pypolycontain

Release v1.3-6-g05ecfd8

Sadra Sadraddini

2021-01-20

Contents:

pypolycontain is a python package for polytopic objects, operations, and polytope containment problems. It is written as part of a project for verification and control of hybrid systems.

1 Setup

Installation is now easy:

```
pip install pypolycontain
```

Or download[1], and:

```
python3 setup.py install
```

2 Dependencies:

- numpy[2] (Use latest version)

2.1 Optional dependencies (for some features)

- Drake[3] (Use latest version)
- pycdd[4] (Use latest version)
- Gurobi[5] (Version 8.0.1 or later) *Free Academic License*
- scipy[6] (Use latest version)

2.1.1 Examples

Getting Started

The simplest tasks are `pypolycontain` is defining polytopic objects, performing operations, and visualizing them. First, import the package alongside `numpy`.

[1]:
```
import numpy as np
import pypolycontain as pp
```

Objects

H-polytope

We define an H-polytope $\mathbb{P} = \{x \in \mathbb{R}^2 | Hx \leq h\}$. We give the following numericals for H and h:

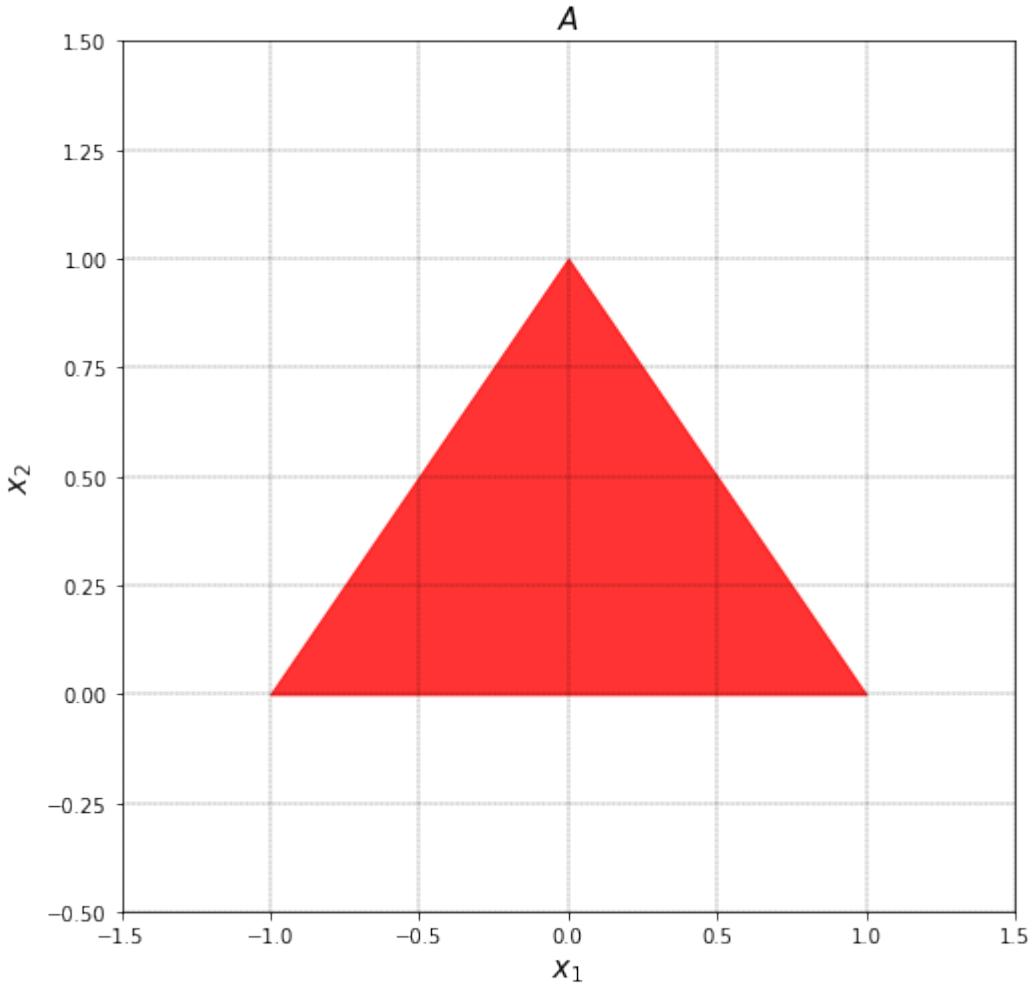
$$H = \begin{pmatrix} 1 & 1 \\ -1 & 1 \\ 0 & -1 \end{pmatrix}, h = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}.$$

[1] https://github.com/sadraddini/pypolycontain/archive/master.zip
[2] https://numpy.org/
[3] https://drake.mit.edu/
[4] https://pycddlib.readthedocs.io/en/latest/index.html
[5] https://gurobi.com
[6] https://scipy.org//

[2]:
```
H=np.array([[1,1],[-1,1],[0,-1]])
h=np.array([1,1,0])
A=pp.H_polytope(H,h)
```

This a triangle as it is defined by intersection of 3 half-spaces in $\mathbb{R}^2$. In order to visualizate the polytope, we call the following function. Note the brackets around `visualize` function - it takes in a list of polytopes as its primary argument.

