pycvxset Documentation

Release 1.0.1

Abraham P. Vinod

Mitsubishi Electric Research Laboratories (MERL)

Email: vinod@merl.com, abraham.p.vinod@ieee.org

Oct 11, 2024

CONTENTS

CHAPTER

ONE

WHAT CAN PYCVXSET DO FOR ME?

pycvxset is a Python package for manipulation and visualization of convex sets.

Currently, pycvxset supports the following three representations:

1. **polytopes**,
2. **ellipsoids**, and
3. **constrained zonotopes** (which are equivalent to polytopes).

Some of the operations enabled by pycvxset include:

- construct sets from a variety of representations and transform between these representations,
- perform various operations on these sets including (but not limited to):
  - plot in 2D and 3D,
  - affine and inverse-affine transformation,
  - Minkowski sum,
  - Pontryagin difference,
  - intersection,
  - checking for containment,
  - projection,
  - slicing, and
  - support function evaluation.

  See the Jupyter notebooks in *examples* folder for more details on how pycvxset can be used in set-based control and perform reachability analysis.

CHAPTER

TWO

QUICK START

2.1 Requirements

pycvxset supports Python 3.9+ on Ubuntu, Windows, and MacOS. As described in *setup.py*, pycvxset has the following core dependencies:

1. numpy
2. scipy
3. cvxpy
4. matplotlib
5. pycddlib: pycvxset requires `pycddlib<=2.1.8.post1`. We are working on removing this restriction.
6. gurobipy: This dependency is **optional**. Almost all functionalities of pycvxset are available without Gurobi. However, pycvxset uses Gurobi (through cvxpy) to perform *some* containment and equality checks involving constrained zonotopes. See *License* section for more details.

2.2 Installation

Refer to *.github/workflows* for exact steps to install pycvxset for different OSes. These steps are summarized below:

1. OS-dependent pre-installation steps:
   - **Ubuntu**: Install gmp.

     ```
     $ sudo apt-get install libgmp-dev
     ```

   - **MacOS**: Install pycddlib manually in order to link with gmp.

     ```
     % brew install gmp
     % python3 -m pip install --upgrade pip
     % git clone https://github.com/mcmtroffaes/pycddlib.git
     % cd pycddlib
     % git checkout 2.1.8.post1
     % git submodule update --init
     % ./cddlib-makefile-gmp.sh
     % env "CFLAGS=-I$(brew --prefix)/include -L$(brew --prefix)/lib" python3 -m␣
     ↪pip install .
     % cd ..
     ```

     These steps are adapted from pycddlib/build.yml.

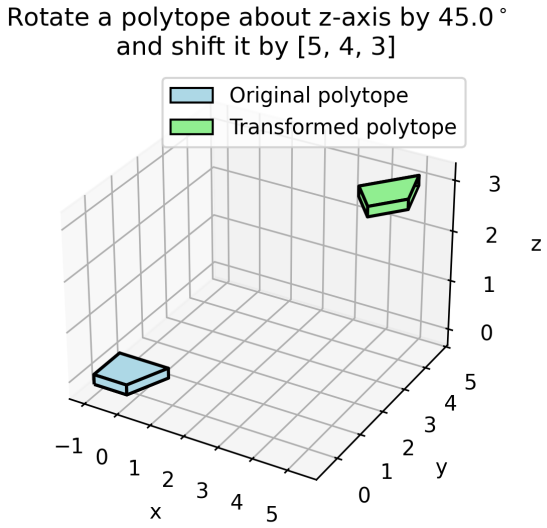
   - **Windows**: No special steps required since pip takes care of it. If plotting fails, you can set matplotlib backend via an environment variable `set MPLBACKEND=Agg`. See https://matplotlib.org/3.5.1/users/explain/backends.html#selecting-a-backend for more details.

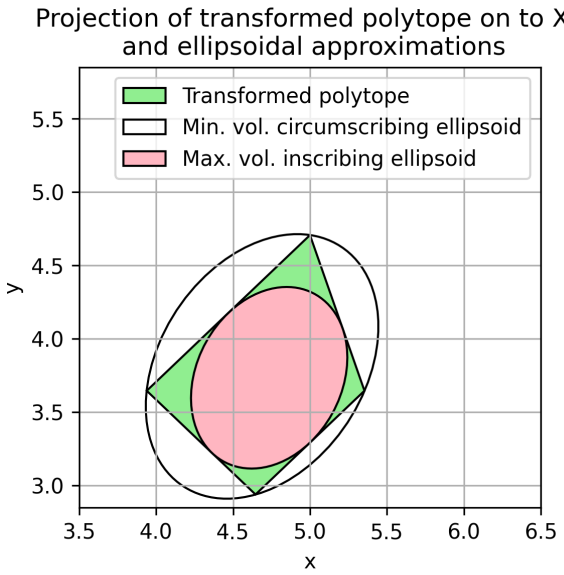
2. Clone the pycvxset repository into a desired folder `PYCVXSET_DIR`.
3. Run `pip install -e .` in the folder `PYCVXSET_DIR`.

2.2.1 Sanity check

Check your installation by running `python3 examples/pycvxset_diag.py` in the folder `PYCVXSET_DIR`. The script should generate a figure with two subplots, each generated using pycvxset.

- *Left subplot*: A 3D plot of two polytopes (one at the origin, and the other translated and rotated). You can interact with this plot using your mouse.
- *Right subplot*: A 2D plot of the projection of the polytope, and its corresponding minimum volume circumscribing and maximum volume inscribing ellipsoids.





2.3 Optional: Testing

1. Use `pip install -e ".[with_tests]"` to install the additional dependencies.
2. Run `$ ./scripts/run_tests_and_update_docs.sh` to view testing results on the command window.
3. Open *./docs/source/_static/codecoverage/overall/index.html* in your browser for coverage results.

2.4 Optional: Documentation

1. Use `pip install -e ".[with_docs_and_tests]"` to install the additional dependencies.
2. To produce the documentation,
   - Run `$./scripts/run_all.sh`. This should take about 15 minutes. For **faster but incomplete options**,
     - To generate just the latex files for the *MANUAL.pdf*, run `$./scripts/run_sphinx_pdf.sh`. This command will assume that the environment has LaTeX setup properly.
     - To build the API documentation without rendering the notebooks or coverage results, run `$./scripts/run_sphinx_html.sh`.
3. View the documentation as,

- PDF at *MANUAL.pdf* ,
- HTML pages in your browser at `./docs/build/index.html`.
  - View the Jupyter notebooks at `./docs/build/tutorials/tutorials.html`.
  - View code coverage from testing at `./docs/build/_static/codecoverage/overall/index.html`.

CHAPTER

THREE

GETTING HELP

Let us say that you have read the *manual* and searched the Discussion and the Issue webpages, but still need help.

Please start a Discussion for

- Getting help in setting up pycvxset to work on your computer.
- Advice on how to use pycvxset for your set manipulations problem.
- Suggestions on documentation of the code.
- Collecting ideas for potential new features/enhancements from the community.

Please open an Issue if you believe that fixing your problem will involve a change in the pycvxset source code. For example

- Bug reports.
- New feature/enhancement proposals with details.

How to submit a good bug report? When opening an Issue, please consider providing:

- High-level description of the behavior you would expect and the actual behavior.
- Which version of pycvxset are you using? The bleeding edge or the version number.
- OS (Windows, Linux, Mac, or something else)
- Can you reliably reproduce the issue?
- Details of the bug (a stack trace can be useful).
- A reduced test case that reproduces the issue for our development team.

CHAPTER

FOUR

CONTRIBUTING

See *CONTRIBUTING.md* for our policy on contributions.

CHAPTER

FIVE

LICENSE

pycvxset is released under `AGPL-3.0-or-later` license, as found in the *LICENSE.md* file.

All files:

```
Copyright (c) 2020-2024 Mitsubishi Electric Research Laboratories (MERL).

SPDX-License-Identifier: AGPL-3.0-or-later
```

except the following files:

1. .gitignore
2. README.md
3. pycvxset/Polytope/__init__.py
4. pycvxset/Polytope/operations_binary.py
5. pycvxset/Polytope/operations_unary.py
6. pycvxset/Polytope/plotting_scripts.py
7. pycvxset/Polytope/vertex_halfspace_enumeration.py
8. pycvxset/__init__.py
9. pycvxset/common/__init__.py
10. setup.py
11. tests/test_polytope_binary.py
12. tests/test_polytope_init.py
13. tests/test_polytope_vertex_facet_enum.py

which have the copyright

```
Copyright (C) 2020-2024 Mitsubishi Electric Research Laboratories (MERL)
Copyright (c) 2019 Tor Aksel N. Heirung

SPDX-License-Identifier: AGPL-3.0-or-later
SPDX-License-Identifier: MIT
```

The method `contains` in *pycvxset/ConstrainedZonotope/operations_binary.py* uses gurobipy (through cvxpy) and requires acceptance of appropriate license terms.

CHAPTER

SIX

ACKNOWLEDGEMENTS

The development of pycvxset started from commit ebd85404 of pytope.

```
Copyright (c) 2019 Tor Aksel N. Heirung

SPDX-License-Identifier: MIT
```

pycvxset extends pytope in several new directions, including:

- Plotting for 3D polytopes using matplotlib,
- Interface with cvxpy for access to a wider range of solvers,
- Introduce new methods for *Polytope* class including circumscribing and inscribing ellipsoids, volume computation, and containment checks,
- Minimize the use of vertex-halfspace enumeration using convex optimization,
- Include extensive documentation and example Jupyter notebooks,
- Implement exhaustive testing with nearly 100% coverage, and
- Support for *Ellipsoid* and *ConstrainedZonotope* set representations.

CHAPTER

SEVEN

CONTACT

For questions or bugs, contact Abraham P. Vinod (Email: vinod@merl.com, abraham.p.vinod@ieee.org).

CHAPTER

EIGHT

PYCVXSET.POLYTOPE

class pycvxset.**Polytope**(***kwargs*)

Bases: object

Polytope class.

Polytope object construction admits **one** of the following combinations (as keyword arguments):

1. (A, b) for a polytope in **halfspace representation (H-rep)** $\{x \mid Ax \leq b\}$,
2. (A, b, Ae, be) for a polytope in **halfspace representation (H-rep) with equality constraints** $\{x \mid Ax \leq b, A_e x = b_e\}$,
3. V for a polytope in **vertex representation (V-rep)** ConvexHull(v_i) where v_i are rows of matrix V,
4. (lb, ub) for an **axis-aligned cuboid** with appropriate bounds $\{x \mid lb \leq x \leq ub\}$, and
5. (c, h) for an **axis-aligned cuboid** centered at c with specified scalar/vector half-sides h, $\{x \mid \forall i \in \{1, \cdots, n\}, |x_i - c_i| \leq h_i\}$.
6. dim for an **empty** Polytope of dimension dim (no argument generates a zero-dimensional **empty** Polytope),

Parameters

- **dim** (*int, optional*) – Dimension of the empty polytope. If NOTHING is provided, dim=0 is assumed.
- **V** (*array_like, optional*) – List of vertices of the polytope (V-Rep). The list must be 2-dimensional with vertices arranged row-wise and the polytope dimension determined by the column count.
- **A** (*array_like, optional*) – Inequality coefficient vectors (H-Rep). The vectors are stacked vertically with the polytope dimension determined by the column count. When A is provided, b must also be provided.
- **b** (*array_like, optional*) – Inequality constants (H-Rep). The constants are expected to be in a 1D numpy array. When b is provided, A must also be provided.
- **Ae** (*array_like*) – Equality coefficient vectors (H-Rep). The vectors are stacked vertically with matching number of columns as A. When Ae is provided, A, b, and be must also be provided.
- **be** (*array_like*) – Equality coefficient constants (H-Rep). The constants are expected to be in a 1D numpy array. When be is provided, A, b, and Ae must also be provided.
- **lb** (*array_like, optional*) – Lower bounds of the axis-aligned cuboid. Must be 1D array, and the polytope dimension is determined by number of elements in lb. When lb is provided, ub must also be provided.
- **ub** (*array_like, optional*) – Upper bounds of the axis-aligned cuboid. Must be 1D array of length as same as lb. When ub is provided, lb must also be provided.

- **c** (*array_like, optional*) – Center of the axis-aligned cuboid. Must be 1D array, and the polytope dimension is determined by number of elements in c. When c is provided, h must also be provided.
- **h** (*array_like, optional*) – Half-side length of the axis-aligned cuboid. Can be a scalar or a vector of length as same as c. When h is provided, c must also be provided.

Raises

- **ValueError** – When arguments provided is not one of [(A, b), (A, b, Ae, be), (lb, ub), (c, h), V, dim, NOTHING]
- **ValueError** – Errors raised by issues with (A, b) and (Ae, be) mismatch in dimensions, not convertible to appropriately-dimensioned numpy arrays, incompatible systems (A, b) and (Ae, be) etc.
- **ValueError** – Errors raised by issues with V not convertible to a 2-D numpy array, etc.
- **ValueError** – Errors raised by issues with lb, ub mismatch in dimensions, not convertible to 1D numpy arrays, etc.
- **ValueError** – Errors raised by issues with c, h mismatch in dimensions, not convertible to 1D numpy arrays, etc.
- **ValueError** – Polytope is not bounded in any direction. We use a sufficient condition for speed, and therefore the detection may not be exhaustive.
- **ValueError** – Polytope is not bounded in some directions. We use a sufficient condition for speed, and therefore the detection may not be exhaustive.
- **UserWarning** – If some rows are removed from (A, b) due to all zeros in A or np.inf in b | (Ae, be) when all zeros in Ae and be.

__init__(***kwargs*)

Constructor for Polytope class.

Methods

__init__(**kwargs)	Constructor for Polytope class.
affine_map(M)	Compute the matrix times set.
chebyshev_centering()	Computes a ball with the largest radius that fits within the polytope.
closest_point(points[, p])	Wrapper for `project()` to compute the point in the convex set closest to the given point.
containment_constraints(x)	Get CVXPY constraints for containment of x (a cvxpy.Variable) in a polytope.
contains(Q)	Check containment of a set $\mathcal{Q}$ (could be a polytope, an ellipsoid, or a constrained zonotope), or a collection of points $Q \in \mathbb{R}^{n_Q \times \mathcal{P}.\text{dim}}$ in the polytope $\mathcal{P}$.
copy()	Create a copy of the polytope
deflate_rectangle(set_to_be_centered)	Compute the rectangle with the smallest volume that contains the given set.
determine_H_rep()	Determine the halfspace representation from a given vertex representation of the polytope.
determine_V_rep()	Determine the vertex representation from a given halfspace representation of the polytope.
distance(points[, p])	Wrapper for `project()` to compute distance of a point to a convex set.
extreme(eta)	Wrapper for `support()` to compute the extreme point.

continues on next page

Table 1 – continued from previous page

interior_point([point_type])	Compute a point in the interior of the polytope.
intersection(Q)	Intersect the polytope $\mathcal{P}$ with another polytope $\mathcal{Q}$.
intersection_under_inverse_affine_map(Q, R)	Compute the intersection of constrained zonotope under an inverse affine map
intersection_with_affine_set(Ae, be)	Intersect the polytope with an affine set.
intersection_with_halfspaces(A, b)	Intersect the polytope with a collection of halfspaces.
inverse_affine_map_under_invertible_mat	Compute the set times matrix, the inverse affine map of the set under an invertible matrix M.
maximum_volume_inscribing_ellipsoid()	Compute the maximum volume ellipsoid that fits within the given polytope.
minimize(x, objective_to_minimize, cvxpy_args)	Solve a convex program with CVXPY objective subject to containment constraints.
minimize_H_rep()	Remove any redundant inequalities from the halfspace representation of the polytope using cdd.
minimize_V_rep([prefer_qhull_over_cdd])	Remove any redundant vertices from the vertex representation of the polytope.
minimum_volume_circumscribing_ellipsoid(	Compute the minimum volume ellipsoid that covers the given polytope (also known as Lowner-John Ellipsoid).
minimum_volume_circumscribing_rectangle(	Compute the minimum volume circumscribing rectangle for a given polytope
minus(Q)	Implement - operation (Pontryagin difference when Q is a polytope, translation by -Q when Q is a point)
normalize()	Normalize a H-Rep such that each row of A has unit ℓ_2-norm.
plot([ax, patch_args, vertex_args, ...])	Plot a 2D or 3D polytope.
plot2d([ax, patch_args, vertex_args, ...])	Plot a 2D polytope using matplotlib's add_patch(Polygon()).
plot3d([ax, patch_args, vertex_args, ...])	Plot a 3D polytope using matplotlib's Line3DCollection.
plus(Q)	Add a point or a set to the polytope
project(x[, p])	Project a point or a collection of points on to a set.
projection(project_away_dim)	Orthogonal projection of a set $\mathcal{P}$ after removing some user-specified dimensions.
slice(dims, constants)	Slice a set restricting certain dimensions to constants.
support(eta)	Evaluates the support function and support vector of a set.
volume()	Compute the volume of the polytope using QHull

Attributes

A	Inequality coefficient vectors A for the polytope $\{Ax \leq b, A_e x = b_e\}$.
Ae	Equality coefficient vectors *Ae* for the polytope $\{Ax \leq b, A_e x = b_e\}$.
H	Inequality constraints in halfspace representation *H=[A, b]* for the polytope $\{Ax \leq b, A_e x = b_e\}$.
He	Equality constraints in halfspace representation *He=[Ae, be]* for the polytope $\{Ax \leq b, A_e x = b_e\}$.
V	Vertex representation (V) where the polytope is given by ConvexHull(v_i) with v_i as the rows of $V = [v_1; v_2; \ldots; v_{n_v ertices}]$.
b	Inequality constants b for the polytope $\{Ax \leq b, A_e x = b_e\}$.
be	Equality constants *be* for the polytope $\{Ax \leq b, A_e x = b_e\}$.
cvxpy_args_lp	CVXPY arguments in use when solving a linear program
cvxpy_args_sdp	CVXPY arguments in use when solving a semi-definite program
cvxpy_args_socp	CVXPY arguments in use when solving a second-order cone program
dim	Dimension of the polytope.
in_H_rep	Check if the polytope have a halfspace representation (H-Rep).
in_V_rep	Check if the polytope have a vertex representation (V-Rep).
is_bounded	Check if the polytope is bounded.
is_empty	Check if the polytope is empty.
is_full_dimensional	Check if the affine dimension of the polytope is the same as the polytope dimension
n_equalities	Number of linear equality constraints used to define the polytope $\{Ax \leq b, A_e x = b_e\}$
n_halfspaces	Number of halfspaces used to define the polytope $\{Ax \leq b, A_e x = b_e\}$
n_vertices	Number of vertices

CHAPTER

NINE

PYCVXSET.POLYTOPE (API DETAILS)

pycvxset.Polytope(**kwargs)	Polytope class.

class pycvxset.Polytope.**Polytope**(***kwargs*)

Bases: object

Polytope class.

