

© **2010-2013** Martin Herceg: ETH Zurich
   mailto:herceg@control.ee.ethz.ch

MPT_DEMO_SETS3

Demo that illustrates operations on polyhedra

SYNTAX

`mpt_demo_sets3`

DESCRIPTION

Demo that illustrates geometric operations with polyhedra. In particular, it is shown how to

- convert between H- and V-representation

- intersection of polyhedra

- comparisons between polyhedra

- set algebra: affine maps, set summation, set difference

SEE ALSO

mpt_demo_sets1, mpt_demo_sets2

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
mailto:herceg@control.ee.ethz.ch

MPT_DEMO_DEPLOYMENT_EXPLICITMPC

Application of explicit MPC controller with the help of Simulink interface

SYNTAX

`mpt_demo_deployment_explicitMPC`

DESCRIPTION

Demonstration of real-time control using explicit MPC controller. This demo shows how to design an explicit controller for usage in real-time control in Real-Time Workshop. The code generation and compilation for real-time has been tested under Real-Time Windows target on Windows 32-bit and 64-bit platforms.

Deployment steps:

1. Generate the explicit controller that has the desired properties.

2. Export the explicit controller to C-code using `mpt_exportToC` function.

3. Create a Simulink scheme with the S-Function block that represents generated controller.

4. In the Simulink scheme, choose code generation options and pick **rtwin.tlc** as the system target file (e.g. for Real-Time Windows target).

5. In the code generation options, go to "Custom Code" tab and put in "Source files" the absolute path to generated files.

6. Press "CTRL+B" that executes the code generation and compiles the code.

7. In the Simulink scheme choose "Simulation-¿External" option and press "Connect To Target".

8. Start the simulation to verify the controller.

SEE ALSO

mpt_demo_deployment_onlineMPC

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
    mailto:herceg@control.ee.ethz.ch

MPT_DEMO_LTI2

Demonstrates online MPC for LTI system

SYNTAX

```
mpt_demo_lti2
```

DESCRIPTION

Demonstrates online MPC for LTI system.

EXAMPLE(s)

Example 1

Define an LTI prediction model $x^+ = Ax + Bu$

```
A = [1 1; 0 1]; B = [1; 0.5];
```

```
lti = LTISystem('A',A,'B',B);
```

Alternativelly, we can import from state-space objects:
```
s = ss(tf(1, [1 0 0]));
d = c2d(s, 1);
lti = LTISystem(d);
```
Define the MPC controller

```
horizon = 5;
```

```
ctrl = MPCController(lti,horizon);
```

Add state constraints

```
ctrl.model.x.min = [-5; -5];
```

```
ctrl.model.x.max = [5; 5];
```

Add input constraints

```
ctrl.model.u.min = -1;
```

```
ctrl.model.u.max = 1;
```

Set quadratic state penalty

```
ctrl.model.x.penalty = QuadFunction(eye(2));
```

Set quadratic input penalty

```
ctrl.model.u.penalty = QuadFunction(1);
```

Obtain the optimal control input for a given initial condition.

```
x0 = [-4; 0]; Uonl = ctrl.evaluate(x0)
```

```
Uonl =

1
```

SEE ALSO

mpt_demo_lti1, mpt_demo_lti3, mpt_demo_lti4, mpt_demo_lti5

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
mailto:michal.kvasnica@stuba.sk

MPT_DEMO_OPT1

Demonstration for using Opt interface

SYNTAX

```
mpt_demo_opt1
```

DESCRIPTION

Demonstration for using Opt interface for solving optimization problems

EXAMPLE(s)

Example 1

Formulate LP problem min $f^T x$ s.t.: $Ax \leq b$.

```
f = randn(8,1);

A = randn(25,8);

b = 5*rand(25,1);
```

Formulate LP

```
problem1 = Opt('f',f,'A',A,'b',b)
```

```
--------------------------------------------------
Linear program
Num variables:                8
Num inequality constraints:   25
Num equality constraints:     0
Solver:                       LCP
--------------------------------------------------
```