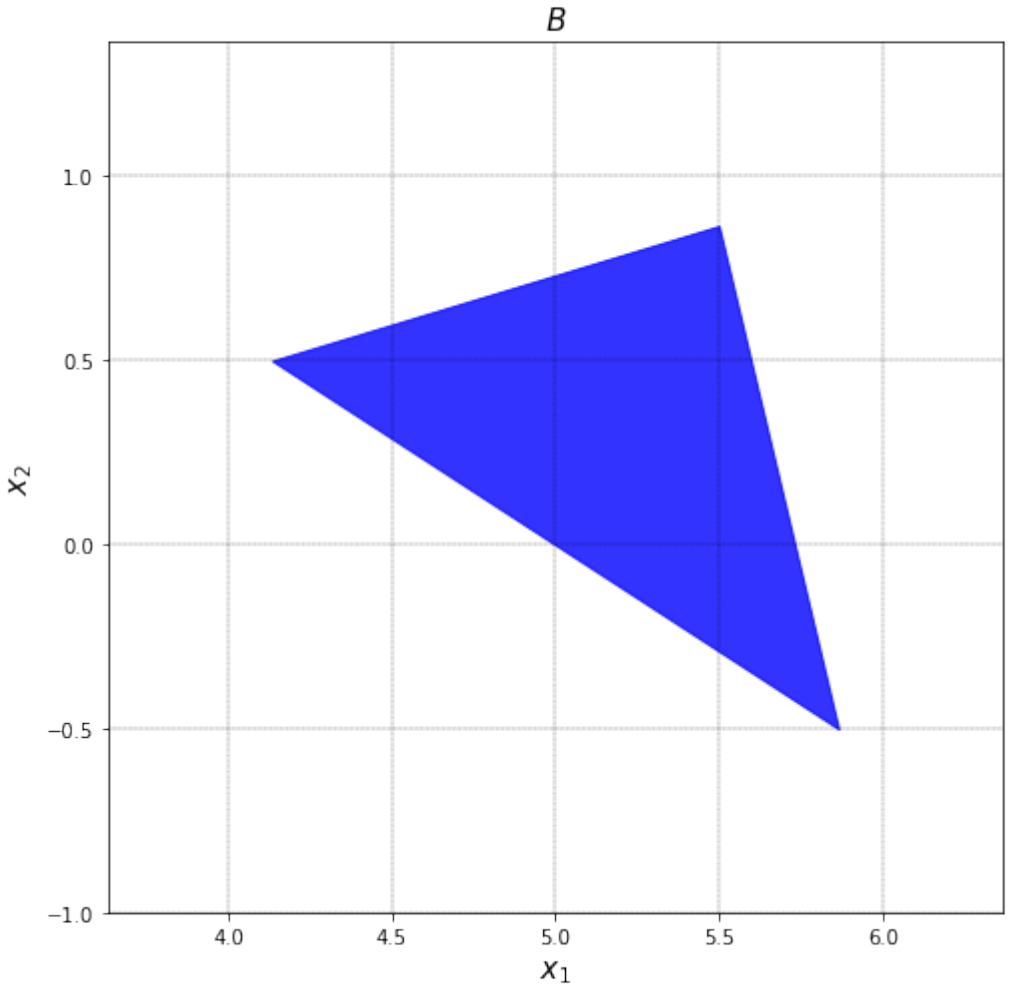
[3]:
```
pp.visualize([A],title=r'$A$')
```



AH-polytope

We define an AH-polytope as $t + T\mathbb{P}$ with the following numbers. The transformation represents a rotation of 30° and translation in x direction.

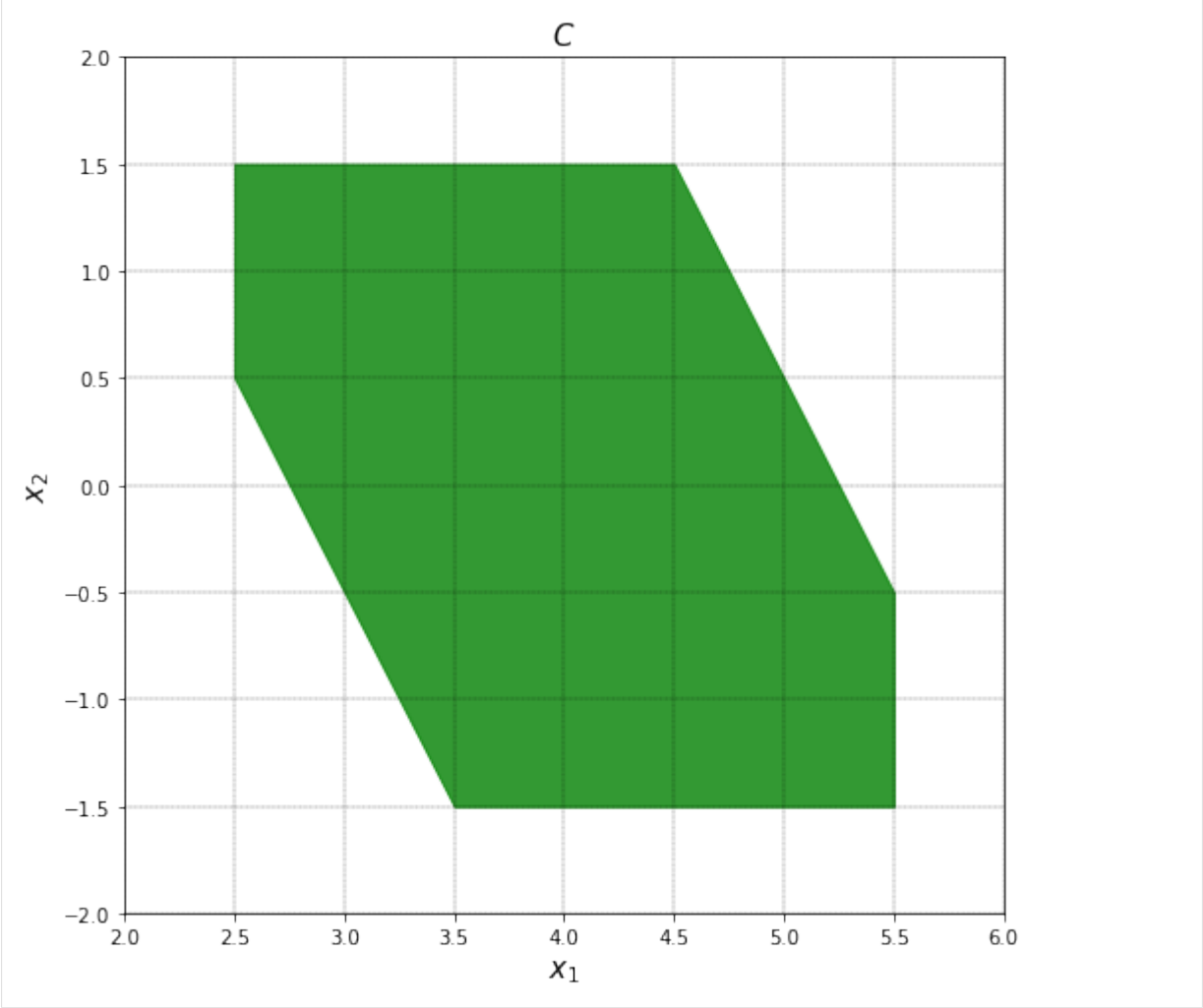
[4]:
```
t=np.array([5,0]).reshape(2,1) # offset
theta=np.pi/6 # 30 degrees
T=np.array([[np.cos(theta),np.sin(theta)],[-np.sin(theta),np.cos(theta)]]) # Linear␣
↪transformation
B=pp.AH_polytope(t,T,A)
pp.visualize([B],title=r'$B$')
```



Zonotope

We define a zonotope as $\mathbb{Z} = x + G[-1, 1]^{n_p}$, where n_p is the number of rows in p.