Polytope object construction admits **one** of the following combinations (as keyword arguments):

1. (A, b) for a polytope in **halfspace representation (H-rep)** $\{x \mid Ax \leq b\}$,
2. (A, b, Ae, be) for a polytope in **halfspace representation (H-rep) with equality constraints** $\{x \mid Ax \leq b, A_e x = b_e\}$,
3. V for a polytope in **vertex representation (V-rep)** ConvexHull(v_i) where v_i are rows of matrix V,
4. (lb, ub) for an **axis-aligned cuboid** with appropriate bounds $\{x \mid lb \leq x \leq ub\}$, and
5. (c, h) for an **axis-aligned cuboid** centered at c with specified scalar/vector half-sides h, $\{x \mid \forall i \in \{1, \cdots, n\}, |x_i - c_i| \leq h_i\}$.
6. dim for an **empty** Polytope of dimension dim (no argument generates a zero-dimensional **empty** Polytope),

Parameters

- **dim** (*int, optional*) – Dimension of the empty polytope. If NOTHING is provided, dim=0 is assumed.
- **V** (*array_like, optional*) – List of vertices of the polytope (V-Rep). The list must be 2-dimensional with vertices arranged row-wise and the polytope dimension determined by the column count.
- **A** (*array_like, optional*) – Inequality coefficient vectors (H-Rep). The vectors are stacked vertically with the polytope dimension determined by the column count. When A is provided, b must also be provided.
- **b** (*array_like, optional*) – Inequality constants (H-Rep). The constants are expected to be in a 1D numpy array. When b is provided, A must also be provided.
- **Ae** (*array_like*) – Equality coefficient vectors (H-Rep). The vectors are stacked vertically with matching number of columns as A. When Ae is provided, A, b, and be must also be provided.
- **be** (*array_like*) – Equality coefficient constants (H-Rep). The constants are expected to be in a 1D numpy array. When be is provided, A, b, and Ae must also be provided.
- **lb** (*array_like, optional*) – Lower bounds of the axis-aligned cuboid. Must be 1D array, and the polytope dimension is determined by number of elements in lb. When lb is provided, ub must also be provided.

- **ub** (*array_like, optional*) – Upper bounds of the axis-aligned cuboid. Must be 1D array of length as same as lb. When ub is provided, lb must also be provided.
- **c** (*array_like, optional*) – Center of the axis-aligned cuboid. Must be 1D array, and the polytope dimension is determined by number of elements in c. When c is provided, h must also be provided.
- **h** (*array_like, optional*) – Half-side length of the axis-aligned cuboid. Can be a scalar or a vector of length as same as c. When h is provided, c must also be provided.

Raises

- **ValueError** – When arguments provided is not one of [(A, b), (A, b, Ae, be), (lb, ub), (c, h), V, dim, NOTHING]
- **ValueError** – Errors raised by issues with (A, b) and (Ae, be) mismatch in dimensions, not convertible to appropriately-dimensioned numpy arrays, incompatible systems (A, b) and (Ae, be) etc.
- **ValueError** – Errors raised by issues with V not convertible to a 2-D numpy array, etc.
- **ValueError** – Errors raised by issues with lb, ub mismatch in dimensions, not convertible to 1D numpy arrays, etc.
- **ValueError** – Errors raised by issues with c, h mismatch in dimensions, not convertible to 1D numpy arrays, etc.
- **ValueError** – Polytope is not bounded in any direction. We use a sufficient condition for speed, and therefore the detection may not be exhaustive.
- **ValueError** – Polytope is not bounded in some directions. We use a sufficient condition for speed, and therefore the detection may not be exhaustive.
- **UserWarning** – If some rows are removed from (A, b) due to all zeros in A or np.inf in b | (Ae, be) when all zeros in Ae and be.

__init__(***kwargs*)

Constructor for Polytope class.

affine_map(*M*)

Compute the matrix times set.

Parameters

M (*array_like*) – A vector or array (0, 1, or 2-D numpy.ndarray-like object) with self.dim columns

Raises

- **ValueError** – When M can not be converted EXACTLY into a 2D array of float
- **ValueError** – When M does not have self.dim columns

Returns

The scaled polytope $\mathcal{R} = M\mathcal{P} = \{Mx : x \in \mathcal{P}\}$

Return type

Polytope

Notes

This function requires $\mathcal{P}$ to be in V-Rep, and performs a vertex enumeration when $\mathcal{P}$ is in H-Rep.

chebyshev_centering()

Computes a ball with the largest radius that fits within the polytope. The balls center is known as the Chebyshev center, and its radius is the Chebyshev radius.

Raises
NotImplementedError – Unable to solve chebyshev centering problem using CVXPY

Returns

A tuple with two items

1. center (numpy.ndarray): Chebyshev center of the polytope
2. radius (float): Chebyshev radius of the polytope

Return type
tuple

Notes

This function is called by the constructor for non-emptiness and boundedness check. It uses limited attributes (dim, A, b, Ae, be). This function requires H-Rep, and will perform a halfspace enumeration when a V-Rep polytope is provided.

We solve the LP (see Section 8.5.1 in [BV04]) for c (for *center*) and R (for *radius*),

$$\begin{aligned} \text{maximize} \quad & R \\ \text{subject to} \quad & Ac + R||A||_{\text{row}} \leq b, \\ & A_e c = b_e, \\ & R \geq 0, \end{aligned}$$

where $||A||_{\text{row}}$ is a vector of dimension *n_halfspaces* with each element as $||a_i||_2$. When (Ae, be) is non-empty, then R is forced to zero post-solve. Consequently, chebyshev_centering also serves a method to find a feasible point in the relative interior of the polytope.

We can also infer the following from the Chebyshev radius R:

1. $R = \infty$, the polytope is unbounded. Note that, this is just a sufficient condition, and an unbounded polytope can have a finite Chebyshev radius. For example, consider a 3-dimensional axis-aligned cuboid $[-1, 1] \times [-1, 1] \times \mathbb{R}$.
2. $0 < R < \infty$, the polytope is nonempty and full-dimensional.
3. $R = 0$, the polytope is nonempty and but not full-dimensional.
4. $R = -\infty$, the polytope is empty.

closest_point(*points*, *p=2*)

Wrapper for *project()* to compute the point in the convex set closest to the given point.

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.
- **p** (*str | int*) – Norm-type. It can be 1, 2, or inf. Defaults to 2.

Returns
Projection of points to the set as a 2D numpy.ndarray. These arrays have as many rows as points.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *project()* function.

containment_constraints(x)

Get CVXPY constraints for containment of x (a cvxpy.Variable) in a polytope.

Parameters
x (*cvxpy.Variable*) – CVXPY variable to be optimized

Raises
ValueError – When polytope is empty

Returns

A tuple with two items:

1. constraint_list (list): CVXPY constraints for the containment of x in the polytope.
2. theta (cvxpy.Variable | None): CVXPY variable representing the convex combination coefficient when polytope is in V-Rep. It is None when the polytope is in H-Rep or empty.

Return type
tuple

contains(Q)

Check containment of a set $\mathcal{Q}$ (could be a polytope, an ellipsoid, or a constrained zonotope), or a collection of points $Q \in \mathbb{R}^{n_Q \times \mathcal{P}.\text{dim}}$ in the polytope $\mathcal{P}$.

Parameters
Q (*array_like | Polytope | ConstrainedZonotope | Ellipsoid*) – Set or a collection of points (each row is a point) to be tested for containment within $\mathcal{P}$. When providing a collection of points, Q is a matrix (N times self.dim) with each row is a point.

Raises
ValueError – When two polytopes | the polytope and the test point(s) are NOT of the same dimension

Returns
When Q is a polytope, a bool is returned which is True if and only if $Q \subseteq P$. On the other hand, if Q is array_like (Q is a point or a collection of points), then an numpy.ndarray of bool is returned, with as many elements as the number of rows in Q.

Return type
bool | numpy.ndarray[bool]

Notes

- *Containment of polytopes*: This function accommodates the following combinations of representations for $\mathcal{P}$ and $\mathcal{Q}$. * P is H-Rep and Q is H-Rep: No enumeration needed. Compare support function evaluations. * P is H-Rep and Q is V-Rep: No enumeration needed. Check for containment of vertices of Q in P. * P is V-Rep and Q is H-Rep: Vertex enumeration needed to compute Q.V * P is V-Rep and Q is V-Rep: No enumeration needed. Check for containment of vertices of Q in P. * Q is empty: Return True * P is empty: Return False
- *Containment of points*: This function accommodates $\mathcal{P}$ to be in H-Rep or V-Rep. When testing if a point is in a polytope where only V-Rep is available for self, we solve a collection of second-order cone programs for each point using *project()* (check if distance between v and $\mathcal{P}$ is nearly zero). Otherwise, we use the observation that for $\mathcal{P}$ with (A, b), then $v \in \mathcal{P}$ if and only if $Av \leq b$. For numerical precision considerations, we use `numpy.isclose()`.

copy()

Create a copy of the polytope

classmethod deflate_rectangle(*set_to_be_centered*)

Compute the rectangle with the smallest volume that contains the given set.

Parameters
set_to_be_centered (Polytope | ConstrainedZonotope | Ellipsoid) – Set to compute the.

Returns
Minimum volume circumscribing rectangle

Return type
Polytope

Notes

This function is a wrapper for *minimum_volume_circumscribing_rectangle()* of attr:*set_to_be_centered*. Please check that function for more details including raising exceptions.

determine_H_rep()

Determine the halfspace representation from a given vertex representation of the polytope.

Raises
ValueError – When H-rep computation fails

Notes

- When the set is empty, we define an empty polytope.
- Otherwise, we use cdd for the halfspace enumeration.
- The computed vertex representation need not be minimal.

determine_V_rep()

Determine the vertex representation from a given halfspace representation of the polytope.

Raises
ValueError – Vertex enumeration yields rays, which indicates that the polytope is unbounded. Numerical issues may also be a culprit.

Notes

We use cdd for the vertex enumeration. cdd uses the halfspace representation $[b, -A]$ where $b - Ax \geq 0 \Leftrightarrow Ax \leq b$.

For a polyhedron described as

$$\mathcal{P} = \text{conv}(v_1, ..., v_n) + \text{nonneg}(r_1, ..., r_s),$$

the V-representation matrix in cdd is [t V] where t is the column vector with n ones followed by s zeroes, and V is the stacked matrix of n vertex row vectors on top of s ray row vectors. *pycvxset* uses only bounded polyhedron, so we should never observe rays.

distance(*points*, *p=2*)

Wrapper for *project()* to compute distance of a point to a convex set.

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.

- **p** (*int | str*) – Norm-type. It can be 1, 2, or inf. Defaults to 2.

Returns
Distance of points to the set as a 1D numpy.ndarray. These arrays have as many rows as points.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *project()* function.

extreme(*eta*)

Wrapper for *support()* to compute the extreme point.

Parameters
eta (*array_like*) – Support directions. Matrix (N times self.dim), where each row is a support direction.

Returns
Support vector evaluation(s) as a 2D numpy.ndarray. The array has as many rows as eta.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *support()* function.

interior_point(*point_type='centroid'*)

Compute a point in the interior of the polytope. When the polytope is not full-dimensional, the point may lie on the boundary.

Parameters
point_type (*str*) – Type of interior point. Valid strings: {centroid, chebyshev}. Defaults to centroid.

Raises

- **NotImplementedError** – When an invalid point_type is provided.
- **ValueError** – When the polytope provided is empty.
- **ValueError** – When the polytope provided is not bounded.

Returns
A feasible point in the polytope in the interior as a 1D array

Return type
numpy.ndarray

Notes

- point_type is centroid: Computes the average of the vertices. The function requires the polytope to be in V-Rep, and a vertex enumeration is performed if the polytope is in H-Rep.
- point_type is chebyshev: Computes the Chebyshev center. The function requires the polytope to be in H-Rep, and a halfspace enumeration is performed if the polytope is in V-Rep.

`intersection`(Q)

Intersect the polytope $\mathcal{P}$ with another polytope $\mathcal{Q}$.

Parameters
`Q` (`Polytope`) – Polytope to be intersected with self

Raises
`ValueError` – Q is not a polytope | Mismatch in dimensions

Returns
The intersection of P and Q

Return type
Polytope

Notes

This function requires $\mathcal{P}$ and Q to be of the same dimension and in H-Rep, and performs halfspace enumerations when P or Q are in V-Rep.

`intersection_under_inverse_affine_map`(Q, R)

Compute the intersection of constrained zonotope under an inverse affine map

Parameters

- `Q` (`Polytope`) – Set to intersect with
- `R` (*array_like*) – Matrix of dimension Y.dim times self.dim

Raises

- `ValueError` – When Q is not a Polytope
- `ValueError` – When R is not of correct dimension
- `ValueError` – When self is not bounded

Returns
The intersection of a polytope with another polytope under an inverse affine map. Specifically, given polytopes $\mathcal{P}$ (self) and $\mathcal{Q}$, and a matrix R, we compute the set $\{x \in \mathcal{P} | Rx \in \mathcal{Q}\}$.

Return type
Polytope

Notes

This function requires both polytopes to be in H-Rep. Halfspace enumeration is performed when necessary.

Unlike *inverse_affine_map_under_invertible_matrix()*, this function does not require R to be invertible or square and also accommodates Q to be an unbounded polytope. However, self must be a bounded set.

`intersection_with_affine_set`(*Ae*, *be*)

Intersect the polytope with an affine set.

Parameters

- **Ae** (*array_like*) – Equality coefficient vectors (H-Rep). The vectors are stacked vertically.
- **be** (*array_like*) – Equality constants (H-Rep). The constants are expected to be in a 1D numpy array.

Raises

ValueError – Mismatch in dimensions | (Ae, be) is not a valid collection of equality constraints.

Returns

The intersection of P and $\{x : A_e x = b_e\}$

Return type

Polytope

Notes

This function requires $\mathcal{P}$.dim = *Ae.shape[1]* and $\mathcal{P}$ should be in H-Rep, and performs halfspace enumeration if $\mathcal{P}$ in V-Rep.

`intersection_with_halfspaces`(*A*, *b*)

Intersect the polytope with a collection of halfspaces.

Parameters

- **A** (*array_like*) – Inequality coefficient vectors (H-Rep). The vectors are stacked vertically.
- **b** (*array_like*) – Inequality constants (H-Rep). The constants are expected to be in a 1D numpy array.

Raises

ValueError – Mismatch in dimensions | (A, b) is not a valid collection of halfspaces

Returns

The intersection of P and $\{x \in \mathbb{R}^n \mid Ax \leq b\}$

Return type

Polytope

Notes

This function requires $\mathcal{P}$.dim = *A.shape[1]* and $\mathcal{P}$ should be in H-Rep, and performs halfspace enumeration if $\mathcal{P}$ in V-Rep.

`inverse_affine_map_under_invertible_matrix`(*M*)

Compute the set times matrix, the inverse affine map of the set under an invertible matrix M.

Parameters

M (*array_like*) – A invertible array of size self.dim times self.dim

Raises

- **ValueError** – When self is empty
- **TypeError** – When M is not convertible into a 2D numpy array of float
- **ValueError** – When M is not a square matrix
- **ValueError** – When M is not invertible

Returns
The inverse-scaled polytope $\mathcal{R} = \mathcal{P} \times M = \{x : Mx \in \mathcal{P}\}$

Return type
Polytope

Notes

- This function accommodates $\mathcal{P}$ to be in H-Rep or in V-Rep. When $\mathcal{P}$ is in H-Rep, $\mathcal{P}M = \{x|Mx \in \mathcal{P}\} = \{x|AMx \leq b, A_eMx = b_e\}$. On the other hand, when $\mathcal{P}$ is in V-Rep, $\mathcal{P}M = ConvexHull(M^{-1}v_i)$.
- We require M to be invertible in order to ensure that the resulting set is representable as a polytope.

`maximum_volume_inscribing_ellipsoid()`

Compute the maximum volume ellipsoid that fits within the given polytope.

Raises

- **`ValueError`** – When polytope is empty or has non-empty (Ae, be)
- **`NotImplementedError`** – Unable to solve convex problem using CVXPY

Returns

A tuple with three items:

1. center (numpy.ndarray): Maximum volume inscribed ellipsoids center
2. shape_matrix (numpy.ndarray): Maximum volume inscribed ellipsoids shape matrix
3. sqrt_shape_matrix (numpy.ndarray): Maximum volume inscribed ellipsoids square root of shape matrix. Returns None if the polytope is not full-dimensional.

Return type
tuple

Notes

This function requires H-Rep, and will perform a vertex enumeration when a V-Rep polytope is provided.