Solve LP

```
res1 = problem1.solve
```

```
res1 =

xopt: [8x1 double]
lambda: [1x1 struct]
obj: -7.22865124839414
how: 'ok'
exitflag: 1
```

Example 2

Formulate MPLP problem min $f^T x$ s.t.: $Ax \leq b + B\theta$.
Do not forget to include bounds on the parameters $-1 \leq \theta \leq 1$.
Formulate MPLP

```
problem2 = Opt('f',f,'A',A,'b',b,'pB',ones(25,1),'Ath',[-1;1],'bth',[1;1])
```

```
-------------------------------------------------
Parametric linear program
Num variables:               8
Num inequality constraints:  25
Num equality constraints:    0
Num parameters:              1
Solver:                      PLCP
-------------------------------------------------
```

Solve MPLP

```
res2 = problem2.solve
```

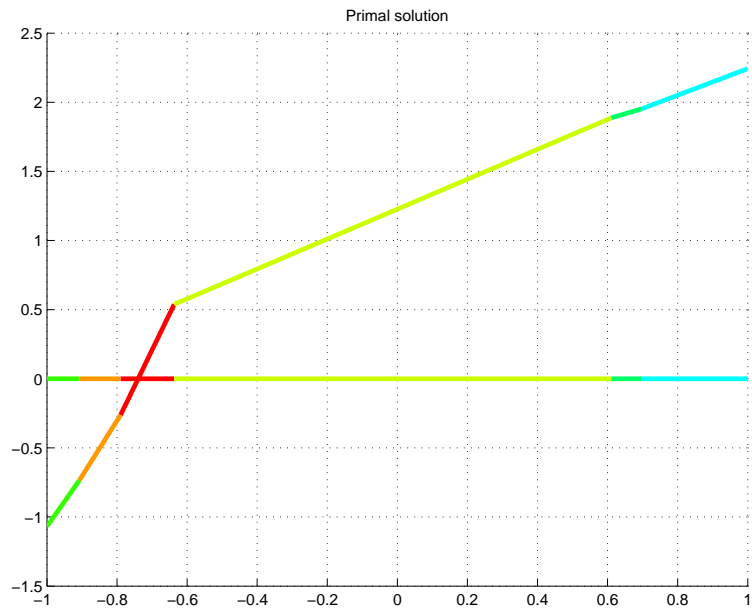```
mpt_plcp: 6 regions

res2 =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Solution is stored as "primal", we can plot it

```
res2.xopt.fplot('primal','show_set',true,'LineWidth',3),title('Primal solution')
```

Primal solution

Example 3

Formulate MPLP problem using MPT2-solver
Formulate MPLP with MPT2-MPLP solver

```
problem3 = Opt('f',f,'A',A,'b',b,'pB',ones(25,1),'Ath',[-1;1],'bth',[1;1],'solver','MPLP')
```

```
--------------------------------------------------
Parametric linear program
Num variables:               8
Num inequality constraints:  25
Num equality constraints:    0
Num parameters:              1
Solver:                      MPLP
--------------------------------------------------
```