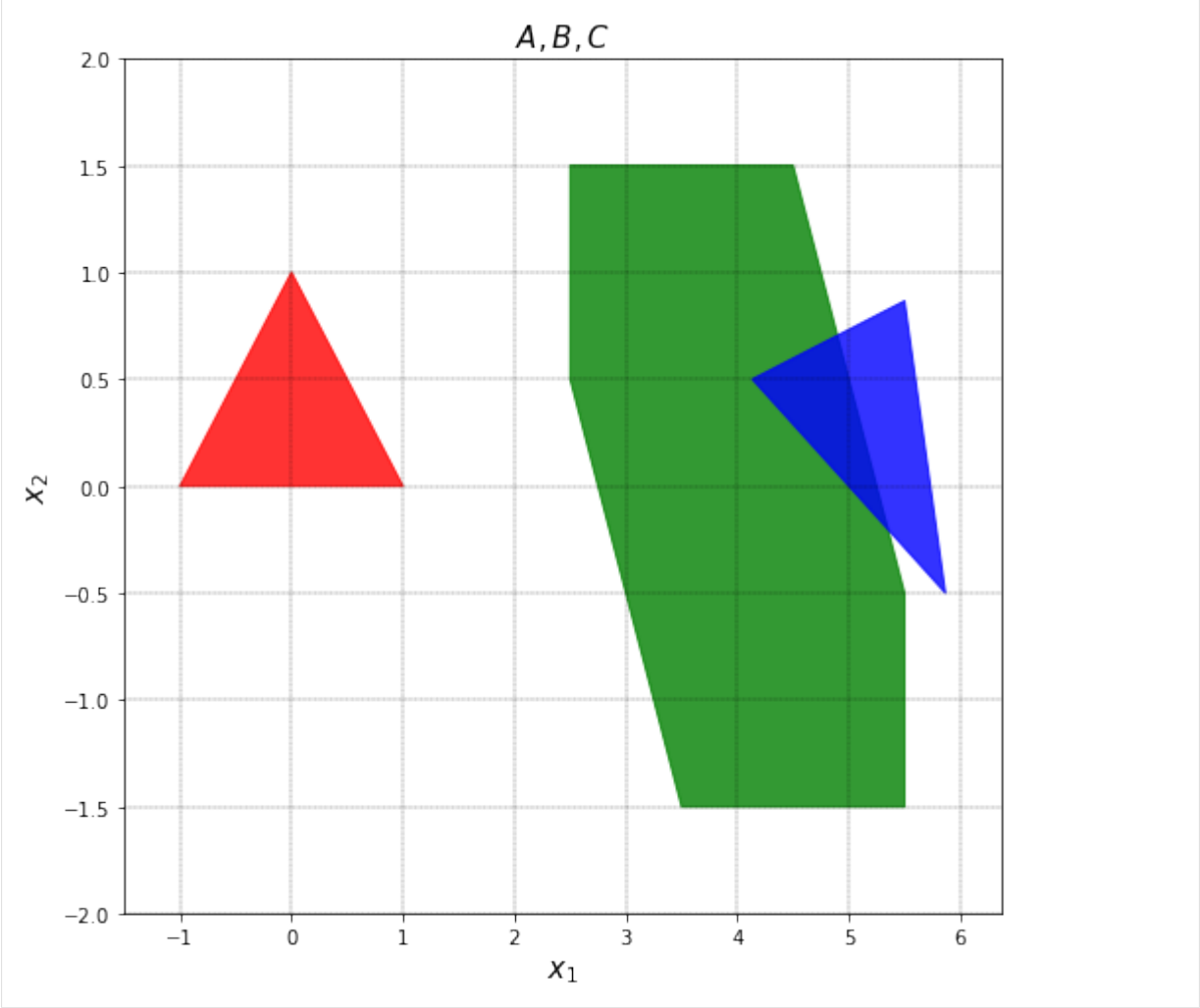
[5]:
```
x=np.array([4,0]).reshape(2,1) # offset
G=np.array([[1,0,0.5],[0,0.5,-1]]).reshape(2,3)
C=pp.zonotope(x=x,G=G)
pp.visualize([C],title=r'$C$')
```



Visualization

The `visualize` function allows for visualizing multiple polytopic objects.

[6]:
```
pp.visualize([A,C,B],title=r'$A,B,C$')
```



You may have noticed the colors. Here are the default colors for various polytopic objects. Using color argument you can change the color.

Object	Default Color
H-polytope	Red
Zonotope	Green
AH-polytope	Blue

Options

`visualize` has a set of options. While it only supports 2D plotting, it can take high dimensional polytopes alongside the argument `list_of_dimensions`. Its `default` is `[0,1]`, meaning the projection into the first and second axis is demonstrated.

You can also add a separate `subplot` environment to plot the polytopes. Take the following example:

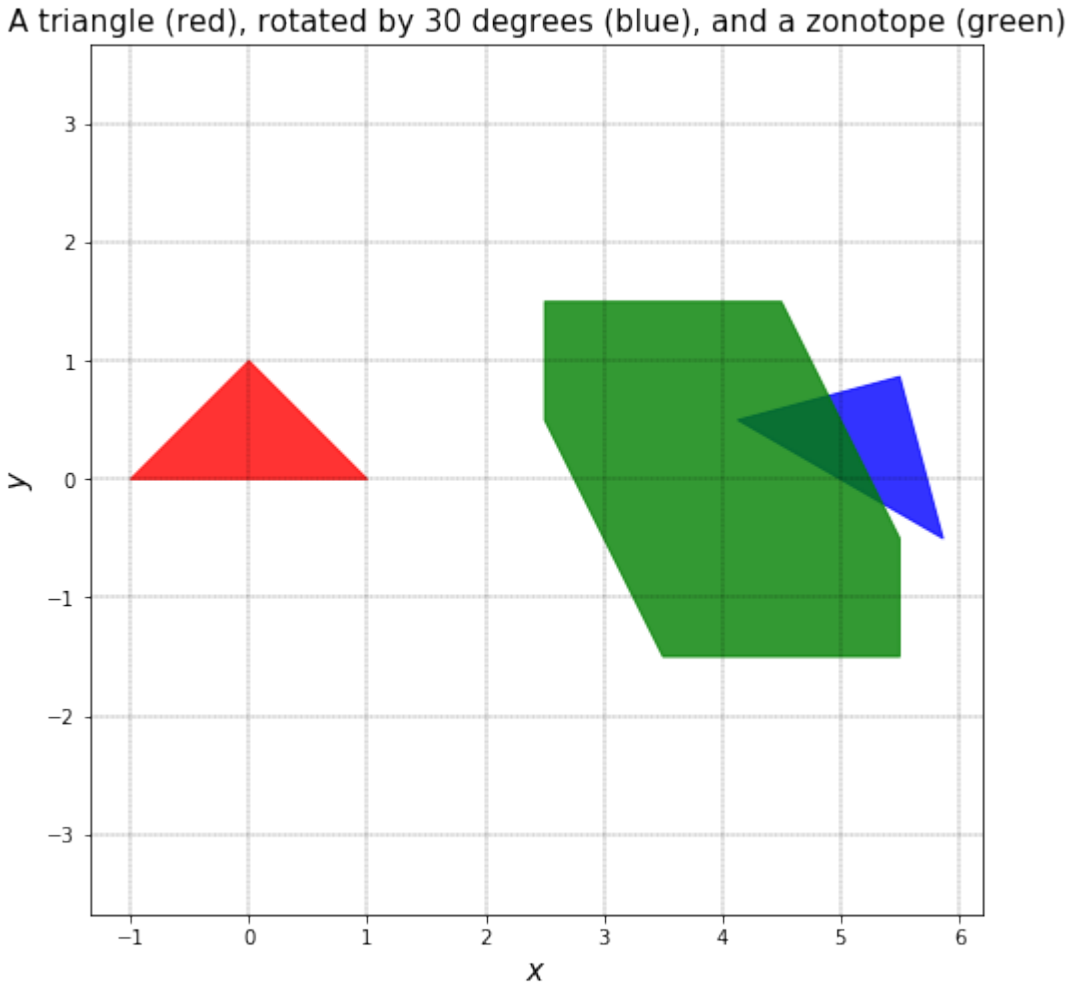
[7]:
```
import matplotlib.pyplot as plt
fig,ax=plt.subplots()
fig.set_size_inches(6, 3)
pp.visualize([A,B,C],ax=ax,fig=fig)
ax.set_title(r'A triangle (red), rotated by 30 degrees (blue), and a zonotope (green)',
 ↪FontSize=15)
ax.set_xlabel(r'$x$',FontSize=15)
```