We solve the SDP for a positive definite matrix $B \in \mathbb{S}^n_{++}$ and $d \in \mathbb{R}^n$,

$$\begin{aligned} \text{maximize} \quad & \log\det B \\ \text{subject to} \quad & \|Ba_i\|_2 + a_i^T d \leq b_i, \\ & A_e d = b_e, \\ & BA_e^\top = 0, \end{aligned}$$

where (a_i, b_i) is the set of hyperplanes characterizing $\mathcal{P}$, and the inscribing ellipsoid is given by $\{Bu + d | \|u\|_2 \leq 1\}$. The center of the ellipsoid is given by $c = d$, and the shape matrix is given by $Q = (BB)^{-1}$. See [EllipsoidalTbx-Min_verticesolEll] and Section 8.4.2 in [BV04] for more details. The last two constraints arise from requiring the inscribing ellipsoid to lie in the affine set $\{A_e x = b_e\}$.

When the polytope is full-dimensional, we can instead solve a second-order cone program (SOCP). Consider the full-dimensional ellipsoid $\{Lu + c | \|u\|_2 \leq 1\}$, where G is a square, lower-triangular matrix with positive diagonal entries. Then, we solve the following (equivalent) optimization problem:

$$\begin{aligned} \text{minimize} \quad & \text{geomean}(L) \\ \text{subject to} \quad & \text{diag}(L) \geq 0 \\ & \|L^T a_i\|_2 + a_i^T d \leq b_i, \end{aligned}$$

with decision variables G and c. Here, we use the observation that geomean(L) is a monotone function of $\log\det(GG^T)$ (the volume of the ellipsoid).

minimize(*x*, *objective_to_minimize*, *cvxpy_args*, *task_str=''*)

Solve a convex program with CVXPY objective subject to containment constraints.

Parameters

- **x** (*cvxpy.Variable*) – CVXPY variable to be optimized
- **objective_to_minimize** (*cvxpy.Expression*) – CVXPY expression to be minimized
- **cvxpy_args** (*dict*) – CVXPY arguments to be passed to the solver
- **task_str** (*str, optional*) – Task string to be used in error messages. Defaults to .

Raises

NotImplementedError – Unable to solve problem using CVXPY

Returns

A tuple with three items:

1. x.value (numpy.ndarray): Optimal value of x. np.nan * np.ones((self.dim,)) if the problem is not solved.
2. problem.value (float): Optimal value of the convex program. np.inf if the problem is infeasible, -np.inf if problem is unbounded, and finite otherwise.
3. problem_status (str): Status of the problem

Return type

tuple

Notes

This function uses *containment_constraints()* to obtain the list of CVXPY expressions that form the containment constraints on x.

Warning

Please pay attention to the NotImplementedError generated by this function. It may be possible to get CVXPY to solve the same problem by switching the solver. For example, consider the following code block.

```
from pycvxset import Polytope
P = Polytope(A=[[1, 1], [-1, -1]], b=[1, 1])
P.cvxpy_args_lp = {'solver': 'CLARABEL'}    # Default solver used in␣
↪pycvxset
try:
    print('Is polytope bounded?', P.is_bounded)
except NotImplementedError as err:
    print(str(err))
P.cvxpy_args_lp = {'solver': 'OSQP'}
print('Is polytope bounded?', P.is_bounded)
```

This code block produces the following output:

```
Unable to solve the task (support function evaluation of the set at eta =␣
↪[-0. -1.]). CVXPY returned error:
Solver 'CLARABEL' failed. Try another solver, or solve with verbose=True␣
↪for more information.

Is polytope bounded? False
```

minimize_H_rep()

Remove any redundant inequalities from the halfspace representation of the polytope using cdd.

Raises
ValueError – When minimal H-Rep computation fails!

minimize_V_rep(*prefer_qhull_over_cdd=True*)

Remove any redundant vertices from the vertex representation of the polytope.

Parameters
prefer_qhull_over_cdd (*bool, optional*) – When True, minimize_V_rep uses qhull. Otherwise, we use cdd.

Raises
ValueError – When minimal V-Rep computation fails!

Notes

Use cdd or qhull for the reduction of vertices. Requires the polytope to be bounded.

minimum_volume_circumscribing_ellipsoid()

Compute the minimum volume ellipsoid that covers the given polytope (also known as Lowner-John Ellipsoid).

Raises

- **ValueError** – When polytope is empty
- **NotImplementedError** – Unable to solve convex problem using CVXPY

Returns

A tuple with three items:

1. center (numpy.ndarray): Minimum volume circumscribed ellipsoids center
2. shape_matrix (numpy.ndarray): Minimum volume circumscribed ellipsoids shape matrix
3. sqrt_shape_matrix (numpy.ndarray): Minimum volume circumscribed ellipsoids square root of shape matrix. Returns None if the polytope is not full-dimensional.

Return type
tuple

Notes

This function requires V-Rep, and will perform a vertex enumeration when a H-Rep polytope is provided.

We solve the SDP for a positive definite matrix $A \in \mathbb{S}^n_{++}$ and $b \in \mathbb{R}^n$,

$$\begin{aligned} \text{maximize} \quad & \log\det A^{-1} \\ \text{subject to} \quad & \|Av_i + b\|_2 \leq 1, \ \forall \text{ vertices } v_i \text{ of } \mathcal{P} \end{aligned}$$

where the circumscribing ellipsoid is given by $\{x | \|Ax+b\|_2 \leq 1\}$, and we use the observation that $\log\det A^{-1} = -\log\det A$. The center of the ellipsoid is given by $c = -A^{-1}b$, and the shape matrix is given by $Q = (A^TA)^{-1}$. See [EllipsoidalTbx-Min_verticesolEll] and Section 8.4.1 in [BV04] for more details.

When the polytope is full-dimensional, we can instead solve a second-order cone program (SOCP). Consider the full-dimensional ellipsoid $\{Lu + c | \|u\|_2 \leq 1\}$, where G is a square, lower-triangular matrix with positive diagonal entries. Then, we solve the following (equivalent) optimization problem:

$$\begin{aligned} \text{minimize} \quad & \text{geomean}(L) \\ \text{subject to} \quad & \text{diag}(L) > 0 \\ & v_i = Lu_i + c, \ \forall \text{ vertices } v_i \text{ of } \mathcal{P}, \\ & u_i \in \mathbb{R}, \ \|u_i\|_2 \leq 1, \end{aligned}$$

with decision variables G, c, and u_i, and geomean is the geometric mean of the diagonal elements of G. Here, we use the observation that geomean(L) is a monotone function of $\log\det(GG^T)$ (the volume of the ellipsoid). For the sake of convexity, we solve the following equivalent optimization problem after a change of variables $L_{\text{inv}} = L^{-1}$ and $c_{L_{\text{inv}}} = L^{-1}c$, and substituting for the variables u_i:

$$\begin{aligned} \text{maximize} \quad & \text{geomean}(L_{\text{inv}}) \\ \text{subject to} \quad & \text{diag}(L_{\text{inv}}) > 0 \\ & \|L_{\text{inv}} v_i - c_{L_{\text{inv}}}\|_2 \leq 1, \ \forall \text{ vertices } v_i \end{aligned}$$

The center of the ellipsoid is $c = Lc_{L_{\text{inv}}}$, and the shape matrix is $Q = GG^T$, where $L = L_{\text{inv}}^{-1}$.

minimum_volume_circumscribing_rectangle()

Compute the minimum volume circumscribing rectangle for a given polytope

Raises
ValueError – When polytope is empty

Returns

A tuple of two items

1. lb (numpy.ndarray): Lower bound l on the polytope, $\mathcal{P} \subseteq \{l\} \oplus \mathbb{R}_{\geq 0}$.
2. ub (numpy.ndarray): Upper bound u on the polytope, $\mathcal{P} \subseteq \{u\} \oplus (-\mathbb{R}_{\geq 0})$.

Return type
tuple

Notes

This function accommodates H-Rep and V-Rep. The lower/upper bound for V-Rep is given by an element-wise minimization/maximization operation, while the lower/upper bound for H-Rep is given by an element-wise support computation (2 * self.dim linear programs). For a H-Rep polytope, the function uses *support()* and is a wrapper for `pycvxset.common.minimum_volume_circumscribing_rectangle()`.

This function is called to check for boundedness of a polytope.

minus(Q)

Implement - operation (Pontryagin difference when Q is a polytope, translation by -Q when Q is a point)

Parameters

- **Q** (*array_like | Polytope | ConstrainedZonotope | Ellipsoid*) – Polytope to be subtracted in Pontryagin
- **polytope** (*difference sense from self or a vector for negative translation of the*)

Raises

- **TypeError** – When Q is neither a Polytope or a point
- **ValueError** – When Q is not of the same dimension as self

Returns
The polytope $\mathcal{R}$ that is the Pontryagin difference of P and Q or a negative translation of P by Q.

Return type
Polytope

Notes

- *Subtraction with a point*: This function accommodates $\mathcal{P}$ in H-Rep and V-Rep. See `plus()` for more details.
- *Subtraction with a polytope*: This function requires $\mathcal{P}$ and Q to be of the same dimension and :math:mathcal{P} in H-Rep, and performs halfspace enumerations when $\mathcal{P}$ is in V-Rep. The H-rep of the polytope $\mathcal{R}$ is, $H_{\mathcal{R}} = [\mathcal{P}.A, \mathcal{P}.b - \rho_{\mathcal{Q}}(\mathcal{P}.A)]$ where $\rho_{\mathcal{Q}}$ is the support function (see `support()`). [KG98]

normalize()
Normalize a H-Rep such that each row of A has unit ℓ_2-norm.

Notes

This function requires P to be in H-Rep. If P is in V-Rep, a halfspace enumeration is performed.

plot(*ax=None*, *patch_args=None*, *vertex_args=None*, *autoscale_enable=True*, *decimal_precision=3*)
Plot a 2D or 3D polytope.

Parameters

- **ax** (*axis object, optional*) – Axis on which the patch is to be plotted
- **patch_args** (*dict, optional*) – Arguments to pass for plotting faces and edges. See [Matplotlib-patch] for options for patch_args for 2D and [Matplotlib-Line3DCollection] for options for patch_args for 3D. Defaults to None, in which case we set edgecolor to black, and facecolor to skyblue.
- **vertex_args** (*dict, optional*) – Arguments to pass for plotting vertices. See [Matplotlib-scatter] for options for vertex_args for 2D and [Matplotlib-Axes3D.scatter] for options for vertex_args for 3D. Defaults to None, in which case we skip plotting the vertices.
- **autoscale_enable** (*bool, optional*) – When set to True, matplotlib adjusts axes to view full polytope. Defaults to True.
- **decimal_precision** (*int, optional*) – When plotting a 3D polytope that is in V-Rep and not in H-Rep, we round vertex to the specified precision to avoid numerical issues. Defaults to PLOTTING_DECIMAL_PRECISION_CDD specified in pycvxset.common.constants.

Raises

- **ValueError** – When an incorrect axes is provided.
- **NotImplementedError** – When a polytope.dim >= 4 or == 1 | autoscale_enable is set to False for 3D plotting.
- **UserWarning** – When an empty polytope is provided.
- **UserWarning** – In 3D plotting, when all faces have less than 3 vertices

Returns
Tuple with axes containing the polygon, handle for plotting patch, handle for plotting vertices

Return type
(axes, handle, handle)

Notes

- plot is a wrapper for `plot2d()` and `plot3d()` functions. See their documentation for more details.
- *Vertex-halfspace enumeration for 2D polytopes*: If a 2D polytope is in H-Rep, vertex enumeration is performed to plot the polytope in 2D. No vertex enumeration is performed for a 2D polytope in V-Rep. This function calls minimize_V_rep to simplify plotting.
- *Vertex-halfspace enumeration for 3D polytopes*: The 3D polytope needs to have both V-Rep and H-Rep. Consequently, at least a halfspace/vertex enumeration is performed for a 3D polytope in single representation.
- *Rounding vertices*: This function calls `pycvxset.common.prune_and_round_vertices()` when a 3D polytope in V-Rep is given to be plotted.
- *Handle returned for the patches*: For 2D plot, axis handle of the single patch is returned. For 3D plotting, multiple patches are present, and plot returns the first patchs axis handle. In this case, the order of patches are determined by the use of pycddlib.Polyhedron.get_input_incidence from the specified (A, b). Also, labeling the first patch is done using Poly3DCollection.
- *Disabling face colors*: We can plot the polytope frames without filling it by setting *patch_args[facecolor] = None* or *patch_args[fill] = False*. If visual comparison of 3D polytopes is desired, it may be better to plot just the polytope frames instead by setting patch_args[facecolor] = None.

> **Warning**
>
> This function may produce erroneous-looking plots when visually comparing multiple 3D sets. For more info on matplotlib limitations, see https://matplotlib.org/stable/api/toolkits/mplot3d/faq.html#my-3d-plot-doesn-t-look-right-at-certain-viewing-angles. If visual comparison is desired, it may be better to plot just the polytope frame instead by setting patch_args[facecolor] = None.

plot2d(*ax=None*, *patch_args=None*, *vertex_args=None*, *autoscale_enable=True*)

Plot a 2D polytope using matplotlibs add_patch(Polygon()).

Parameters

- **ax** (*axis object, optional*) – Axis on which the patch is to be plotted
- **patch_args** (*dict, optional*) – Arguments to pass for plotting faces and edges. See [Matplotlib-patch] for options for patch_args. Defaults to None, in which case we set edgecolor to black, and facecolor to skyblue.
- **vertex_args** (*dict, optional*) – Arguments to pass for plotting vertices. See [Matplotlib-scatter] for options for vertex_args. Defaults to None, in which case we skip plotting the vertices.
- **autoscale_enable** (*bool, optional*) – When set to True (default), matplotlib adjusts axes to view full polytope.

Raises
ValueError – When a 2D polytope is provided

Returns
Tuple with axes containing the polygon, handle for plotting patch, handle for plotting vertices

Return type
(axes, handle, handle)

Notes

This function requires polytope in V-Rep, and performs a vertex enumeration if the polytope is H-Rep.

We sort the vertices in counter-clockwise direction with respect to the centroid, and then plot it using matplotlibs Polygon. We can plot just the polytope frames without filling it by setting *patch_args[facecolor] = None* or *patch_args[fill] = False*.

plot3d(*ax=None*, *patch_args=None*, *vertex_args=None*, *autoscale_enable=True*, *decimal_precision=3*)

Plot a 3D polytope using matplotlibs Line3DCollection.

Parameters

- **ax** (*Axes, optional*) – Axes to plot. Defaults to None, in which case a new axes is created. The function assumes that the provided axes was defined with projection=3d.
- **patch_args** (*dict, optional*) – Arguments to pass for plotting faces and edges. See [Matplotlib-Line3DCollection] for options for patch_args. Defaults to None, in which case we set edgecolor to black, and facecolor to skyblue.
- **vertex_args** (*dict, optional*) – Arguments to pass for plotting vertices. See [Matplotlib-Axes3D.scatter] for options for vertex_args. Defaults to None, in which case we skip plotting the vertices.
- **autoscale_enable** (*bool, optional*) – When set to True (default), matplotlib adjusts axes to view full polytope.
- **decimal_precision** (*int, optional*) – When plotting a 3D polytope that is in V-Rep and not in H-Rep, we round vertex to the specified precision to avoid numerical issues. Defaults to PLOTTING_DECIMAL_PRECISION_CDD specified in pycvxset.common.constants.

Raises

- **ValueError** – When either a non-3D polytope is provided.
- **UserWarning** – When all faces have less than 3 vertices

Returns

Tuple with axes containing the polygon, handle for plotting patch, handle for plotting vertices

Return type

(axes, handle, handle)

Notes

- This function requires polytope in H-Rep and V-Rep, and uses CDD to compute incidences, and vertices. Consequently, at least one halfspace/vertex enumeration is performed.
- Since 3D plotting involves multiple patches, the first patchs axis handle is returned. In this case, the order of patches are determined by the use of pycddlib.Polyhedron.get_input_incidence from the specified (A, b).
- This function calls *pycvxset.common.prune_and_round_vertices()* when a 3D polytope in V-Rep is given to be plotted.
- When label is passed in patch_args, the label is only applied to the first patch. Subsequent patches will not have a label. Moreover, the first patch is plotted using Poly3DCollection, while the subsequent patches are plotted using Line3DCollection in this case. Otherwise, when no label is provided, all patches are plotted using Line3DCollection.
- We iterate over each halfspace, rotate the halfspace about its centroid so that halfspace is now parallel to XY plane, and then sort the vertices based on their XY coordinates in counter-clockwise direction with respect to the centroid, and then plot it using matplotlibs Line3DCollection.

> **Warning**
>
> This function may produce erroneous-looking plots when visually comparing multiple 3D sets. For more info on matplotlib limitations, see https://matplotlib.org/stable/api/toolkits/mplot3d/faq.html#my-3d-plot-doesn-t-look-right-at-certain-viewing-angles. If visual comparison is desired, it may be better to plot just the polytope frame instead by setting patch_args[facecolor] = None.

plus(Q)

Add a point or a set to the polytope

Parameters
Q (*array_like | Polytope*) – Point or set to add to the polytope.

Raises
ValueError – When the point dimension does not match the polytope dimension.

Returns
Polytope which is the sum of self and Q.

Return type
Polytope

Notes

- Given a polytope $\mathcal{P}$, and a set Q, this function computes the Minkowski sum of Q and the polytope, defined as $\mathcal{R} = \{x + q | x \in \mathcal{P}, q \in \mathcal{Q}\}$. On the other hand, when Q is a point, this function computes the polytope $\mathcal{R} = \{x + Q | x \in \mathcal{P}\}$.
- *Addition with a point*: This function allows for $\mathcal{P}$ to be in V-Rep or in H-Rep. For $\mathcal{P}$ in V-rep, the polytope $\mathcal{R}$ is the convex hull of the vertices of $\mathcal{P}$ shifted by point. Given $\{v_i\}$ as the collection of vertices of $\mathcal{P}$, $\mathcal{R} = \text{convexHull}(v_i + \text{point})$. For $\mathcal{P}$ in H-rep, the polytope $\mathcal{R}$ is defined by all points $r = \text{point} + x$ with $Ax \leq b$, $A_e x = b_e$. Thus, $\mathcal{R} = \{x : Ax \leq b + A\text{point}, A_e x = b_e + A_e\text{point}\}$.
- *Addition with a polytope*: This function requires self and Q to be in V-Rep, and performs a vertex enumeration when self or Q are in H-Rep.. In vertex representation, $\mathcal{R}$ is the convex hull of the pairwise sum of all combinations of points in $\mathcal{P}$ and $\mathcal{Q}$.

project(x, *p=2*)

Project a point or a collection of points on to a set.