Solve MPLP using MPT2
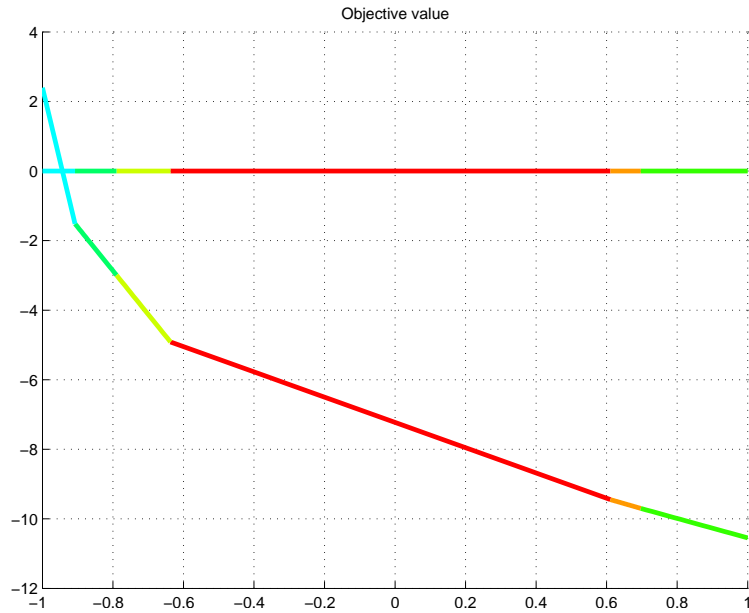
```
res3 = problem3.solve

Calling mpt_mplp_26 with default options...
mpt_mplp: 6 regions

res3 =

xopt: [1x1 PolyUnion]
mplpsol: [1x1 struct]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Plot the objective value

```
res3.xopt.fplot('obj','show_set',true,'LineWidth',3),title('Objective value')
```


Example 4

Formulate problem using YALMIP
Model data

```
A = [0.5 -1;1 0]; B = [1;0]; nu = 1;
```

MPC data

```
Q = eye(2); R = 1; N = 4;
```

Initial state

```
x0 = sdpvar(2,1);
```

Setup the problem

```
u = sdpvar(nu,N);
```

```
constraints = [];
```

```
objective = 0;
```

```
x = x0;
```

```
for k = 1:N
x = A*x + B*u(k);
objective = objective + norm(Q*x,1) + norm(R*u(k),1);
constraints = [constraints,-1 <= u(k) <= 1,-5 <= x <= 5];
end
```

Formulate the problem

```
problem4 = Opt(constraints,objective,x0,u)
```

```
--------------------------------------------------
Parametric linear program
Num variables:              24
Num inequality constraints: 56
Num equality constraints:    0
Num lower bounds            24
Num upper bounds            24
Num parameters:              2
Solver:                    PLCP
--------------------------------------------------
```
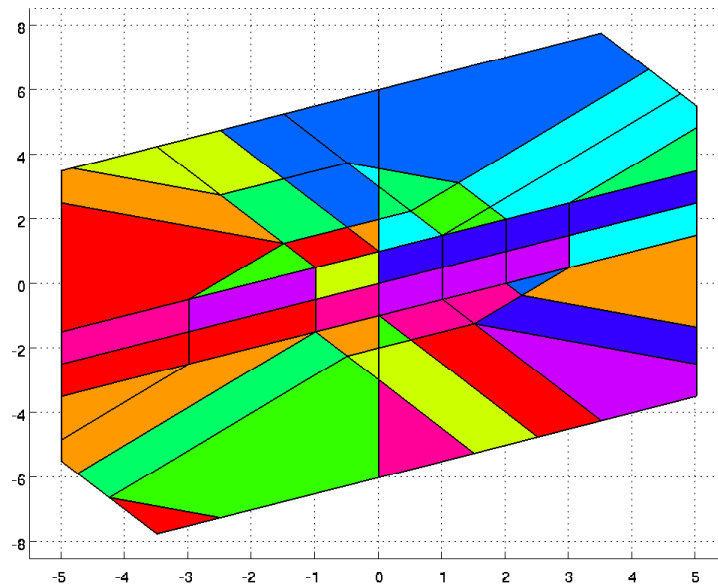
Solve the problem

```
res4 = problem4.solve
```

```
regions:     8,unexplored: 11
regions:    24,unexplored: 12
mpt_plcp: 52 regions

res4 =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Plot the partition

```
res4.xopt.plot
```

Example 5

Formulate problem using MPT2
Process model

```
sysStruct.A = [1 1; 0 1];
sysStruct.B = [1; 0.5];
sysStruct.C = [1 0; 0 1];
sysStruct.D = [0;0];
sysStruct.ymin = [-5; -5];
sysStruct.ymax = [5; 5];
sysStruct.umin = -1;
sysStruct.umax = 1;
```