(continues on next page)

(continued from previous page)

```
ax.set_ylabel(r'$y$',FontSize=15)
ax.axis('equal')
```

[7]:
```
(-1.343301270189222,
 6.209326673973662,
 -1.6499999999999997,
 1.6499999999999997)
```



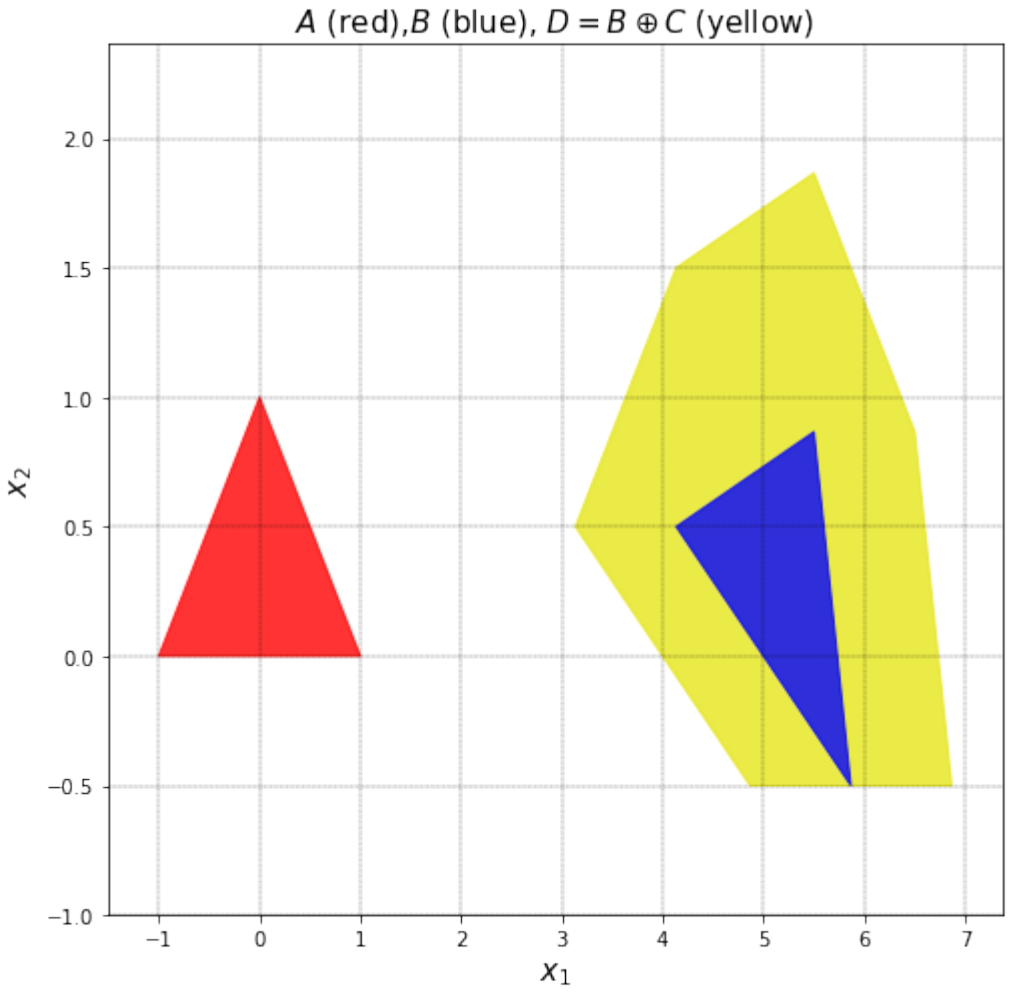
Operations

`pypolycontain` supports a broad range of polytopic operations. The complete list of operations is here[7].

Minkowski sum

[8]:
```
D=pp.operations.minkowski_sum(A,B)
D.color=(0.9, 0.9, 0.1)
pp.visualize([D,A,B],title=r'$A$ (red),$B$ (blue), $D=B\oplus C$ (yellow)')
```