Given a set $\mathcal{P}$ and a test point $y \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program,

$$\begin{aligned} \text{minimize} \quad & \|x - y\|_p \\ \text{subject to} \quad & x \in \mathcal{P} \end{aligned}$$

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.
- **p** (*str | int*) – Norm-type. It can be 1, 2, or inf. Defaults to 2, which is the Euclidean norm.

Raises

- **ValueError** – Set is empty
- **ValueError** – Dimension mismatch no. of columns in points is different from self.dim.
- **ValueError** – Points is not convertible into a 2D array

- **NotImplementedError** – Unable to solve problem using CVXPY

Returns

A tuple with two items:

1. projected_point (numpy.ndarray): Projection point(s) as a 2D numpy.ndarray. Matrix (N times self.dim), where each row is a projection of the point in points to the set $\mathcal{P}$.
2. distance (numpy.ndarray): Distance(s) as a 1D numpy.ndarray. Vector (N,), where each row is a projection of the point in points to the set $\mathcal{P}$.

Return type
tuple

Notes

- This function allows for $\mathcal{P}$ to be in V-Rep or in H-Rep.
- Given a polytope $\mathcal{P}$ in V-Rep and a test point $y \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program with decision variables $x \in \mathbb{R}^{\mathcal{P}.\text{dim}}$ and $\theta \in \mathbb{R}^{\mathcal{P}.\text{n\_vertices}}$,

$$\begin{aligned} \text{minimize} \quad & \|x - y\|_p \\ \text{subject to} \quad & x = \sum_i \theta_i v_i \\ & \sum_i \theta_i = 1, \theta_i \geq 0 \end{aligned}$$

- Given a polytope $\mathcal{P}$ in H-Rep and a test point $y \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program with a decision variable $x \in \mathbb{R}^{\mathcal{P}.\text{dim}}$,

$$\begin{aligned} \text{minimize} \quad & \|x - y\|_p \\ \text{subject to} \quad & Ax \leq b \\ & A_e x = b_e \end{aligned}$$

projection(*project_away_dim*)

Orthogonal projection of a set $\mathcal{P}$ after removing some user-specified dimensions.

$$\mathcal{R} = \{r \in \mathbb{R}^m \mid \exists v \in \mathbb{R}^{n-m},\ \text{Lift}(r, v) \in \mathcal{P}\}$$

Here, $m = \mathcal{P}.\text{dim} - \text{length}(\text{project\_away\_dim})$, and $\text{Lift}(r, v)$ lifts (undos the projection) using the appropriate components of v. This function uses *affine_map()* to implement the projection by designing an appropriate affine map $M \in \{0, 1\}^{m \times \mathcal{P}.\text{dim}}$ with each row of M corresponding to some standard axis vector $e_i \in \mathbb{R}^m$.

Parameters
project_away_dim (*array_like*) – Dimensions to projected away in integer interval [0, 1, , n - 1].

Returns
m-dimensional set obtained via projection.

Return type
Polytope

Notes

This function requires P to be in V-Rep, and performs a vertex enumeration when P is in H-Rep.

slice(*dims*, *constants*)

Slice a set restricting certain dimensions to constants.

Parameters

- **dims** (*array_like*) – List of dims to restrict to a constant in the integer interval [0, 1, , n - 1].
- **constants** (*array_like*) – List of constants

Raises

- **ValueError** – dims has entries beyond n
- **ValueError** – dims and constants are not 1D arrays of same size

Returns

Polytope that has been sliced at the specified dimensions.

Return type

Polytope

Notes

- This function requires $\mathcal{P}$ to be in H-Rep, and performs a vertex halfspace when $\mathcal{P}$ is in V-Rep.
- This function uses *intersection_with_affine_set()* to implement the slicing by designing an appropriate affine set from dims and constants.

support(*eta*)

Evaluates the support function and support vector of a set.

The support function of a set $\mathcal{P}$ is defined as $\rho_{\mathcal{P}}(\eta) = \max_{x \in \mathcal{P}} \eta^\top x$. The support vector of a set $\mathcal{P}$ is defined as $\nu_{\mathcal{P}}(\eta) = \arg\max_{x \in \mathcal{P}} \eta^\top x$.

Parameters

eta (*array_like*) – Support directions. Matrix (N times self.dim), where each row is a support direction.

Raises

- **ValueError** – Set is empty
- **ValueError** – Mismatch in eta dimension
- **ValueError** – eta is not convertible into a 2D array
- **NotImplementedError** – Unable to solve problem using CVXPY

Returns

A tuple with two items:

1. support_function_evaluations (numpy.ndarray): Support function evaluation(s) as a 2D numpy.ndarray. Vector (N,) with as many rows as eta.
2. support_vectors (numpy.ndarray): Support vectors as a 2D numpy.ndarray. Matrix N x self.dim with as many rows as eta.

Return type

tuple

Notes

- This function allows for :math:mathcal{P} to be in V-Rep or in H-Rep.
- Given a polytope $\mathcal{P}$ in V-Rep and a support direction $\eta \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program with decision variables $x \in \mathbb{R}^{\mathcal{P}.\text{dim}}$ and $\theta \in \mathbb{R}^{\mathcal{P}.\text{n\_vertices}}$,

$$\begin{aligned} \text{maximize} \quad & \eta^\top x \\ \text{subject to} \quad & x = \sum_i \theta_i v_i \\ & \sum_i \theta_i = 1, \theta_i \geq 0 \end{aligned}$$

- Given a polytope $\mathcal{P}$ in H-Rep and a support direction $\eta \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program with a decision variable $x \in \mathbb{R}^{\mathcal{P}.\text{dim}}$,

$$\begin{aligned} \text{maximize} \quad & \eta^\top x \\ \text{subject to} \quad & Ax \leq b \\ & A_e x = b_e \end{aligned}$$

volume()

Compute the volume of the polytope using QHull

Returns
Volume of the polytope

Return type
float

Notes

- Works with V-representation: Yes
- Works with H-representation: No
- Performs a vertex-halfspace enumeration when H-rep is provided
- Returns 0 when the polytope is empty
- Requires polytope to be full-dimensional

property A

Inequality coefficient vectors *A* for the polytope $\{Ax \leq b, A_e x = b_e\}$.

Returns
Inequality coefficient vector (H-Rep). A is np.empty((0, self.dim)) for empty polytope.

Return type
numpy.ndarray

Notes

This function requires the polytope to be in H-Rep, and performs a halfspace enumeration if required.

property Ae

Equality coefficient vectors *Ae* for the polytope $\{Ax \leq b, A_e x = b_e\}$.

Returns

Equality coefficient vector (H-Rep). Ae is np.empty((0, self.dim)) for empty or full-dimensional polytope.

Return type
numpy.ndarray

Notes

This function requires the polytope to be in H-Rep, and performs a halfspace enumeration if required.

property H

Inequality constraints in halfspace representation *H=[A, b]* for the polytope $\{Ax \leq b, A_e x = b_e\}$.

Returns
H-Rep in [A, b]. H is np.empty((0, self.dim + 1)) for empty polytope.

Return type
numpy.ndarray

Notes

This function requires the polytope to be in H-Rep, and performs a halfspace enumeration if required.

property He

Equality constraints in halfspace representation *He=[Ae, be]* for the polytope $\{Ax \leq b, A_e x = b_e\}$.

Returns
H-Rep in [Ae, be]. He is np.empty((0, self.dim + 1)) for empty or full-dimensional polytope.

Return type
numpy.ndarray

Notes

This function requires the polytope to be in H-Rep, and performs a halfspace enumeration if required.

property V

Vertex representation (V) where the polytope is given by ConvexHull(v_i) with v_i as the rows of $V = [v_1; v_2; \ldots; v_{n_vertices}]$.

Returns
Vertices of the polytope, arranged row-wise. V is np.empty((0, self.dim)) if polytope is empty.

Return type
numpy.ndarray

Notes

This function requires the polytope to be in V-Rep, and performs a vertex enumeration if required.

property b

Inequality constants *b* for the polytope $\{Ax \leq b, A_e x = b_e\}$.

Returns
Inequality constants (H-Rep). b is np.empty((0,)) for empty polytope.

Return type
numpy.ndarray

Notes

This function requires the polytope to be in H-Rep, and performs a halfspace enumeration if required.

property be

Equality constants *be* for the polytope $\{Ax \leq b, A_e x = b_e\}$.

Returns
Equality constants (H-Rep). be is np.empty((0,)) for empty or full-dimensional polytope.

Return type
numpy.ndarray

Notes

This function requires the polytope to be in H-Rep, and performs a halfspace enumeration if required.

property cvxpy_args_lp

CVXPY arguments in use when solving a linear program

Returns
CVXPY arguments in use when solving a linear program. Defaults to dictionary in *pycvxset.common.DEFAULT_CVXPY_ARGS_LP*.

Return type
dict

property cvxpy_args_sdp

CVXPY arguments in use when solving a semi-definite program

Returns
CVXPY arguments in use when solving a semi-definite program. Defaults to dictionary in *pycvxset.common.DEFAULT_CVXPY_ARGS_SDP*.

Return type
dict

property cvxpy_args_socp

CVXPY arguments in use when solving a second-order cone program

Returns
CVXPY arguments in use when solving a second-order cone program. Defaults to dictionary in *pycvxset.common.DEFAULT_CVXPY_ARGS_SOCP*.

Return type
dict

property dim

Dimension of the polytope. In H-Rep polytope (A, b), this is the number of columns of A, while in V-Rep, this is the number of components of the vertices.

Returns
Dimension of the polytope

Return type
int

`property in_H_rep`

Check if the polytope have a halfspace representation (H-Rep).

Returns
When True, the polytope has halfspace representation (H-Rep). Otherwise, False.

Return type
bool

`property in_V_rep`

Check if the polytope have a vertex representation (V-Rep).

Returns
When True, the polytope has vertex representation (V-Rep). Otherwise, False.

Return type
bool

`property is_bounded`

Check if the polytope is bounded.

Returns
True if the polytope is bounded, and False otherwise.

Return type
bool

Notes

This property is well-defined when the polytope is in V-Rep, but solves for a minimum volume circumscribing rectangle to check for boundedness.

`property is_empty`

Check if the polytope is empty.

Returns
When True, the polytope is empty

Return type
bool

Notes

This property is well-defined when the polytope is in V-Rep, but may solve a Chebyshev centering problem when the polytope is in H-Rep. may because sometimes we can infer the emptiness of the polytope in H-Rep.

`property is_full_dimensional`

Check if the affine dimension of the polytope is the same as the polytope dimension

Returns
True when the affine hull containing the polytope has the dimension *self.dim*

Return type
bool

Notes

This function can have self to be in V-Rep or H-Rep. See Sec. 2.1.3 of [BV04] for discussion on affine dimension.

An empty polytope is full dimensional if dim=0, otherwise it is not full-dimensional.

When the n-dimensional polytope is in V-rep, it is full-dimensional when its affine dimension is n. Recall that, the affine dimension is the dimension of the affine hull of the polytope is the linear subspace spanned by the vectors formed by subtracting the vertices with one of the vertices. Consequently, its dimension is given by the rank of the matrix P.V[1:] - P.V[0]. When there are fewer than self.dim + 1 vertices, we know it is coplanar without checking for matrix rank (simplex needs at least self.dim + 1 vertices and that is the polytope with the fewest vertices). For numerical stability, we zero-out all delta vertices below PYCVXSET_ZERO.

When the n-dimensional polytope is in H-Rep, it is full-dimensional if it can fit a n-dimensional ball of appropriate center and radius inside it (Chebyshev radius).

property n_equalities

Number of linear equality constraints used to define the polytope $\{Ax \leq b, A_e x = b_e\}$

Returns
Number of linear equality constraints

Return type
int

Notes

A call to this property performs a halfspace enumeration if the polytope is in V-Rep.

property n_halfspaces

Number of halfspaces used to define the polytope $\{Ax \leq b, A_e x = b_e\}$

Returns
Number of halfspaces

Return type
int

Notes

A call to this property performs a halfspace enumeration if the polytope is in V-Rep.

property n_vertices

Number of vertices

Returns
Number of vertices

Return type
int

Notes

A call to this property performs a vertex enumeration if the polytope is in H-Rep.

CHAPTER

TEN

PYCVXSET.ELLIPSOID

class pycvxset.**Ellipsoid**(*c*, ***kwargs*)

Bases: `object`

Ellipsoid class.

We can define a full-dimensional set $\mathcal{P}$ using **one** of the following combinations:

1. (c, Q) for ellipsoid in the **quadratic form** $\mathcal{P} = \{x \in \mathbb{R}^n \mid (x-c)^T Q^{-1}(x-c) \leq 1\}$ with a n-dimensional positive-definite matrix Q and a n-dimensional vector c, and
2. (c, G) for ellipsoid as an **affine transformation of a unit-ball** $\mathcal{P} = \{c + Gu \in \mathbb{R}^n \mid \|u\|_2 \leq 1\}$ with a n-dimensional square lower-triangular G that satisfies $GG^T = Q$, and a n-dimensional vector c.

Parameters

- **c** (*array_like*) – Center of the ellipsoid c. Vector of length (self.dim,)
- **Q** (*array_like, optional*) – Shape matrix of the ellipsoid Q. Symmetric, positive-definite matrix (self.dim times self.dim). If provided, G must not be provided.
- **G** (*array_like, optional*) – Square root of the shape matrix of the ellipsoid G that satisfies $GG^T = Q$. Lower-triangular square matrix (self.dim times self.dim) with non-zero diagonal. If provided, Q must not be provided.

Raises

ValueError – When c or Q or G does not satisfy implicit properties

__init__(*c*, ***kwargs*)

Constructor for Ellipsoid

Methods

__init__(c, **kwargs)	Constructor for Ellipsoid
affine_map(M)	Compute the affine transformation of the given ellipsoid based on a given scalar/matrix.
chebyshev_centering()	Compute the Chebyshev center and radius of the ellipsoid.
closest_point(points[, p])	Wrapper for `project()` to compute the point in the convex set closest to the given point.
containment_constraints(x)	Get CVXPY constraints for containment of x (a cvxpy.Variable) in an ellipsoid.
contains(Q)	Check containment of a set or a collection of points in an ellipsoid.
copy()	Create a copy of the ellipsoid
deflate(set_to_be_centered)	Compute the minimum volume ellipsoid that covers the given set (also known as Lowner-John Ellipsoid).
distance(points[, p])	Wrapper for `project()` to compute distance of a point to a convex set.
extreme(eta)	Wrapper for `support()` to compute the extreme point.
inflate(set_to_be_centered)	Compute the maximum volume ellipsoid that fits within the given set.
inflate_ball(set_to_be_centered)	Compute the largest ball (Chebyshev ball) of a given set.
interior_point()	Compute an interior point to the Ellipsoid
inverse_affine_map_under_invertible_matı	Compute the inverse affine transformation of an ellipsoid based on a given scalar/matrix.
maximum_volume_inscribing_ellipsoid()	Compute the maximum volume inscribing ellipsoid for a given ellipsoid.
minimize(x, objective_to_minimize, cvxpy_args)	Solve a convex program with CVXPY objective subject to containment constraints.
minimum_volume_circumscribing_ellipsoid(	Compute the minimum volume circumscribing ellipsoid for a given ellipsoid.
minimum_volume_circumscribing_rectangle(	Compute the minimum volume circumscribing rectangle for a set.
plot([method, ax, direction_vectors, ...])	Plot a polytopic approximation of the set.
plus(point)	Add a point to an ellipsoid
polytopic_inner_approximation([...])	Compute a polytopic inner-approximation of a given set via ray shooting.
polytopic_outer_approximation([...])	Compute a polytopic outer-approximation of a given set via ray shooting.
project(x[, p])	Project a point or a collection of points on to a set.
projection(project_away_dim)	Orthogonal projection of a set $\mathcal{P}$ after removing some user-specified dimensions.
support(eta)	Evaluates the support function and support vector of a set.
volume()	Compute the volume of the ellipsoid.

Attributes

G	Lower-triangular square root G of the shape matrix Q that satisfies $GG^T = Q$
Q	Shape matrix of the ellipsoid Q
c	Center of the ellipsoid c
cvxpy_args_lp	CVXPY arguments in use when solving a linear program
cvxpy_args_sdp	CVXPY arguments in use when solving a semi-definite program
cvxpy_args_socp	CVXPY arguments in use when solving a second-order cone program
dim	Dimension of the ellipsoid dim
is_empty	Check if the ellipsoid is empty.
is_full_dimensional	Check if the ellipsoid is full-dimensional.

CHAPTER

ELEVEN

PYCVXSET.ELLIPSOID (API DETAILS)

pycvxset.Ellipsoid(c, **kwargs)	Ellipsoid class.

class pycvxset.Ellipsoid.**Ellipsoid**(*c*, ***kwargs*)

Bases: `object`

Ellipsoid class.

We can define a full-dimensional set $\mathcal{P}$ using **one** of the following combinations:

1. (c, Q) for ellipsoid in the **quadratic form** $\mathcal{P} = \{x \in \mathbb{R}^n \mid (x-c)^T Q^{-1}(x-c) \leq 1\}$ with a n-dimensional positive-definite matrix Q and a n-dimensional vector c, and
2. (c, G) for ellipsoid as an **affine transformation of a unit-ball** $\mathcal{P} = \{c + Gu \in \mathbb{R}^n \mid \|u\|_2 \leq 1\}$ with a n-dimensional square lower-triangular G that satisfies $GG^T = Q$, and a n-dimensional vector c.