Problem formulation

```
probStruct.norm=2;
probStruct.Q=eye(2);
probStruct.R=1;
probStruct.N=5;
probStruct.subopt_lev=0;
```

Generate matrices of the appropriate formulation of parametric problem.

```
Matrices = mpt_constructMatrices(sysStruct,probStruct);
```

```
Function mpt_constructMatrices is obsolete and will be removed in a future MPT version.
Iteration 1...
Iteration 2...
Iteration 3...
Iteration 4...
```

Define the problem using `Opt` class.

```
problem5 = Opt(Matrices)
```

```
--------------------------------------------------
Parametric quadratic program
Num variables:                   5
Num inequality constraints:  20
Num equality constraints:     0
Num parameters:                  2
Solver:                       PLCP
--------------------------------------------------
```

Solve the problem

```
res5 = problem5.solve
```
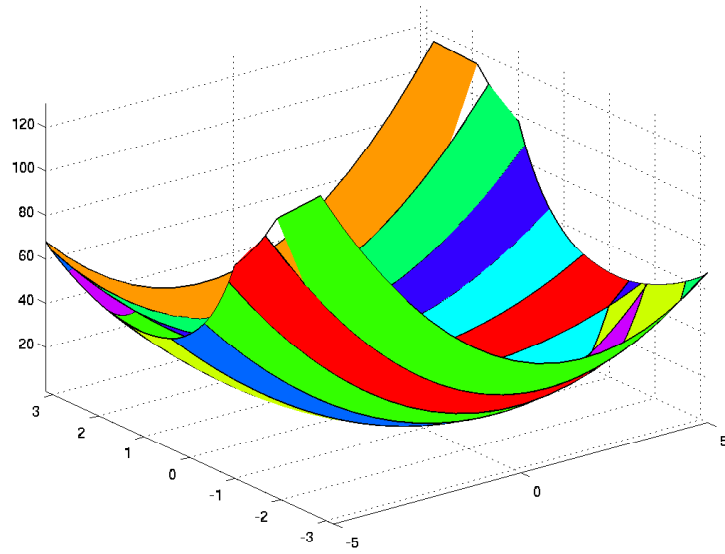
```
mpt_plcp: 25 regions

res5 =

xopt: [1x1 PolyUnion]
exitflag: 1
how: 'ok'
stats: [1x1 struct]
```

Plot he objective function

```
res5.xopt.fplot('obj')
```

AUTHOR(s)

© **2010-2013** Martin Herceg: ETH Zurich
  mailto:herceg@control.ee.ethz.ch

MPT_DEMO_LTI1

Simulation of LTISystem

SYNTAX

```
mpt_demo_lti1
```

DESCRIPTION

Simulation of LTI systems.

EXAMPLE(s)

Example 1

Define an LTI system $x^+ = Ax + Bu$, $y = Cx$

```
A = [1 1; 0 1]; B = [1; 0.5]; C = [1 0];
```

```
lti = LTISystem('A',A,'B',B,'C',C)
```

LTISystem with 2 states,1 input,1 output

Set the initial state of the system

```
lti.initialize([1; 1.5]);
```

Ask for the states

```
x = lti.getStates()
```

x =

1
1.5

Update the system's state using some control action.

```
u = 0.5;
```

The following updates the internal state

```
lti.update(u);
```

Ask for the updated states

```
x = lti.getStates()
```

```
x =

3
1.75
```

The state-update and current output can also be directly obtained from update():

```
u = -0.6;

[next_x,current_y] = lti.update(u)
```

```
next_x =

4.15
1.45
```

```
current_y =

3
```

SEE ALSO

mpt_demo_lti2, mpt_demo_lti3, mpt_demo_lti4, mpt_demo_lti5

AUTHOR(s)

© **2003-2013** Michal Kvasnica: STU Bratislava
     mailto:michal.kvasnica@stuba.sk