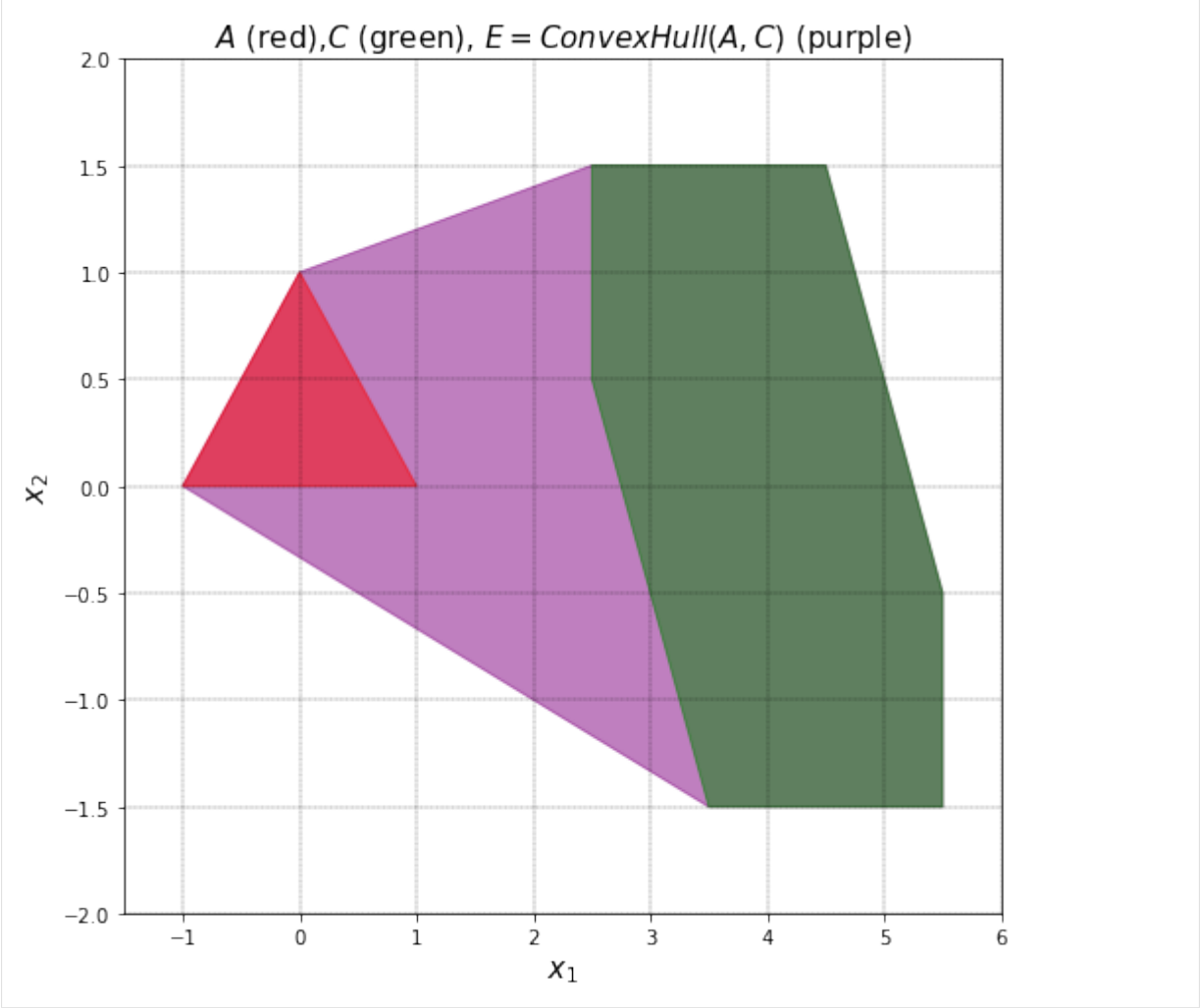
[7] https://pypolycontain.readthedocs.io/en/latest/operations.html



Convex Hull

Let's take the intersection of A and C.

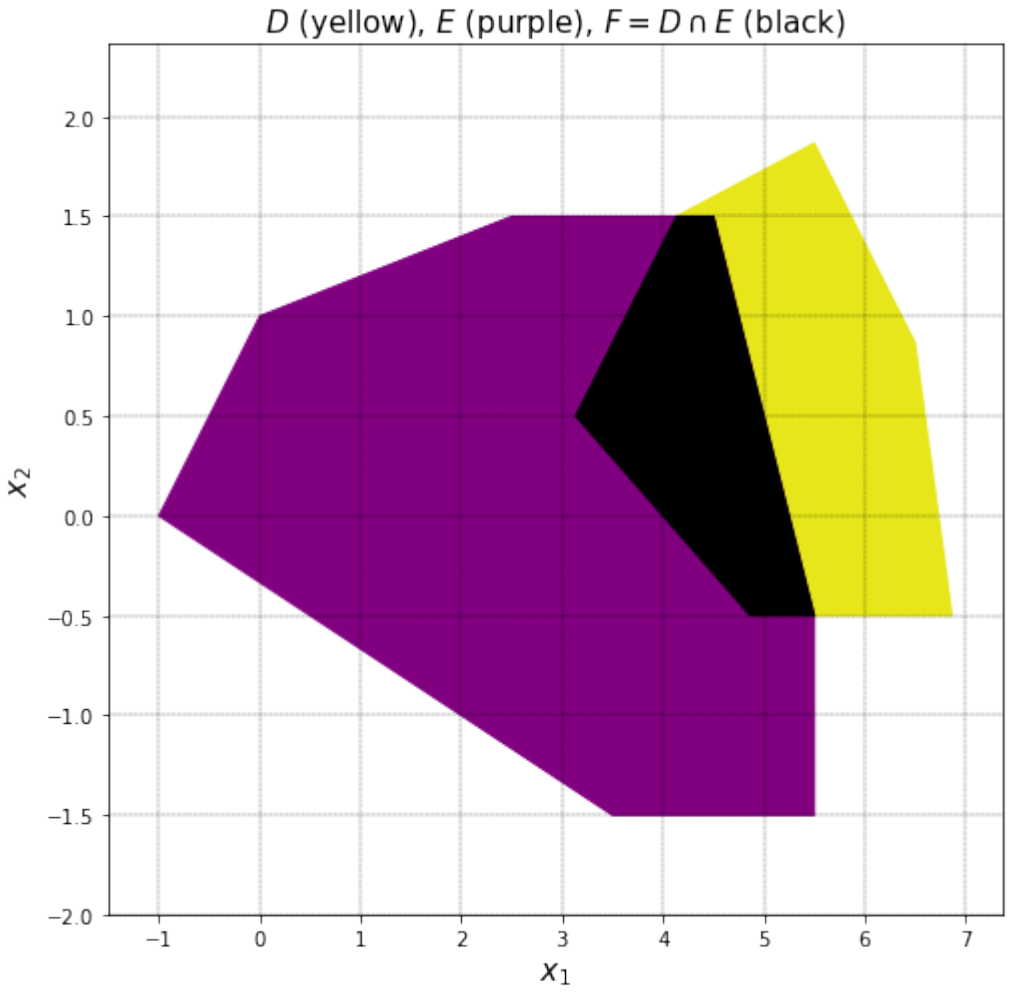
[9]:
```
E=pp.convex_hull(A,C)
E.color='purple'
pp.visualize([E,A,C],alpha=0.5,title=r'$A$ (red),$C$ (green), $E={ConvexHull}(A,C)$␣
↪(purple)')
```



Intersection

Let's take the intersection of B and D.