Parameters

- **c** (*array_like*) – Center of the ellipsoid c. Vector of length (self.dim,)
- **Q** (*array_like, optional*) – Shape matrix of the ellipsoid Q. Symmetric, positive-definite matrix (self.dim times self.dim). If provided, G must not be provided.
- **G** (*array_like, optional*) – Square root of the shape matrix of the ellipsoid G that satisfies $GG^T = Q$. Lower-triangular square matrix (self.dim times self.dim) with non-zero diagonal. If provided, Q must not be provided.

Raises

ValueError – When c or Q or G does not satisfy implicit properties

__init__(*c*, ***kwargs*)

Constructor for Ellipsoid

affine_map(*M*)

Compute the affine transformation of the given ellipsoid based on a given scalar/matrix.

Parameters

M (*int | float | array_like*) – Scalar not equal to zero or matrix (m times self.dim) for the affine map where $1 \leq m \leq$ self.dim.

Raises

- **ValueError** – When M is not convertible into a 2D numpy array of float
- **ValueError** – When M has more rows than self.dim or M has columns not equal to self.dim
- **ValueError** – When M is not full row-rank
- **ValueError** – When M is a scalar but not positive enough

Returns
Affine transformation of the given ellipsoid $\mathcal{R} = M\mathcal{P} = \{Mx | x \in \mathcal{P}\}$

Return type
Ellipsoid

Notes

The matrix M needs to have M has self.dim columns, and full row-rank to preserve full-dimensionality of ellipsoids. See [HJ13rank] for the non-degeneracy sufficient condition on M.

chebyshev_centering()

Compute the Chebyshev center and radius of the ellipsoid.

closest_point(*points*, *p=2*)

Wrapper for *project()* to compute the point in the convex set closest to the given point.

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.
- **p** (*str | int*) – Norm-type. It can be 1, 2, or inf. Defaults to 2.

Returns
Projection of points to the set as a 2D numpy.ndarray. These arrays have as many rows as points.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *project()* function.

containment_constraints(x)

Get CVXPY constraints for containment of x (a cvxpy.Variable) in an ellipsoid.

Parameters
x (*cvxpy.Variable*) – CVXPY variable to be optimized

Returns

A tuple with two items:

1. constraint_list (list): CVXPY constraints for the containment of x in the ellipsoid.
2. xi (cvxpy.Variable): CVXPY variable representing the latent dimension variable.

Return type
tuple

contains(Q)

Check containment of a set or a collection of points in an ellipsoid.

Parameters
Q (*array_like | Polytope | Ellipsoid*) – Polytope/Ellipsoid or a collection of points (each row is a point) to be tested for containment. When providing a collection of points, Q is a matrix (N times self.dim) with each row is a point.

Raises

- **ValueError** – Test point(s) are NOT of the same dimension
- **ValueError** – Test point(s) can not be converted into a 2D numpy array of floats

- **ValueError** – Q is a constrained zonotope
- **NotImplementedError** – Unable to perform ellipsoidal containment check using CVXPY

Returns

An element of the array is True if the point is in the ellipsoid, with as many elements as the number of rows in *test_points*.

Return type
bool or numpy.ndarray[bool]

Notes

- **Containment of a polytope**: This function requires the polytope Q to be in V-Rep. If Q is in H-Rep, a vertex enumeration is performed. This function then checks if all vertices of Q are in the given ellipsoid, which occurs if and only if the polytope is contained within the ellipsoid [BV04].
- **Containment of an ellipsoid**: This function solves a feasibility semi-definite program and requires the ellipsoids to be full-dimensional [BV04].
- **Containment of points**: For each point v, the function checks if there is a $u \in \mathbb{R}^{\mathcal{P}.\dim}$ such that $\|u\|_2$ leq 1 and $Gu + c = v$ [BV04]. This can be efficiently done via numpy.linalg.lstsq.

copy()

Create a copy of the ellipsoid

classmethod deflate(*set_to_be_centered*)

Compute the minimum volume ellipsoid that covers the given set (also known as Lowner-John Ellipsoid).

Parameters
set_to_be_centered (Polytope | ConstrainedZonotope) – Set to be circumscribed.

Returns
Minimum volume circumscribing ellipsoid

Return type
Ellipsoid

Notes

This function is a wrapper for *minimum_volume_circumscribing_ellipsoid()* of the set `set_to_be_centered`. Please check that function for more details including raising exceptions. [EllipsoidalTbx-Min_verticesolEll]

distance(*points*, *p=2*)

Wrapper for *project()* to compute distance of a point to a convex set.

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.
- **p** (*int | str*) – Norm-type. It can be 1, 2, or inf. Defaults to 2.

Returns
Distance of points to the set as a 1D numpy.ndarray. These arrays have as many rows as points.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *project()* function.

extreme(*eta*)

Wrapper for *support()* to compute the extreme point.

Parameters
eta (*array_like*) – Support directions. Matrix (N times self.dim), where each row is a support direction.

Returns
Support vector evaluation(s) as a 2D numpy.ndarray. The array has as many rows as eta.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *support()* function.

classmethod inflate(*set_to_be_centered*)

Compute the maximum volume ellipsoid that fits within the given set.

Parameters
set_to_be_centered (Polytope | ConstrainedZonotope) – Set to be inscribed.

Returns
Maximum volume inscribing ellipsoid

Return type
Ellipsoid

Notes

This function is a wrapper for *maximum_volume_inscribing_ellipsoid()* of the set `set_to_expand_within`. Please check that function for more details including raising exceptions. [EllipsoidalTbx-MinVolEll]

classmethod inflate_ball(*set_to_be_centered*)

Compute the largest ball (Chebyshev ball) of a given set.

Parameters
set_to_be_centered (Polytope | ConstrainedZonotope) – Set to compute Chebyshev ball for.

Returns
Maximum volume inscribing ellipsoid

Return type
Ellipsoid

Notes

This function is a wrapper for `chebyshev_center()` of attr:*set_to_be_centered*. Please check that function for more details including raising exceptions.

interior_point()

Compute an interior point to the Ellipsoid

inverse_affine_map_under_invertible_matrix(*M*)

Compute the inverse affine transformation of an ellipsoid based on a given scalar/matrix.

Parameters
M (*int | float | array_like*) – Scalar or invertible square matrix for the affine map

Raises

- **TypeError** – When M is not convertible into a 2D square numpy matrix
- **TypeError** – When M is not invertible

Returns
Inverse affine transformation of the given ellipsoid $\mathcal{R} = \mathcal{P}M = \{x | Mx \in \mathcal{P}\}$

Return type
Ellipsoid

maximum_volume_inscribing_ellipsoid()

Compute the maximum volume inscribing ellipsoid for a given ellipsoid.

minimize(*x*, *objective_to_minimize*, *cvxpy_args*, *task_str=''*)

Solve a convex program with CVXPY objective subject to containment constraints.

Parameters

- **x** (*cvxpy.Variable*) – CVXPY variable to be optimized
- **objective_to_minimize** (*cvxpy.Expression*) – CVXPY expression to be minimized
- **cvxpy_args** (*dict*) – CVXPY arguments to be passed to the solver
- **task_str** (*str, optional*) – Task string to be used in error messages. Defaults to .

Raises
NotImplementedError – Unable to solve problem using CVXPY

Returns

A tuple with three items:

1. x.value (numpy.ndarray): Optimal value of x. np.nan * np.ones((self.dim,)) if the problem is not solved.
2. problem.value (float): Optimal value of the convex program. np.inf if the problem is infeasible, -np.inf if problem is unbounded, and finite otherwise.
3. problem_status (str): Status of the problem

Return type
tuple

Notes

This function uses *containment_constraints()* to obtain the list of CVXPY expressions that form the containment constraints on x.

Warning

Please pay attention to the NotImplementedError generated by this function. It may be possible to get CVXPY to solve the same problem by switching the solver. For example, consider the following code block.

```
from pycvxset import Polytope
P = Polytope(A=[[1, 1], [-1, -1]], b=[1, 1])
P.cvxpy_args_lp = {'solver': 'CLARABEL'}    # Default solver used in␣
↪pycvxset
try:
    print('Is polytope bounded?', P.is_bounded)
except NotImplementedError as err:
    print(str(err))
P.cvxpy_args_lp = {'solver': 'OSQP'}
print('Is polytope bounded?', P.is_bounded)
```

This code block produces the following output:

```
Unable to solve the task (support function evaluation of the set at eta =␣
↪[-0. -1.]). CVXPY returned error:
Solver 'CLARABEL' failed. Try another solver, or solve with verbose=True␣
↪for more information.

Is polytope bounded? False
```

minimum_volume_circumscribing_ellipsoid()

Compute the minimum volume circumscribing ellipsoid for a given ellipsoid.

minimum_volume_circumscribing_rectangle()

Compute the minimum volume circumscribing rectangle for a set.

Raises

ValueError – When set is empty

Returns

A tuple of two elements

- lb (numpy.ndarray): Lower bound l on the set, $\mathcal{P} \subseteq \{l\} \oplus \mathbb{R}_{\geq 0}$.
- ub (numpy.ndarray): Upper bound u on the set, $\mathcal{P} \subseteq \{u\} \oplus (-\mathbb{R}_{\geq 0})$.

Return type

tuple

Notes

This function computes the lower/upper bound by an element-wise support computation (2n linear programs), where n is attr:*self.dim*. To reuse the *support()* function for the lower bound computation, we solve the optimization for each $i \in \{1, 2, ..., n\}$,

$$\inf_{x \in \mathcal{P}} e_i^\top x = -\sup_{x \in \mathcal{P}} -e_i^\top x = -\rho_{\mathcal{P}}(-e_i),$$

where $e_i \in \mathbb{R}^n$ denotes the standard coordinate vector, and $\rho_{\mathcal{P}}$ is the support function of $\mathcal{P}$.

plot(*method='inner'*, *ax=None*, *direction_vectors=None*, *n_vertices=None*, *n_halfspaces=None*, *patch_args=None*, *vertex_args=None*, *center_args=None*, *autoscale_enable=True*, *decimal_precision=3*)

Plot a polytopic approximation of the set.

Parameters

- **method** (*str, optional*) – Type of polytopic approximation to use. Can be [inner or outer]. Defaults to inner.
- **ax** (*axis object, optional*) – Axis on which the patch is to be plotted
- **direction_vectors** (*array_like, optional*) – Directions to use when performing ray shooting. Matrix (N times self.dim) for some N >= 1. Defaults to None, in which case we use *pycvxset.common.spread_points_on_a_unit_sphere()* to compute the direction vectors.
- **n_vertices** (*int, optional*) – Number of vertices to use when computing the polytopic inner-approximation. Ignored if method is outer or when direction_vectors are provided. More than n_vertices may be used in some cases (see notes). Defaults to None.
- **n_halfspaces** (*int, optional*) – Number of halfspaces to use when computing the polytopic outer-approximation. Ignored if method is outer or when direction_vectors are provided. More than n_halfspaces may be used in some cases (see notes). Defaults to None.
- **patch_args** (*dict, optional*) – Arguments to pass for plotting faces and edges. See *pycvxset.Polytope.Polytope.plot()* for more details. Defaults to None.
- **vertex_args** (*dict, optional*) – Arguments to pass for plotting vertices. See *pycvxset.Polytope.Polytope.plot()* for more details. Defaults to None.
- **center_args** (*dict, optional*) – For ellipsoidal set, arguments to pass to scatter plot for the center. If a label is desired, pass it in center_args.
- **autoscale_enable** (*bool, optional*) – When set to True, matplotlib adjusts axes to view full polytope. See *pycvxset.Polytope.Polytope.plot()* for more details. Defaults to True.
- **decimal_precision** (*int, optional*) – When plotting a 3D polytope that is in V-Rep and not in H-Rep, we round vertex to the specified precision to avoid numerical issues. Defaults to PLOTTING_DECIMAL_PRECISION_CDD specified in pycvxset.common.constants.

Returns

See *pycvxset.Polytope.Polytope.plot()* for details.

Return type

tuple

Notes

This function is a wrapper for *polytopic_inner_approximation()* and *polytopic_outer_approximation()* for more details in polytope construction.

plus(*point*)

Add a point to an ellipsoid

Parameters
point (*array_like*) – Vector (self.dim,) that describes the point to be added.

Raises

- **ValueError** – point is a set (ConstrainedZonotope or Ellipsoid or Polytope)
- **TypeError** – point can not be converted into a numpy array of float
- **ValueError** – point can not be converted into a 1D numpy array of float
- **ValueError** – Mismatch in dimension

Returns
Sum of the ellipsoid and the point.

Return type
Ellipsoid

polytopic_inner_approximation(*direction_vectors=None*, *n_vertices=None*, *verbose=False*)

Compute a polytopic inner-approximation of a given set via ray shooting.

Parameters

- **direction_vectors** (*array_like, optional*) – Directions to use when performing ray shooting. Matrix (N times self.dim) for some $N \geq 1$. Defaults to None.
- **n_vertices** (*int, optional*) – Number of vertices to be used for the inner-approximation. n_vertices is overridden whenever direction_vectors are provided. Defaults to None.
- **verbose** (*bool, optional*) – If true, *pycvxset.common.spread_points_on_a_unit_sphere()* is passed with verbose. Defaults to False.

Returns
Polytopic inner-approximation in V-Rep of a given set with n_vertices no smaller than user-provided n_vertices.

Return type
Polytope

Notes

We compute the polytope using *extreme()* evaluated along the direction vectors computed by *pycvxset.common.spread_points_on_a_unit_sphere()*. When direction_vectors is None and n_vertices is None, we select n_vertices $=$ 2self.dim $+$ $2^{\text{self.dim}}$SPOAUS_DIRECTIONS_PER_QUADRANT (as defined in `pycvxset.common.constants()`). [BV04]

polytopic_outer_approximation(*direction_vectors=None*, *n_halfspaces=None*, *verbose=False*)

Compute a polytopic outer-approximation of a given set via ray shooting.

Parameters

- **direction_vectors** (*array_like, optional*) – Directions to use when performing ray shooting. Matrix (N times self.dim) for some $N \geq 1$. Defaults to None.

- **n_halfspaces** (*int, optional*) – Number of halfspaces to be used for the inner-approximation. n_vertices is overridden whenever direction_vectors are provided. Defaults to None.
- **verbose** (*bool, optional*) – If true, *pycvxset.common.spread_points_on_a_unit_sphere()* is passed with verbose. Defaults to False.

Returns
Polytopic outer-approximation in H-Rep of a given set with n_halfspaces no smaller than user-provided n_vertices.

Return type
Polytope

Notes

We compute the polytope using *support()* evaluated along the direction vectors computed by *pycvxset.common.spread_points_on_a_unit_sphere()*. When direction_vectors is None and n_halfspaces is None, we select n_halfspaces $=$ 2self.dim $+$ $2^{\text{self.dim}}$SPOAUS_DIRECTIONS_PER_QUADRANT (as defined in `pycvxset.common.constants`). [BV04]

project(*x*, *p=2*)

Project a point or a collection of points on to a set.

Given a set $\mathcal{P}$ and a test point $y \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program,

$$\begin{aligned} \text{minimize} \quad & \|x - y\|_p \\ \text{subject to} \quad & x \in \mathcal{P} \end{aligned}$$

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.
- **p** (*str | int*) – Norm-type. It can be 1, 2, or inf. Defaults to 2, which is the Euclidean norm.

Raises

- **ValueError** – Set is empty
- **ValueError** – Dimension mismatch no. of columns in points is different from self.dim.
- **ValueError** – Points is not convertible into a 2D array
- **NotImplementedError** – Unable to solve problem using CVXPY

Returns

A tuple with two items:

1. projected_point (numpy.ndarray): Projection point(s) as a 2D numpy.ndarray. Matrix (N times self.dim), where each row is a projection of the point in points to the set $\mathcal{P}$.
2. distance (numpy.ndarray): Distance(s) as a 1D numpy.ndarray. Vector (N,), where each row is a projection of the point in points to the set $\mathcal{P}$.

Return type
tuple

Notes

For a point $y \in \mathbb{R}^{\mathcal{P}.\text{dim}}$ and an ellipsoid $\mathcal{P} = \{Gu + c \mid \|u\|_2 \leq 1\}$ with $GG^T = Q$, this function solves a convex program with decision variables $x, u \in \mathbb{R}^{\mathcal{P}.\text{dim}}$,

$$\begin{aligned} \text{minimize} \quad & \|x - y\|_p \\ \text{subject to} \quad & x = Gu + c \\ & \|u\|_2 \leq 1 \end{aligned}$$

projection(*project_away_dim*)

Orthogonal projection of a set $\mathcal{P}$ after removing some user-specified dimensions.

$$\mathcal{R} = \{r \in \mathbb{R}^m \mid \exists v \in \mathbb{R}^{n-m},\ \text{Lift}(r, v) \in \mathcal{P}\}$$

Here, $m = \mathcal{P}.\text{dim} - \text{length}(\text{project\_away\_dim})$, and $\text{Lift}(r, v)$ lifts (undos the projection) using the appropriate components of v. This function uses *affine_map()* to implement the projection by designing an appropriate affine map $M \in \{0, 1\}^{m \times \mathcal{P}.\text{dim}}$ with each row of M corresponding to some standard axis vector $e_i \in \mathbb{R}^m$.

Parameters
project_away_dim (*array_like*) – Dimensions to projected away in integer interval [0, 1, , n - 1].

Returns
m-dimensional set obtained via projection.

Return type
Ellipsoid

support(*eta*)

Evaluates the support function and support vector of a set.

The support function of a set $\mathcal{P}$ is defined as $\rho_{\mathcal{P}}(\eta) = \max_{x \in \mathcal{P}} \eta^\top x$. The support vector of a set $\mathcal{P}$ is defined as $\nu_{\mathcal{P}}(\eta) = \arg\max_{x \in \mathcal{P}} \eta^\top x$.

Parameters
eta (*array_like*) – Support directions. Matrix (N times self.dim), where each row is a support direction.

Raises

- **ValueError** – Set is empty
- **ValueError** – Mismatch in eta dimension
- **ValueError** – eta is not convertible into a 2D array
- **NotImplementedError** – Unable to solve problem using CVXPY

Returns

A tuple with two items:

1. support_function_evaluations (numpy.ndarray): Support function evaluation(s) as a 2D numpy.ndarray. Vector (N,) with as many rows as eta.
2. support_vectors (numpy.ndarray): Support vectors as a 2D numpy.ndarray. Matrix N x self.dim with as many rows as eta.

Return type
tuple

Notes

Using duality, the support function and vector of an ellipsoid has a closed-form expressions. For a support direction $\eta \in \mathbb{R}^{\mathcal{P}.\text{dim}}$ and an ellipsoid $\mathcal{P} = \{Gu + c | \|u\|_2 \leq 1\}$ with $GG^T = Q$,

$$\rho_{\mathcal{P}}(\eta) = \eta^\top c + \sqrt{\eta^\top Q \eta} = \eta^\top c + \|G^T \eta\|_2$$
$$\nu_{\mathcal{P}}(\eta) = c + \frac{GG^\top \eta}{\|G^T \eta\|_2} = c + \frac{Q\eta}{\|G^T \eta\|_2}$$

volume()

Compute the volume of the ellipsoid.

Notes

We used the following observations: 1. from [BV04] , the volume of an ellipsoid is proportional to $det(G)$. 2. Square-root of the determinant of the shape matrix coincides with the determinant of G 3. Since G is lower-triangular, its determinant is the product of its diagonal elements.

property G

Lower-triangular square root G of the shape matrix Q that satisfies $GG^T = Q$

property Q

Shape matrix of the ellipsoid Q

property c

Center of the ellipsoid c

property cvxpy_args_lp

CVXPY arguments in use when solving a linear program

Returns
CVXPY arguments in use when solving a linear program. Defaults to dictionary in *pycvxset.common.DEFAULT_CVXPY_ARGS_LP*.

Return type
dict

property cvxpy_args_sdp

CVXPY arguments in use when solving a semi-definite program

Returns
CVXPY arguments in use when solving a semi-definite program. Defaults to dictionary in *pycvxset.common.DEFAULT_CVXPY_ARGS_SDP*.

Return type
dict

property cvxpy_args_socp

CVXPY arguments in use when solving a second-order cone program

Returns
CVXPY arguments in use when solving a second-order cone program. Defaults to dictionary in *pycvxset.common.DEFAULT_CVXPY_ARGS_SOCP*.

Return type
dict

property dim

Dimension of the ellipsoid dim

property is_empty

Check if the ellipsoid is empty. Always False by construction.

property is_full_dimensional

Check if the ellipsoid is full-dimensional. Always True by construction.

CHAPTER

TWELVE

PYCVXSET.CONSTRAINEDZONOTOPE

class pycvxset.**ConstrainedZonotope**(***kwargs*)

Bases: `object`

Constrained zonotope class

Constrained zonotope defines a polytope in the working dimension $\mathbb{R}^n$ as an affine transformation of a polytope defined in latent space $B_\infty(A_e, b_e) \subset \mathbb{R}^{N_C}$. Here, $B_\infty(A_e, b_e)$ is defined as the intersection of a unit ℓ_∞-norm ball and a collection of M_C linear constraints $\{\xi \in \mathbb{R}^{N_C} | A_e\xi = b_e\}$.

Formally, a **constrained zonotope** is defined as follows,

$$\mathcal{C} = \{G\xi + c \mid \xi \in B_\infty(A_e, b_e)\} \subset \mathbb{R}^n,$$

where

$$B_\infty(A_e, b_e) = \{\xi \mid \|\xi\|_\infty \leq 1, A_e\xi = b_e\} \subset \mathbb{R}^{N_C},$$

with $G \in \mathbb{R}^{n \times N_C}$, $c \in \mathbb{R}^n$, $A_e \in \mathbb{R}^{M_C \times N_C}$, and $b \in \mathbb{R}^{M_C}$.

A constrained zonotope provide an alternative and equivalent representation of any convex and compact polytope. Furthermore, a constrained zonotope admits closed-form expressions for several set manipulations that can often be accomplished without invoking any optimization solvers. See [SDGR16] [RK22] [VWD24] for more details.

A **zonotope** is a special class of constrained zonotopes, and are defined as

$$\mathcal{Z} = \{G\xi + c \mid \|\xi\|_\infty \leq 1\} \subset \mathbb{R}^n.$$

In other words, a zonotope is a constrained zonotope with no equality constraints in the latent dimension space. In *ConstrainedZonotope*, we model zonotopes by having (Ae,be) be empty (n_equalities is zero).

Constrained zonotope object construction admits **one** of the following combinations (as keyword arguments):

1. dim for an **empty** constrained zonotope of dimension dim,
2. (G, c, Ae, be) for a **constrained zonotope**,
3. (G, c) for a **zonotope**,
4. (lb, ub) for a **zonotope** equivalent to an **axis-aligned cuboid** with appropriate bounds $\{x \mid lb \leq x \leq ub\}$, and
5. (c, h) for a **zonotope** equivalent to an **axis-aligned cuboid** centered at c with specified scalar/vector half-sides h, $\{x \mid \forall i \in \{1, 2, ..., n\}, |x_i - c_i| \leq h_i\}$.
6. (c=p, G=None) for a **zonotope** equivalent to a **single point** p,
7. P for a **constrained zonotope** equivalent to the *pycvxset.Polytope.Polytope* object P,

Parameters

- **dim** (*int, optional*) – Dimension of the empty constrained zonotope. If NOTHING is provided, dim=0 is assumed.

- **c** (*array_like, optional*) – Affine transformation translation vector. Must be 1D array, and the constrained zonotope dimension is determined by number of elements in c. When c is provided, either (G) or (G, Ae, be) or (h) must be provided additionally. When h is provided, c is the centroid of the resulting zonotope.
- **G** (*array_like*) – Affine transformation matrix. The vectors are stacked vertically with matching number of rows as c. When G is provided, (c, Ae, be) OR (c) must also be provided. To define a constrained zonotope with a single point, set c to the point AND G to None (do not set (Ae, be) or set them to (None, None)).
- **Ae** (*array_like*) – Equality coefficient vectors. The vectors are stacked vertically with matching number of columns as G. When Ae is provided, (G, c, be) must also be provided.
- **be** (*array_like*) – Equality coefficient constants. The constants are expected to be in a 1D numpy array. When be is provided, (G, c, Ae) must also be provided.
- **lb** (*array_like, optional*) – Lower bounds of the axis-aligned cuboid. Must be 1D array, and the constrained zonotope dimension is determined by number of elements in lb. When lb is provided, ub must also be provided.
- **ub** (*array_like, optional*) – Upper bounds of the axis-aligned cuboid. Must be 1D array of length as same as lb. When ub is provided, lb must also be provided.
- **h** (*array_like, optional*) – Half-side length of the axis-aligned cuboid. Can be a scalar or a vector of length as same as c. When h is provided, c must also be provided.
- **polytope** (Polytope, *optional*) – Polytope to use to construct constrained zonotope.

Raises

- **ValueError** – (G, c) is not compatible.
- **ValueError** – (G, c, Ae, be) is not compatible.
- **ValueError** – (lb, ub) is not valid
- **ValueError** – (c, h) is not valid
- **ValueError** – Provided polytope is not bounded.
- **UserWarning** – When a row with all zeros in Ae and be.

__init__(***kwargs*)

Constructor for ConstrainedZonotope class

Methods

__init__(**kwargs)	Constructor for ConstrainedZonotope class
affine_map(M)	Multiply a matrix or a scalar with a constrained zonotope
approximate_pontryagin_difference(norm_ty ...)	Approximate Pontryagin difference between a constrained zonotope and an affine transformation of a unit-norm ball.
chebyshev_centering()	Computes a ball with the largest radius that fits within the constrained zonotope.
closest_point(points[, p])	Wrapper for `project()` to compute the point in the convex set closest to the given point.
containment_constraints(x)	Get CVXPY constraints for containment of x (a cvxpy.Variable) in a constrained zonotope.
contains(Q[, verbose, time_limit])	Check containment of a set $\mathcal{Q}$ (could be a polytope or a constrained zonotope), or a collection of points $Q \in \mathbb{R}^{n_Q \times \mathcal{P}.\text{dim}}$ in the given constrained zonotope.
copy()	Get a copy of the constrained zonotope
distance(points[, p])	Wrapper for `project()` to compute distance of a point to a convex set.
extreme(eta)	Wrapper for `support()` to compute the extreme point.
interior_point()	Compute an interior point of the constrained zonotope.
intersection(Q)	Compute the intersection of constrained zonotope with another constrained zonotope or polytope.
intersection_under_inverse_affine_map(Y, R)	Compute the intersection of constrained zonotope with another constrained zonotope under an inverse affine map
intersection_with_affine_set(Ae, be)	Compute the intersection of a constrained zonotope with an affine set.
intersection_with_halfspaces(A, b)	Compute the intersection of a constrained zonotope with a collection of halfspaces (polyhedron).
inverse_affine_map_under_invertible_matı	Compute the inverse affine map of a constrained zonotope for a given matrix M.
maximum_volume_inscribing_ellipsoid()	Compute the maximum volume ellipsoid that fits within the given constrained zonotope.
minimize(x, objective_to_minimize, cvxpy_args)	Solve a convex program with CVXPY objective subject to containment constraints.
minimum_volume_circumscribing_rectangle(	Compute the minimum volume circumscribing rectangle for a set.
minus(Q)	Pontryagin difference of a constrained zonotope with a set (zonotope) or a point Q.
plot([method, ax, direction_vectors, ...])	Plot a polytopic approximation of the set.
plus(Q)	Add a point or a set Q to a constrained zonotope (Minkowski sum).
polytopic_inner_approximation([...])	Compute a polytopic inner-approximation of a given set via ray shooting.
polytopic_outer_approximation([...])	Compute a polytopic outer-approximation of a given set via ray shooting.
project(x[, p])	Project a point or a collection of points on to a set.
projection(project_away_dim)	Orthogonal projection of a set $\mathcal{P}$ after removing some user-specified dimensions.
remove_redundancies()	Remove any redundancies in the equality system using pycddlib
slice(dims, constants)	Slice a set restricting certain dimensions to constants.
support(eta)	Evaluates the support function and support vector of a set.

Attributes

Ae	Equality coefficient vectors Ae for the constrained zonotope.
G	Affine transformation matrix G for the constrained zonotope.
He	Equality constraints *He=[Ae, be]* for the constrained zonotope.
be	Equality constants be for the constrained zonotope.
c	Affine transformation vector c for the constrained zonotope.
cvxpy_args_lp	CVXPY arguments in use when solving a linear program
cvxpy_args_socp	CVXPY arguments in use when solving a second-order cone program
dim	Dimension of the constrained zonotope.
is_bounded	Check if the constrained zonotope is bounded (which is always True)
is_empty	Check if the constrained zonotope is empty
is_full_dimensional	Check if the affine dimension of the constrained zonotope is the same as the constrained zonotope dimension
is_zonotope	Check if the constrained zonotope is a zonotope
latent_dim	Latent dimension of the constrained zonotope.
n_equalities	Number of equality constraints used when defining the constrained zonotope.

CHAPTER

THIRTEEN

PYCVXSET.CONSTRAINEDZONOTOPE (API DETAILS)

pycvxset.ConstrainedZonotope(**kwargs)	Constrained zonotope class

class pycvxset.ConstrainedZonotope.**ConstrainedZonotope**(***kwargs*)

Bases: `object`

Constrained zonotope class

Constrained zonotope defines a polytope in the working dimension $\mathbb{R}^n$ as an affine transformation of a polytope defined in latent space $B_\infty(A_e, b_e) \subset \mathbb{R}^{N_C}$. Here, $B_\infty(A_e, b_e)$ is defined as the intersection of a unit ℓ_∞-norm ball and a collection of M_C linear constraints $\{\xi \in \mathbb{R}^{N_C} | A_e\xi = b_e\}$.

Formally, a **constrained zonotope** is defined as follows,

$$\mathcal{C} = \{G\xi + c \mid \xi \in B_\infty(A_e, b_e)\} \subset \mathbb{R}^n,$$

where

$$B_\infty(A_e, b_e) = \{\xi \mid \|\xi\|_\infty \leq 1, A_e\xi = b_e\} \subset \mathbb{R}^{N_C},$$

with $G \in \mathbb{R}^{n \times N_C}$, $c \in \mathbb{R}^n$, $A_e \in \mathbb{R}^{M_C \times N_C}$, and $b \in \mathbb{R}^{M_C}$.

A constrained zonotope provide an alternative and equivalent representation of any convex and compact polytope. Furthermore, a constrained zonotope admits closed-form expressions for several set manipulations that can often be accomplished without invoking any optimization solvers. See [SDGR16] [RK22] [VWD24] for more details.

A **zonotope** is a special class of constrained zonotopes, and are defined as

$$\mathcal{Z} = \{G\xi + c \mid \|\xi\|_\infty \leq 1\} \subset \mathbb{R}^n.$$

In other words, a zonotope is a constrained zonotope with no equality constraints in the latent dimension space. In *ConstrainedZonotope*, we model zonotopes by having (Ae,be) be empty (n_equalities is zero).

Constrained zonotope object construction admits **one** of the following combinations (as keyword arguments):

1. dim for an **empty** constrained zonotope of dimension dim,
2. (G, c, Ae, be) for a **constrained zonotope**,
3. (G, c) for a **zonotope**,
4. (lb, ub) for a **zonotope** equivalent to an **axis-aligned cuboid** with appropriate bounds $\{x \mid lb \leq x \leq ub\}$, and
5. (c, h) for a **zonotope** equivalent to an **axis-aligned cuboid** centered at c with specified scalar/vector half-sides h, $\{x \mid \forall i \in \{1, 2, ..., n\}, |x_i - c_i| \leq h_i\}$.
6. (c=p, G=None) for a **zonotope** equivalent to a **single point** p,
7. P for a **constrained zonotope** equivalent to the *pycvxset.Polytope.Polytope* object P,

Parameters

- **dim** (*int, optional*) – Dimension of the empty constrained zonotope. If NOTHING is provided, dim=0 is assumed.
- **c** (*array_like, optional*) – Affine transformation translation vector. Must be 1D array, and the constrained zonotope dimension is determined by number of elements in c. When c is provided, either (G) or (G, Ae, be) or (h) must be provided additionally. When h is provided, c is the centroid of the resulting zonotope.
- **G** (*array_like*) – Affine transformation matrix. The vectors are stacked vertically with matching number of rows as c. When G is provided, (c, Ae, be) OR (c) must also be provided. To define a constrained zonotope with a single point, set c to the point AND G to None (do not set (Ae, be) or set them to (None, None)).
- **Ae** (*array_like*) – Equality coefficient vectors. The vectors are stacked vertically with matching number of columns as G. When Ae is provided, (G, c, be) must also be provided.
- **be** (*array_like*) – Equality coefficient constants. The constants are expected to be in a 1D numpy array. When be is provided, (G, c, Ae) must also be provided.
- **lb** (*array_like, optional*) – Lower bounds of the axis-aligned cuboid. Must be 1D array, and the constrained zonotope dimension is determined by number of elements in lb. When lb is provided, ub must also be provided.
- **ub** (*array_like, optional*) – Upper bounds of the axis-aligned cuboid. Must be 1D array of length as same as lb. When ub is provided, lb must also be provided.
- **h** (*array_like, optional*) – Half-side length of the axis-aligned cuboid. Can be a scalar or a vector of length as same as c. When h is provided, c must also be provided.
- **polytope** (Polytope, *optional*) – Polytope to use to construct constrained zonotope.

Raises

- **ValueError** – (G, c) is not compatible.
- **ValueError** – (G, c, Ae, be) is not compatible.
- **ValueError** – (lb, ub) is not valid
- **ValueError** – (c, h) is not valid
- **ValueError** – Provided polytope is not bounded.
- **UserWarning** – When a row with all zeros in Ae and be.

__init__(***kwargs*)

Constructor for ConstrainedZonotope class

affine_map(*M*)

Multiply a matrix or a scalar with a constrained zonotope

Parameters

M (*array_like*) – Matrix (N times self.dim) or a scalar to be multiplied with a constrained zonotope

Returns

Constrained zonotope which is the product of M and self. Specifically, given a constrained zonotope $\mathcal{P}$, and a matrix $M \in \mathbb{R}^{m \times P.\text{dim}}$ or scalar M, then this function returns a constrained zonotope $\mathcal{R} = \{Mx | x \in \mathcal{P}\}$.

Return type

ConstrainedZonotope

Notes

This function implements (11) of [SDGR16].

approximate_pontryagin_difference(*norm_type*, *G_S*, *c_S*, *method='inner-least-squares'*)

Approximate Pontryagin difference between a constrained zonotope and an affine transformation of a unit-norm ball.

Specifically, we approximate the Pontryagin difference $\mathcal{P} \ominus \mathcal{S} = \{x \in \mathbb{R}^n \mid x + \mathcal{S} \subseteq \mathcal{P}\}$ between a constrained zonotope $\mathcal{P}$ and a subtrahend $\mathcal{S}$. The subtrahend must be specifically of the form $\mathcal{S} = \{G_S\xi + c_S | \|\xi\|_p \leq 1\}$ for norm_type $p \in \{1, 2, \infty\}$.