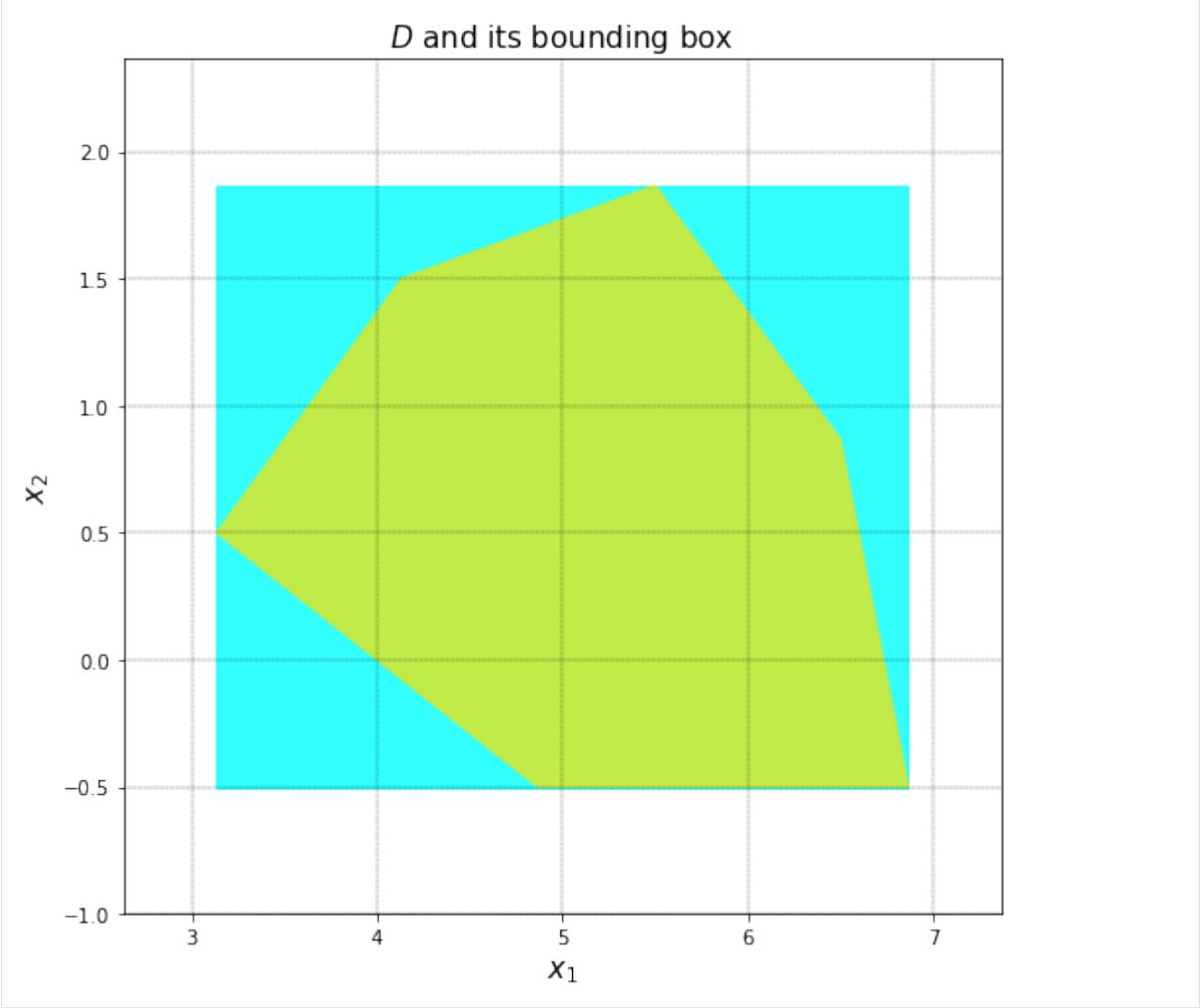
[10]:
```
F=pp.intersection(D,E)
F.color='black'
pp.visualize([D,E,F],alpha=1,title=r'$D$ (yellow), $E$ (purple), $F=D \cap E$ (black)')
```



Bounding Box

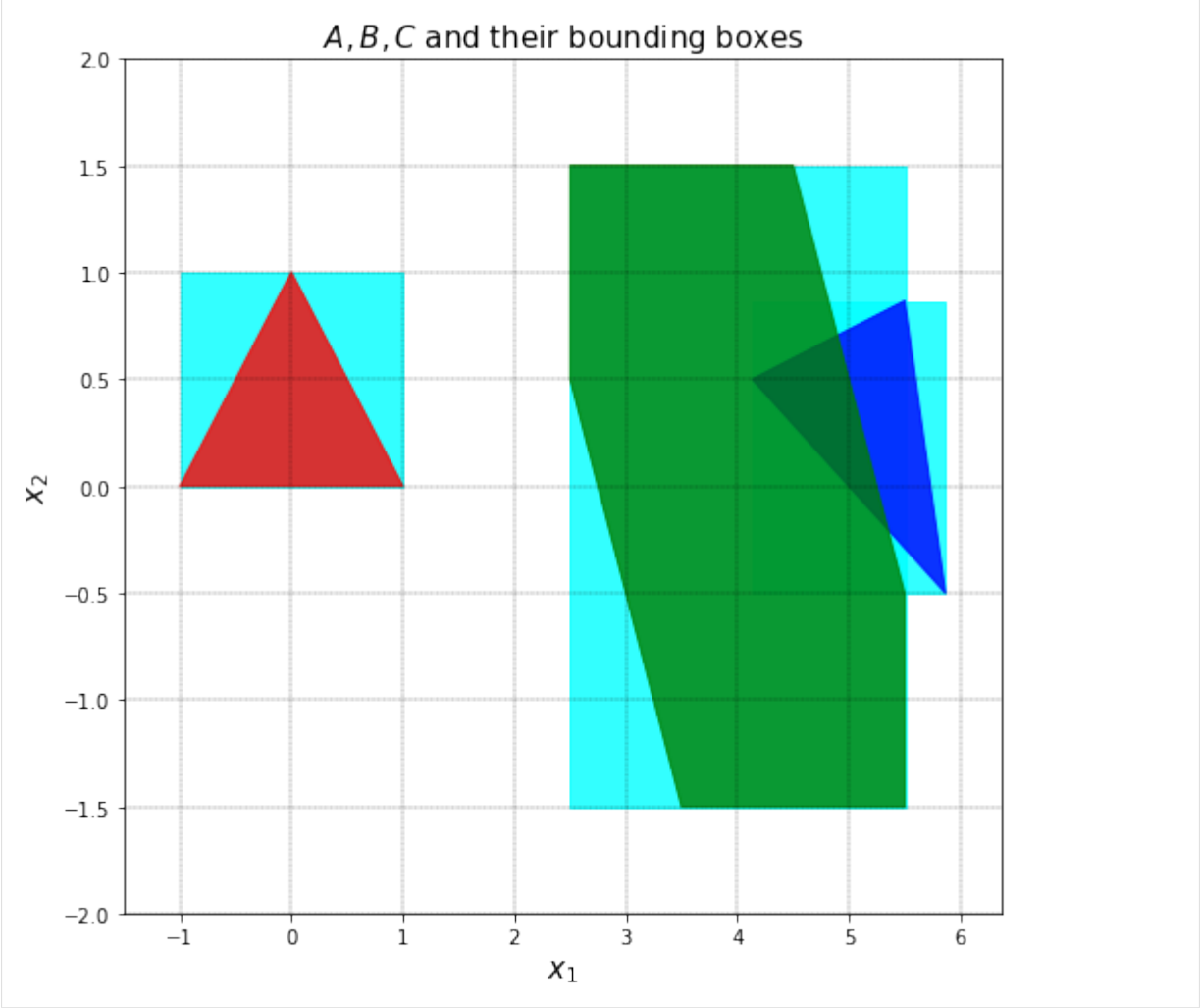
We can compute the bounding box of a polytopic object. For instance, let's compute the bounding box of D.