- For p=1, the subtrahend is a convex hull of intervals specified by the columns of G_S with each interval is symmetric about c_S.
- For p=2, the subtrahend is an ellipsoid is characterized by affine transformation G_S of a unit Euclidean norm ball which is then shifted to c_S. Here, G_S is given by *pycvxset.Ellipsoid.Ellipsoid.G*. See *pycvxset.Ellipsoid.Ellipsoid()* to obtain G_S and c_S for broader classes of ellipsoids.
- For p=inf, the subtrahend is a zonotope characterized by (G_S, c_S).

Parameters

- **norm_type** (*int | str*) – Norm type.
- **G_S** (*array_like*) – Affine transformation matrix (self.dim times N) for scaling the ball
- **c_S** (*array_like*) – Affine transformation vector (self.dim,) for translating the ball
- **method** (*str, optional*) – Approximation method to use. Can be one of [inner-least-squares]. Defaults to inner-least-squares.

Raises

- **ValueError** – Norm type is not in [1, 2, inf]
- **ValueError** – Mismatch in dimension (no. of columns of G_S, length of c_S, and self.dim should match)

Returns

Approximated Pontryagin difference between self and the affine transformation of the unit-norm ball.

Return type

ConstrainedZonotope

Notes

For method inner-least-squares, this function implements Table 1 of [VWD24].

chebyshev_centering()

Computes a ball with the largest radius that fits within the constrained zonotope. The balls center is known as the Chebyshev center, and its radius is the Chebyshev radius.

Raises

- **ValueError** – When the constrained zonotope is not full-dimensional
- **ValueError** – When the constrained zonotope is empty
- **NotImplementedError** – Unable to solve the linear program using CVXPY

Returns

A tuple with two items

1. center (numpy.ndarray): Approximate Chebyshev radius of the constrained zonotope
2. radius (float): Approximate Chebyshev radius of the constrained zonotope

Return type
tuple

Notes

Unlike *pycvxset.Polytope.Polytope.chebyshev_centering()*, this function computes an approximate Chebyshev center and radius. Specifically, it guarantees that a ball of the computed radius, centered at the computed center is contained in the constrained zonotope. However, it does not guarantee that the computed radius is the radius of the largest possible ball contained in the given constrained zonotope. For more details, see [VWS24].

closest_point(*points*, *p=2*)

Wrapper for *project()* to compute the point in the convex set closest to the given point.

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.
- **p** (*str | int*) – Norm-type. It can be 1, 2, or inf. Defaults to 2.

Returns
Projection of points to the set as a 2D numpy.ndarray. These arrays have as many rows as points.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *project()* function.

containment_constraints(*x*)

Get CVXPY constraints for containment of x (a cvxpy.Variable) in a constrained zonotope.

Parameters
x (*cvxpy.Variable*) – CVXPY variable to be optimized

Raises
ValueError – When constrained zonotope is empty

Returns

A tuple with two items:

1. constraint_list (list): CVXPY constraints for the containment of x in the constrained zonotope.
2. xi (cvxpy.Variable | None): CVXPY variable representing the latent dimension variable. It is None, when the constrained zonotope is a single point.

Return type
tuple

contains(*Q*, *verbose=False*, *time_limit=60.0*)

Check containment of a set $\mathcal{Q}$ (could be a polytope or a constrained zonotope), or a collection of points $Q \in \mathbb{R}^{n_Q \times \mathcal{P}.\text{dim}}$ in the given constrained zonotope.

Parameters

- **Q** (*array_like | Polytope | ConstrainedZonotope*) – Polytope object, ConstrainedZonotope object, or a collection of points to be tested for containment within the constrained zonotope. When providing a collection of points, Q is a matrix (N times self.dim) with each row is a point.
- **verbose** (*bool, optional*) – Verbosity flag to provide cvxpy when solving the MIQP related to checking containment of a constrained zonotope within another constrained zonotope. Defaults to False.
- **time_limit** (*int, optional*) – Time limit for GUROBI solver when solving the MIQP related to checking containment of a constrained zonotope within another constrained zonotope. Defaults to 10.

Raises

- **ValueError** – Dimension mismatch between Q and the constrained zonotope
- **ValueError** – Q is Ellipsoid
- **ValueError** – time_limit exceeded when checking containment of two constrained zonotopes
- **NotImplementedError** – GUROBI is not installed when checking containment of two constrained zonotopes
- **NotImplementedError** – Unable to perform containment check between two constrained zonotopes
- **NotImplementedError** – Failed to check containment between two constrained zonotopes, due to an unhandled status

Returns

Boolean corresponding to $\mathcal{Q} \subseteq \mathcal{P}$ or $\mathcal{Q} \in \mathcal{P}$.

Return type

bool | numpy.ndarray([bool])

Notes

- We use $\mathcal{P}$ to denote the constrained zonotope characterized by self.
- When Q is a constrained zonotope, a bool is returned which is True if and only if $\mathcal{Q} \subseteq \mathcal{P}$. This function uses the non-convex programming capabilities of GUROBI (via CVXPY) to check containment between two constrained zonotopes.
- When Q is a polytope, a bool is returned which is True if and only if $\mathcal{Q} \subseteq \mathcal{P}$.
  - When Q is in V-Rep, the containment problem simplifies to checking if all vertices of Q are contained $\mathcal{P}$.
  - When Q is in H-Rep, this function converts Q into a constrained zonotope, and then checks for containment.
- When Q is a single n-dimensional point, a bool is returned which is True if and only if $Q \in \mathcal{P}$. This function uses *distance()* to check containment of a point in a constrained zonotope.
- When Q is a collection of n-dimensional points (Q is a matrix with each row describing a point), an array of is returned where each element is True if and only if $Q_i \in \mathcal{P}$. This function uses *distance()* to check containment of a point in a constrained zonotope.

Warning

This function requires gurobipy when checking if the *ConstrainedZonotope* object represented by *self* contains a *ConstrainedZonotope* object Q or *pycvxset.Polytope.Polytope* object Q in H-Rep.

copy()

Get a copy of the constrained zonotope

distance(*points*, *p=2*)

Wrapper for *project()* to compute distance of a point to a convex set.

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.
- **p** (*int | str*) – Norm-type. It can be 1, 2, or inf. Defaults to 2.

Returns
Distance of points to the set as a 1D numpy.ndarray. These arrays have as many rows as points.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *project()* function.

extreme(*eta*)

Wrapper for *support()* to compute the extreme point.

Parameters
eta (*array_like*) – Support directions. Matrix (N times self.dim), where each row is a support direction.

Returns
Support vector evaluation(s) as a 2D numpy.ndarray. The array has as many rows as eta.

Return type
numpy.ndarray

Notes

For more detailed description, see documentation for *support()* function.

interior_point()

Compute an interior point of the constrained zonotope.

Returns
A point that lies in the (relative) interior of the constrained zonotope.

Return type
numpy.ndarray

Notes

This function returns $Gv + c$, where v is the Chebyshev center of the polytope $B_\infty(A_e, b_e) = \{\xi \mid \|\xi\|_\infty \leq 1, A_e\xi = b_e\} \subset \mathbb{R}^{N_G}$. See *pycvxset.Polytope.Polytope.chebyshev_centering()* for more details on Chebyshev centering.

intersection(Q)

Compute the intersection of constrained zonotope with another constrained zonotope or polytope.

Parameters
Q (ConstrainedZonotope | Polytope) – Set to intersect with

Raises
TypeError – When Q is neither a ConstrainedZonotope object or a Polytope object

Returns
Intersection of self with Q

Return type
ConstrainedZonotope

Notes

- When Q is a constrained zonotope, this function uses *intersection_under_inverse_affine_map()* with R set to identity matrix.
- when Q is a polytope in H-Rep, this function uses *intersection_with_halfspaces()* and *intersection_with_affine_set()*.

intersection_under_inverse_affine_map(Y, R)

Compute the intersection of constrained zonotope with another constrained zonotope under an inverse affine map

Parameters

- **Y** (ConstrainedZonotope) – Set to intersect with
- **R** (*array_like*) – Matrix (Y.dim times self.dim) for the inverse-affine map.

Raises

- **ValueError** – When Y is not a ConstrainedZonotope
- **ValueError** – When R is not of correct dimension

Returns
Intersection of self with Y under an inverse affine map R. Specifically, given constrained zonotopes $\mathcal{P}$ (self) and $\mathcal{Y}$, and a matrix R, we compute the set $\mathcal{P} \cap_R \mathcal{Y} = \{x \in \mathcal{P} | Rx \in \mathcal{Y}\}$. When R is invertible, $\mathcal{P} \cap_R \mathcal{Y} = (R^{-1}\mathcal{P}) \cap \mathcal{Y}$.

Return type
ConstrainedZonotope

Notes

This function implements (13) of [SDGR16]. This function does not require R to be invertible.

intersection_with_affine_set(*Ae*, *be*)

Compute the intersection of a constrained zonotope with an affine set.

Parameters

- **Ae** (*array_like*) – Equality coefficient matrix (N times self.dim) that define the affine set $\{x|A_e x = b_e\}$.
- **be** (*array_like*) – Equality constant vector (N,) that define the affine set $\{x|A_e x = b_e\}$.

Returns
The intersection of a constrained zonotope with the affine set.

Return type
ConstrainedZonotope

Notes

This function implements imposes the constraints $\{A_e x = b_e\}$ as constraints in the latent dimension of the constrained zonotope $A_e(G\xi + c) = b_e$ for every feasible ξ.

intersection_with_halfspaces(*A*, *b*)

Compute the intersection of a constrained zonotope with a collection of halfspaces (polyhedron).

Parameters

- **A** (*array_like*) – Inequality coefficient matrix (N times self.dim) that define the polyhedron $\{x|Ax \leq b\}$.
- **b** (*array_like*) – Inequality constant vector (N,) that define the polyhedron $\{x|Ax \leq b\}$.

Returns
The intersection of a constrained zonotope with a collection of halfspaces.

Return type
ConstrainedZonotope

Notes

This function implements (10) of [RK22] for each halfspace. We skip redundant inequalities when encountered and we return empty set when intersection yields an empty set.

inverse_affine_map_under_invertible_matrix(*M*)

Compute the inverse affine map of a constrained zonotope for a given matrix M.

Parameters
M (*array_like*) – Square invertible matrix of dimension self.dim

Raises

- **ValueError** – M is not convertible into a 2D float array
- **ValueError** – M is not square
- **ValueError** – M is not invertible

Returns
Inverse affine map of self under M. Specifically, Given a constrained zonotope $\mathcal{P}$ and an invertible self.dim-dimensional matrix $M \in \mathbb{R}^{\mathcal{P}.dim \times \mathcal{P}.\text{dim}}$, this function computes the constrained zonotope $\mathcal{R} = M^{-1}\mathcal{P}$.

Return type
ConstrainedZonotope

Notes

This function is a wrapper for *affine_map()*. We require M to be invertible in order to ensure that the resulting set is representable as a constrained zonotope.

maximum_volume_inscribing_ellipsoid()

Compute the maximum volume ellipsoid that fits within the given constrained zonotope.

Raises

- **ValueError** – When the constrained zonotope is not full-dimensional
- **ValueError** – When the constrained zonotope is empty
- **NotImplementedError** – Unable to solve the convex program using CVXPY

Returns

A tuple with three items:

1. center (numpy.ndarray): Approximate maximum volume inscribed ellipsoids center
2. shape_matrix (numpy.ndarray): Approximate maximum volume inscribed ellipsoids shape matrix
3. sqrt_shape_matrix (numpy.ndarray): Approximate maximum volume inscribed ellipsoids square root of shape matrix.

Return type
tuple

Notes

Unlike *pycvxset.Polytope.Polytope.maximum_volume_inscribing_ellipsoid()*, this function computes an approximate maximum volume inscribed ellipsoid. Specifically, it guarantees that the computed ellipsoid is contained in the constrained zonotope. However, it does not guarantee that the computed ellipsoid is the largest possible ellipsoid (in terms of volume) contained in the given constrained zonotope. For more details, see [VWS24].

minimize(*x*, *objective_to_minimize*, *cvxpy_args*, *task_str=''*)

Solve a convex program with CVXPY objective subject to containment constraints.

Parameters

- **x** (*cvxpy.Variable*) – CVXPY variable to be optimized
- **objective_to_minimize** (*cvxpy.Expression*) – CVXPY expression to be minimized
- **cvxpy_args** (*dict*) – CVXPY arguments to be passed to the solver
- **task_str** (*str, optional*) – Task string to be used in error messages. Defaults to .

Raises
NotImplementedError – Unable to solve problem using CVXPY

Returns

A tuple with three items:

1. x.value (numpy.ndarray): Optimal value of x. np.nan * np.ones((self.dim,)) if the problem is not solved.

2. problem.value (float): Optimal value of the convex program. np.inf if the problem is infeasible, -np.inf if problem is unbounded, and finite otherwise.
3. problem_status (str): Status of the problem

Return type
tuple

Notes

This function uses *containment_constraints()* to obtain the list of CVXPY expressions that form the containment constraints on x.

Warning

Please pay attention to the NotImplementedError generated by this function. It may be possible to get CVXPY to solve the same problem by switching the solver. For example, consider the following code block.

```
from pycvxset import Polytope
P = Polytope(A=[[1, 1], [-1, -1]], b=[1, 1])
P.cvxpy_args_lp = {'solver': 'CLARABEL'}    # Default solver used in␣
↪pycvxset
try:
    print('Is polytope bounded?', P.is_bounded)
except NotImplementedError as err:
    print(str(err))
P.cvxpy_args_lp = {'solver': 'OSQP'}
print('Is polytope bounded?', P.is_bounded)
```

This code block produces the following output:

```
Unable to solve the task (support function evaluation of the set at eta =␣
↪[-0. -1.]). CVXPY returned error:
Solver 'CLARABEL' failed. Try another solver, or solve with verbose=True␣
↪for more information.

Is polytope bounded? False
```

minimum_volume_circumscribing_rectangle()

Compute the minimum volume circumscribing rectangle for a set.

Raises
ValueError – When set is empty

Returns

A tuple of two elements

- lb (numpy.ndarray): Lower bound l on the set, $\mathcal{P} \subseteq \{l\} \oplus \mathbb{R}_{\geq 0}$.
- ub (numpy.ndarray): Upper bound u on the set, $\mathcal{P} \subseteq \{u\} \oplus (-\mathbb{R}_{\geq 0})$.

Return type
tuple

Notes

This function computes the lower/upper bound by an element-wise support computation (2n linear programs), where n is attr:*self.dim*. To reuse the *support()* function for the lower bound computation, we solve the optimization for each $i \in \{1, 2, ..., n\}$,

$$\inf_{x \in \mathcal{P}} e_i^\top x = -\sup_{x \in \mathcal{P}} -e_i^\top x = -\rho_{\mathcal{P}}(-e_i),$$

where $e_i \in \mathbb{R}^n$ denotes the standard coordinate vector, and $\rho_{\mathcal{P}}$ is the support function of $\mathcal{P}$.

minus(*Q*)

Pontryagin difference of a constrained zonotope with a set (zonotope) or a point Q.

Parameters
Q (*array_like | Ellipsoid | ConstrainedZonotope*) – Point/set to use as subtrahend in the Pontryagin difference.

Raises

- **TypeError** – When Q is not one of the following convertible into a 1D numpy array, or an ellipsoid, or a zonotope.
- **ValueError** – When Q has a dimension mismatch with self.
- **UserWarning** – When using with Q that is a set, warn that an inner-approximation is returned.

Returns
Pontryagin difference of self and Q

Return type
ConstrainedZonotope

Notes

This function uses *approximate_pontryagin_difference()* when Q is a set and uses *plus()* when Q is a point.

plot(*method='inner'*, *ax=None*, *direction_vectors=None*, *n_vertices=None*, *n_halfspaces=None*, *patch_args=None*, *vertex_args=None*, *center_args=None*, *autoscale_enable=True*, *decimal_precision=3*)

Plot a polytopic approximation of the set.

Parameters

- **method** (*str, optional*) – Type of polytopic approximation to use. Can be [inner or outer]. Defaults to inner.
- **ax** (*axis object, optional*) – Axis on which the patch is to be plotted
- **direction_vectors** (*array_like, optional*) – Directions to use when performing ray shooting. Matrix (N times self.dim) for some N >= 1. Defaults to None, in which case we use *pycvxset.common.spread_points_on_a_unit_sphere()* to compute the direction vectors.
- **n_vertices** (*int, optional*) – Number of vertices to use when computing the polytopic inner-approximation. Ignored if method is outer or when direction_vectors are provided. More than n_vertices may be used in some cases (see notes). Defaults to None.
- **n_halfspaces** (*int, optional*) – Number of halfspaces to use when computing the polytopic outer-approximation. Ignored if method is outer or when direction_vectors are provided. More than n_halfspaces may be used in some cases (see notes). Defaults to None.

- **patch_args** (*dict, optional*) – Arguments to pass for plotting faces and edges. See *pycvxset.Polytope.Polytope.plot()* for more details. Defaults to None.
- **vertex_args** (*dict, optional*) – Arguments to pass for plotting vertices. See *pycvxset.Polytope.Polytope.plot()* for more details. Defaults to None.
- **center_args** (*dict, optional*) – For ellipsoidal set, arguments to pass to scatter plot for the center. If a label is desired, pass it in center_args.
- **autoscale_enable** (*bool, optional*) – When set to True, matplotlib adjusts axes to view full polytope. See *pycvxset.Polytope.Polytope.plot()* for more details. Defaults to True.
- **decimal_precision** (*int, optional*) – When plotting a 3D polytope that is in V-Rep and not in H-Rep, we round vertex to the specified precision to avoid numerical issues. Defaults to PLOTTING_DECIMAL_PRECISION_CDD specified in pycvxset.common.constants.

Returns
See *pycvxset.Polytope.Polytope.plot()* for details.