[11]:
```
G=pp.bounding_box(D)
pp.visualize([G,D],title=r'$D$ and its bounding box')
```



Or the following:

[12]:
```
mylist=[A,B,C]
pp.visualize([pp.bounding_box(p) for p in mylist]+mylist,title=r'$A,B,C$ and their␣
↪bounding boxes')
```



2.1.2 Objects

pypolycontain has the following polytopic objects.

H-polytope

class pypolycontain.objects.**H_polytope**(*H*, *h*, *symbolic=False*, *color='red'*)

An H-polytope is a set defined as follows:

$$\mathbb{P} = \{x \in \mathbb{R}^n | Hx \leq h\},$$

where

Attributes:

- $H \in \mathbb{R}^{q \times n}$: *numpy.ndarray[float[[q,n]]*
- $h \in \mathbb{R}^q$: *numpy.ndarray[float[[q,1]]*

define the hyperplanes. The inequality is interpreted element-wise and q is the number of hyperplanes.

V-polytope

class pypolycontain.objects.**V_polytope**(*list_of_vertices*)

V-polytopes are a convex hull of vertices.

$$\mathbb{V} = \{x \in \mathbb{R}^n | x = \sum_{i=1}^{N} \lambda_i v_i, \sum_{i=1}^{N} \lambda_i = 1, \lambda_i \geq 0, i = 1, \cdots, N\}$$

where each $v_i, i = 1, \cdots, N$ is a point (some or all are effectively vertices).

Attributes:

- list_of_vertices= *list* of *numpy.ndarray[float[n,1]]*.

AH-polytope

class pypolycontain.objects.**AH_polytope**(*t*, *T*, *P*, *color='blue'*)

An AH_polytope is an affine transformation of an H-polytope and is defined as:

$$\mathbb{Q} = \{t + Tx | x \in \mathbb{R}^p, Hx \leq h\}$$

Attributes:

- P: The underlying H-polytope $P : \{x \in \mathbb{R}^p | Hx \leq h\}$
- T: $\mathbb{R}^{n \times p}$ matrix: linear transformation
- t: $\mathbb{R}^n$ vector: translation

Zonotope

class pypolycontain.objects.**zonotope**(*G*, *x=None*, *name=None*, *color='green'*)

A Zonotope is a set defined as follows:

$$\mathbb{Z} = \langle x, G \rangle = \{x + Gp | p \in [-1, 1]^q\},$$

where

Attributes:

- $G \in \mathbb{R}^{n \times q}$: *numpy.ndarray[float[[n,q]]* is the zonotope generator.
- $x \in \mathbb{R}^n$: *numpy.ndarray[float[[n,1]]* is the zonotope centroid. Default is zero vector.

The order of the zonotope is defined as $\frac{q}{n}$.

pypolycontain.objects.zonotope.**volume**(*self*)

Computes the volume of the zonotope in dimensions. The formula is based on the paper in [Gover2002]

Unitbox

class pypolycontain.objects.**unitbox**(*N*)

A unitbox in $\mathbb{R}^n$ is $[-1, 1]^n$.

2.1.3 Operations

pypolycontain.operations.**AH_polytope_vertices**(*P*, *N=200*, *epsilon=0.001*, *solver='Gurobi'*)

Returns N*2 matrix of vertices

pypolycontain.operations.**AddMatrixInequalityConstraint_classical**(*mathematical_program*, *A*, *X*, *B*)

pypolycontain.operations.**Hausdorff_distance**(*Q1*, *Q2*, *directed=False*, *ball='infinty_norm'*, *solver='gurobi'*, *k=-1*)

pypolycontain.operations.**Lambda_H_Gamma**(*mathematical_program*, *Lambda*, *H_1*, *H_2*, *Gamma*)

Lambda H_1 = H_2 Gamma

pypolycontain.operations.**Lambda_h_Inequality**(*mathematical_program*, *Lambda*, *beta*, *H*, *h_1*, *h_2*)

Adds Lambda H-1 le h_2 + H beta to the Mathematical Program

pypolycontain.operations.**Lambda_h_Inequality_D**(*mathematical_program*, *Lambda*, *beta*, *H*, *h_1*, *h_2*, *D*)

Adds Lambda H-1 le h_2 D + H beta to the Mathematical Program

pypolycontain.operations.**affine_map**(*T*, *P*, *t=None*, *get_inverse=True*)

Returns the affine map of a polytope.

pypolycontain.operations.**bounding_box**(*Q*, *solver='Gurobi'*)

Computes the bounding box of a polytope by solving $2n$ linear programs. Each linear program is in the form:

$$l_i = \begin{array}{ll} \min & e_i^T x \\ \text{subject to} & x \in \mathbb{P} \end{array}$$

and

$$u_i = \begin{array}{ll} \max & e_i^T x \\ \text{subject to} & x \in \mathbb{P} \end{array}$$

where l, u define the lower and upper corners of the bounding box.

pypolycontain.operations.**boxing_order_reduction**(*zonotope*, *desired_order=1*)

boxing method for zonotope order reduction inputs: input zonotope , order of the output zonotope output: zonotope

Based on Kopetzki, Anna-Kathrin, Bastian Schurmann, and Matthias Althoff. "Methods for order reduction of zonotopes." 2017 IEEE 56th Annual Conference on Decision and Control (CDC). IEEE, 2017.

pypolycontain.operations.**check_non_empty**(*Q*, *tol=1e-05*, *solver='gurobi'*)

pypolycontain.operations.**check_subset**(*P1*, *P2*, *k=-1*)

Checks if .math..'P1 subseteq P2' Inputs:

pypolycontain.operations.**convex_hull**(*P1*, *P2*)

Inputs: P1, P2: AH_polytopes

Output: returns :math:' ext{ConvexHull}(mathbb{P}_1,mathbb{P}_2)' as an AH-polytope

pypolycontain.operations.**convex_hull_of_point_and_polytope**(*x*, *Q*)

Inputs: x: numpy n*1 array Q: AH-polytope in R^n

Returns: AH-polytope representing convexhull(x,Q)

$$\text{conv}(x, Q) := \{y | y = \lambda q + (1 - \lambda)x, q \in Q\}.$$

pypolycontain.operations.**decompose**(*zonotope*, *dimensions*)
@author: kasra Decompising a given set into bunch of fewer dimensional sets such that the Cartesian product of those sets is a subset of the given set.

pypolycontain.operations.**directed_Hausdorff_hyperbox**(*b1*, *b2*)
The directed Hausdorff hyperbox min epsilon such that b1 in b2+epsilon

pypolycontain.operations.**distance_hyperbox**(*b1*, *b2*)
The distance between boxes

pypolycontain.operations.**distance_point_polytope**(*P*, *x*, *ball='infinity'*, *solver='Gurobi'*)
Computes the distance of point x from AH-polytope Q

pypolycontain.operations.**distance_polytopes**(*Q1*, *Q2*, *ball='infinity'*, *solver='gurobi'*)
Finds the closest two points in two polytopes and their distance. It is zero if polytopes have non-empty intersection

pypolycontain.operations.**get_nonzero_cost_vectors**(*cost*)

pypolycontain.operations.**intersection**(*P1*, *P2*)

Inputs: P1, P2: polytopic objects

Output: returns $\mathbb{P}_1 \cap \mathbb{P}_2$ as an AH-polytope

If both objects are H-polytopes, return H-polytope

pypolycontain.operations.**intersection_old**(*P1*, *P2*)

Inputs: P1, P2: AH_polytopes $\mathbb{P}_1, \mathbb{P}_2$. Converted to AH-polytopes

Output: returns $\mathbb{P}_1 \cap \mathbb{P}_2$ as an AH-polytope

pypolycontain.operations.**make_ball**(*n*, *norm*)

pypolycontain.operations.**minkowski_sum**(*P1*, *P2*)

Inputs: P1, P2: AH_polytopes

Returns: returns the Mkinkowski sum $P_1 \oplus P_2$ as an AH-polytope.

Background: The Minkowski sum of two sets is defined as:

$$A \oplus B = \{a + b \big| a \in A, b \in B\}.$$

pypolycontain.operations.**pca_order_reduction**(*zonotope*, *desired_order=1*)
PCA method for zonotope order reduction inputs: input zonotope , order of the output zonotope output: zonotope

Based on Kopetzki, Anna-Kathrin, Bastian Schürmann, and Matthias Althoff. "Methods for order reduction of zonotopes." 2017 IEEE 56th Annual Conference on Decision and Control (CDC). IEEE, 2017.

pypolycontain.operations.**point_membership**(*Q*, *x*, *tol=1e-05*, *solver='gurobi'*)

pypolycontain.operations.**point_membership_fuzzy**(*Q*, *x*, *tol=1e-05*, *solver='gurobi'*)
Fuzzy membership check. If x contains NaN, the entry is unconstrained @param Q: Polytope in R^n @param x: n*1 numpy array, may contain NaNs @param tol: @param solver: solver to use @return: boolean of whether x is in Q

pypolycontain.operations.**positive_matrix**(*mathematical_program*, *Lambda*)
All elements are non-negative

pypolycontain.operations.**sorting_generator**(*G*, *desired_numberofcolumns*)
The goal is deviding the generator into to parts. One part that is used for zonotope order reduction methods. And the other part which is used to enforce the reduced zonotope to have the desire order.

pypolycontain.operations.**translate**(*t*, *P*)
Shifts the polytope by t vector

2.1.4 Conversions

pypolycontain.conversions.**AH_to_H_old**(*Q*, *Po*, *solver='Gurobi'*)
Converting Q to an H-polytope using an optimization-based method

WARNING: To be deprecated

pypolycontain.conversions.**AH_to_V**(*P*, *N=360*, *epsilon=0.001*, *solver='Gurobi'*)
Returns the V-polytope form of a 2D pp.AH_polytope. The method is based on ray shooting.

Inputs:

- P: AH-polytope
- N `defualt=360`: number of rays
- solver: `default=Gurobi`. The linear-programming optimization solver.

Returns:

- V: matrix

Note: This method only works for 2D AH-polytopes and its for visualization. For generic use, first use H-V on $\mathbb{P}$ and then apply affine transformation. Note that H-V uses elimination-based vertex enumeration method that is not scalable.

---

pypolycontain.conversions.**H_to_V**(*P*)
Returns the vertices of an H_polytope.

Inputs:

- P: H_polytope in $\mathbb{R}^n$

Output:

- V: list of vertices. Each vertex is *numpy.ndarray[float[n,1]]*

Method: The method is based on double description method, and is using pycddlib.

> **Warning:** This method can be very slow or numerically unstable for polytopes in high dimensions and/or large number of hyperplanes

pypolycontain.conversions.**to_AH_polytope**(*P*)
Converts the polytopic object P into an AH-polytope. If applied on a AH-polytope, a deepcopy is returned

pypolycontain.conversions.**to_V**(*P*, *N=500*)
returns the vertices of the polytopic object P in a vertical stack form.

pypolycontain.conversions.**vcube**(*n*)
$2^n \times n$ array of vectors of vertices in unit cube in $\mathbb{R}$

pypolycontain.conversions.**zonotope_to_V**(*Z*)
Finds the vertices of a zonotope

2.1.5 Visualization

Important: We only support 2D visualization.

pypolycontain.visualize.**visualize**(*list_of_objects, fig=None, ax=None, a=0.5, alpha=0.8, tuple_of_projection_dimensions=[0, 1], title='pypolycontain visualization', show_vertices=False, TitleSize=15, FontSize=15, equal_axis=False, grid=True, N_points=1000, figsize=(8, 8)*)

Visualization.

inputs:

- list_of_objects:
- fig:
- tuple_of_projection_dimensions:
-

3 Indices and tables

- genindex
- modindex
- search

4 Main Developer

- Sadra Sadraddini[8]

5 Contributors

- Albert Wu[9]
- Kasra Ghasemi[10]

References

[Gover2002] Gover, Eugene, and Nishan Krikorian. "Determinants and the volumes of parallelotopes and zonotopes." Linear Algebra and its Applications 433, no. 1 (2010): 28-40.

[8] http://www.sadraddini.com/
[9] https://github.com/wualbert/
[10] https://github.com/Kasraghasemi

Python Module Index

p