Return type
tuple

Notes

This function is a wrapper for *polytopic_inner_approximation()* and *polytopic_outer_approximation()* for more details in polytope construction.

plus(*Q*)

Add a point or a set Q to a constrained zonotope (Minkowski sum).

Parameters
Q (*array_like | Polytope | ConstrainedZonotope*) – The point or set to add

Raises
- **TypeError** – When Q is not one of the following convertible into a 1D numpy array or a constrained zonotope.
- **ValueError** – When Q has a dimension mismatch with self.
- **UserWarning** – When using with Q that is a set, warn that an inner-approximation is returned.

Returns
Minkowski sum of self and Q.

Return type
ConstrainedZonotope

Notes

Given a constrained zonotope $\mathcal{P}$ and a set Q, this function computes the Minkowski sum of Q and the constrained zonotope, defined as $\mathcal{R} = \{x + q | x \in \mathcal{P}, q \in \mathcal{Q}\}$. On the other hand, when Q is a point, this function computes the constrained zonotope $\mathcal{R} = \{x + Q | x \in \mathcal{P}\}$.

This function implements (12) of [SDGR16] when Q is a constrained zonotope. When Q is a Polytope in H-Rep, this function converts it into a constrained zonotope, and then uses (12) of [SDGR16].

polytopic_inner_approximation(*direction_vectors=None*, *n_vertices=None*, *verbose=False*)

Compute a polytopic inner-approximation of a given set via ray shooting.

Parameters

- **direction_vectors** (*array_like, optional*) – Directions to use when performing ray shooting. Matrix (N times self.dim) for some $N \geq 1$. Defaults to None.
- **n_vertices** (*int, optional*) – Number of vertices to be used for the inner-approximation. n_vertices is overridden whenever direction_vectors are provided. Defaults to None.
- **verbose** (*bool, optional*) – If true, *pycvxset.common.spread_points_on_a_unit_sphere()* is passed with verbose. Defaults to False.

Returns
Polytopic inner-approximation in V-Rep of a given set with n_vertices no smaller than user-provided n_vertices.

Return type
Polytope

Notes

We compute the polytope using *extreme()* evaluated along the direction vectors computed by *pycvxset.common.spread_points_on_a_unit_sphere()*. When direction_vectors is None and n_vertices is None, we select n_vertices = 2self.dim + $2^{\text{self.dim}}$SPOAUS_DIRECTIONS_PER_QUADRANT (as defined in `pycvxset.common.constants()`). [BV04]

polytopic_outer_approximation(*direction_vectors=None*, *n_halfspaces=None*, *verbose=False*)

Compute a polytopic outer-approximation of a given set via ray shooting.

Parameters

- **direction_vectors** (*array_like, optional*) – Directions to use when performing ray shooting. Matrix (N times self.dim) for some $N \geq 1$. Defaults to None.
- **n_halfspaces** (*int, optional*) – Number of halfspaces to be used for the inner-approximation. n_vertices is overridden whenever direction_vectors are provided. Defaults to None.
- **verbose** (*bool, optional*) – If true, *pycvxset.common.spread_points_on_a_unit_sphere()* is passed with verbose. Defaults to False.

Returns
Polytopic outer-approximation in H-Rep of a given set with n_halfspaces no smaller than user-provided n_vertices.

Return type
Polytope

Notes

We compute the polytope using *support()* evaluated along the direction vectors computed by *pycvxset.common.spread_points_on_a_unit_sphere()*. When direction_vectors is None and n_halfspaces is None, we select n_halfspaces = 2self.dim + $2^{\text{self.dim}}$SPOAUS_DIRECTIONS_PER_QUADRANT (as defined in `pycvxset.common.constants`). [BV04]

project(*x*, *p=2*)

Project a point or a collection of points on to a set.

Given a set $\mathcal{P}$ and a test point $y \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program,

$$\begin{aligned} \text{minimize} \quad & \|x - y\|_p \\ \text{subject to} \quad & x \in \mathcal{P} \end{aligned}$$

Parameters

- **points** (*array_like*) – Points to project. Matrix (N times self.dim), where each row is a point.
- **p** (*str | int*) – Norm-type. It can be 1, 2, or inf. Defaults to 2, which is the Euclidean norm.

Raises

- **ValueError** – Set is empty
- **ValueError** – Dimension mismatch no. of columns in points is different from self.dim.
- **ValueError** – Points is not convertible into a 2D array
- **NotImplementedError** – Unable to solve problem using CVXPY

Returns

A tuple with two items:

1. projected_point (numpy.ndarray): Projection point(s) as a 2D numpy.ndarray. Matrix (N times self.dim), where each row is a projection of the point in points to the set $\mathcal{P}$.
2. distance (numpy.ndarray): Distance(s) as a 1D numpy.ndarray. Vector (N,), where each row is a projection of the point in points to the set $\mathcal{P}$.

Return type
tuple

Notes

Given a constrained zonotope $\mathcal{P}$ and a test point $y \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program with decision variables $x \in \mathbb{R}^{\mathcal{P}.\text{dim}}$ and $\xi \in \mathbb{R}^{\mathcal{P}.\text{latent\_dim}}$,

$$\begin{aligned} \text{minimize} \quad & \|x - y\|_p \\ \text{subject to} \quad & x = G_P \xi + c_P \\ & A_P \xi = b_P \\ & \|\xi\|_\infty \leq 1 \end{aligned}$$

projection(*project_away_dim*)

Orthogonal projection of a set $\mathcal{P}$ after removing some user-specified dimensions.

$$\mathcal{R} = \{r \in \mathbb{R}^m \mid \exists v \in \mathbb{R}^{n-m},\ \text{Lift}(r, v) \in \mathcal{P}\}$$

Here, $m = \mathcal{P}.\text{dim} - \text{length}(\text{project\_away\_dim})$, and $\text{Lift}(r, v)$ lifts (undos the projection) using the appropriate components of v. This function uses *affine_map()* to implement the projection by designing an appropriate affine map $M \in \{0, 1\}^{m \times \mathcal{P}.\text{dim}}$ with each row of M corresponding to some standard axis vector $e_i \in \mathbb{R}^m$.

Parameters
project_away_dim (*array_like*) – Dimensions to projected away in integer interval [0, 1, , n - 1].

Returns
m-dimensional set obtained via projection.

Return type
ConstrainedZonotope

`remove_redundancies`()

Remove any redundancies in the equality system using pycddlib

`slice`(*dims*, *constants*)

Slice a set restricting certain dimensions to constants.

Parameters

- **dims** (*array_like*) – List of dims to restrict to a constant in the integer interval [0, 1, , n - 1].
- **constants** (*array_like*) – List of constants

Raises

- **ValueError** – dims has entries beyond n
- **ValueError** – dims and constants are not 1D arrays of same size

Returns
Constrained zonotope that has been sliced at the specified dimensions.

Return type
ConstrainedZonotope

Notes

This function uses *intersection_with_affine_set()* to implement the slicing by designing an appropriate affine set from dims and constants.

`support`(*eta*)

Evaluates the support function and support vector of a set.

The support function of a set $\mathcal{P}$ is defined as $\rho_{\mathcal{P}}(\eta) = \max_{x \in \mathcal{P}} \eta^\top x$. The support vector of a set $\mathcal{P}$ is defined as $\nu_{\mathcal{P}}(\eta) = \arg\max_{x \in \mathcal{P}} \eta^\top x$.

Parameters
eta (*array_like*) – Support directions. Matrix (N times self.dim), where each row is a support direction.

Raises

- **ValueError** – Set is empty
- **ValueError** – Mismatch in eta dimension
- **ValueError** – eta is not convertible into a 2D array
- **NotImplementedError** – Unable to solve problem using CVXPY

Returns

A tuple with two items:

1. support_function_evaluations (numpy.ndarray): Support function evaluation(s) as a 2D numpy.ndarray. Vector (N,) with as many rows as eta.
2. support_vectors (numpy.ndarray): Support vectors as a 2D numpy.ndarray. Matrix N x self.dim with as many rows as eta.

Return type
tuple

Notes

Given a constrained zonotope $\mathcal{P}$ and a support direction $\eta \in \mathbb{R}^{\mathcal{P}.\text{dim}}$, this function solves a convex program with decision variables $x \in \mathbb{R}^{\mathcal{P}.\text{dim}}$ and $\xi \in \mathbb{R}^{\mathcal{P}.\text{latent\_dim}}$,

$$\begin{aligned} \text{maximize} \quad & \eta^\top x \\ \text{subject to} \quad & x = G_P \xi + c_P \\ & A_P \xi = b_P \\ & \|\xi\|_\infty \leq 1 \end{aligned}$$

property Ae

Equality coefficient vectors Ae for the constrained zonotope.

Returns
Equality coefficient vectors Ae for the constrained zonotope. Ae is np.empty((0, self.latent_dim)) for a zonotope.

Return type
numpy.ndarray

property G

Affine transformation matrix G for the constrained zonotope.

Returns
Affine transformation matrix G.

Return type
numpy.ndarray

property He

Equality constraints *He=[Ae, be]* for the constrained zonotope.

Returns
H-Rep in [Ae, be]. He is np.empty((0, self.latent_dim + 1)) for a zonotope.

Return type
numpy.ndarray

property be

Equality constants be for the constrained zonotope.

Returns
Equality constants be for the constrained zonotope. be is np.empty((0,)) for a zonotope.

Return type
numpy.ndarray

property c

Affine transformation vector c for the constrained zonotope.

Returns
Affine transformation vector c.

Return type
numpy.ndarray

property cvxpy_args_lp

CVXPY arguments in use when solving a linear program

Returns
CVXPY arguments in use when solving a linear program. Defaults to dictionary in *pycvxset.common.DEFAULT_CVXPY_ARGS_LP*.

Return type
dict

property cvxpy_args_socp

CVXPY arguments in use when solving a second-order cone program

Returns
CVXPY arguments in use when solving a second-order cone program. Defaults to dictionary in *pycvxset.common.DEFAULT_CVXPY_ARGS_SOCP*.

Return type
dict

property dim

Dimension of the constrained zonotope.

Returns
Dimension of the constrained zonotope.

Return type
int

Notes

We determine dimension from G, since c is set to None in case of empty (constrained) zonotope.

property is_bounded

Check if the constrained zonotope is bounded (which is always True)

property is_empty

Check if the constrained zonotope is empty

Raises
NotImplementedError – Unable to solve the feasibility problem using CVXPY

Returns
When True, the polytope is empty

Return type
bool

property is_full_dimensional

Check if the affine dimension of the constrained zonotope is the same as the constrained zonotope dimension

Raises
NotImplementedError – When full-dimensionality needs to be checked for a general constrained zonotope

Returns
True when the affine hull containing the constrained zonotope has the dimension *self.dim*

Return type
bool

Notes

An empty polytope is full dimensional if dim=0, otherwise it is not full-dimensional. See Sec. 2.1.3 of [BV04] for discussion on affine dimension.

A constrained zonotope is full-dimensional, then G has full row rank and the solution set of $B_\infty(Ae, be)$ is at least n-dimensional, i.e., Ae has a row-rank of

property is_zonotope

Check if the constrained zonotope is a zonotope

property latent_dim

Latent dimension of the constrained zonotope.

Returns
Latent dimension of the constrained zonotope.

Return type
int

property n_equalities

Number of equality constraints used when defining the constrained zonotope.

Returns
Number of equality constraints used when defining the constrained zonotope

Return type
int

CHAPTER

FOURTEEN

PYCVXSET.COMMON

The following functions are few additional methods provided with pycvxset that may be useful to the user.

pycvxset.common.approximate_volume_from_gr	Estimate area of a two-dimensional set using a grid of given step size
pycvxset.common.is_constrained_zonotope(Q)	Check if the set is a constrained zonotope
pycvxset.common.is_ellipsoid(Q)	Check if the set is an ellipsoid
pycvxset.common.is_polytope(Q)	Check if the set is a polytope
pycvxset.common.prune_and_round_vertices(V	Filter through the vertices to skip any point that has another point (down in the list) that is close to it in the list.
pycvxset.common.spread_points_on_a_unit_sp	Spread points on a unit sphere in n-dimensional space.

14.1 pycvxset.common.approximate_volume_from_grid

pycvxset.common.**approximate_volume_from_grid**(*set_to_compute_area_for*, *area_grid_step_size*)

Estimate area of a two-dimensional set using a grid of given step size

Parameters

- **set_to_compute_area_for** (Polytope | ConstrainedZonotope | Ellipsoid) – Set for which area is to be computed
- **area_grid_step_size** (*float*) – Scalar step size that is constant in both dimensions

Raises

ValueError – When set is not 2-dimensional

Returns

Approximate area of the set

Return type

float

Notes

This function creates a 2D grid of points with a given step size, and computes the fraction of points that lie in the set. Consequently, the computed area is an approximation of the actual area of the set. The area of the bounding box is computed using the `minimum_volume_circumscribing_rectangle()` method associated with the set.

14.2 pycvxset.common.is_constrained_zonotope

pycvxset.common.**is_constrained_zonotope**(*Q*)

Check if the set is a constrained zonotope

Parameters
Q (*object*) – Set to check

Returns
Returns True if the set is a constrained zonotope, False otherwise

Return type
bool

14.3 pycvxset.common.is_ellipsoid

pycvxset.common.**is_ellipsoid**(*Q*)

Check if the set is an ellipsoid

Parameters
Q (*object*) – Set to check

Returns
Returns True if the set is an ellipsoid, False otherwise

Return type
bool

14.4 pycvxset.common.is_polytope

pycvxset.common.**is_polytope**(*Q*)

Check if the set is a polytope

Parameters
Q (*object*) – Set to check

Returns
Returns True if the set is a polytope, False otherwise

Return type
bool

14.5 pycvxset.common.prune_and_round_vertices

pycvxset.common.**prune_and_round_vertices**(*V*, *decimal_precision=3*)

Filter through the vertices to skip any point that has another point (down in the list) that is close to it in the list.

Parameters

- **V** (*array_like*) – Matrix of vertices (N times self.dim)
- **decimal_precision** (*int, optional*) – The decimal precision for rounding the vertices. Defaults to PLOTTING_DECIMAL_PRECISION_CDD from pycvxset.common.constants.

Returns
The pruned and rounded array of vertices.

Return type
numpy.ndarray

14.6 pycvxset.common.spread_points_on_a_unit_sphere

pycvxset.common.**spread_points_on_a_unit_sphere**(*n_dim*, *n_points=None*, *cvxpy_socp_args=None*, *verbose=False*)

Spread points on a unit sphere in n-dimensional space.

Parameters

- **n_dim** (*int*) – The dimension of the sphere.
- **n_points** (*int*) – The number of points to be spread on the unit sphere. Defaults to None, in which case, we choose
- **=** (*n_points*)
- **cvxpy_socp_args** (*dict*) – Additional arguments to be passed to the CVXPY solver. Defaults to None, in which case the function uses DEFAULT_CVXPY_ARGS_SOCP from pycvxset.common.constants.
- **verbose** (*bool, optional*) – Whether to print verbose output. Defaults to False.

Returns

A tuple containing three items:
opt_locations (ndarray): The spread points on the unit sphere. # minimum_separation (float): The minimum separation between the points. # opt_locations_first_quad (ndarray): The spread points in the first quadrant.

Return type
tuple

Raises

- **ValueError** – If n_points is less than 2 * n_dim.
- **UserWarning** – If n_points - 2 * n_dim is not a multiple of 2^n_dim.
- **NotImplementedError** – Unable to solve the convexified problem using CVXPY
- **NotImplementedError** – Convex-concave procedure did not converge!

Notes

This function uses the CVXPY library to solve a convex optimization problem to spread the points on the unit sphere. The spread points are returned as opt_locations, the separation between the points is returned as separation, and the spread points in the first quadrant are returned as opt_locations_first_quad.

For n_dim in [1, 2], the points are available in closed-form. For n_dim >= 3, we solve the following non-convex optimization problem using convex-concave procedure:

$$
\begin{array}{lll}
\text{maximize} & R & \\
\text{subject to} & R \geq 0 & \\
& x \geq R/2 & \\
& \|x_i - x_j\| \geq R, & 1 \leq i < j \leq n_{\text{points}} \\
& \|x_i - e_j\| \geq R, & 1 \leq i \leq n_{\text{points}}, 1 \leq j \leq n_{\text{dim}} \\
& \|x_i\| \leq 1, & 1 \leq i \leq n_{\text{points}} \\
& \|x_i\| \geq 0.8, & 1 \leq i \leq n_{\text{points}}
\end{array}
$$

The optimization problem seeks to spread points (apart from the standard axes) in the first quadrant so that their pairwise separation is maximized, while they have a norm close to 1. The second constraint is motivated to ensure that the reflections of the points about the quadrant plane is also separated by R.

pycvxset.common.**approximate_volume_from_grid**(*set_to_compute_area_for*, *area_grid_step_size*)

Estimate area of a two-dimensional set using a grid of given step size

Parameters

- **set_to_compute_area_for** (Polytope | ConstrainedZonotope | Ellipsoid) – Set for which area is to be computed
- **area_grid_step_size** (*float*) – Scalar step size that is constant in both dimensions

Raises
ValueError – When set is not 2-dimensional

Returns
Approximate area of the set

Return type
float

Notes

This function creates a 2D grid of points with a given step size, and computes the fraction of points that lie in the set. Consequently, the computed area is an approximation of the actual area of the set. The area of the bounding box is computed using the `minimum_volume_circumscribing_rectangle()` method associated with the set.

pycvxset.common.**is_constrained_zonotope**(*Q*)

Check if the set is a constrained zonotope

Parameters
Q (*object*) – Set to check

Returns
Returns True if the set is a constrained zonotope, False otherwise

Return type
bool

pycvxset.common.**is_ellipsoid**(*Q*)

Check if the set is an ellipsoid

Parameters
Q (*object*) – Set to check

Returns
Returns True if the set is an ellipsoid, False otherwise

Return type
bool

pycvxset.common.**is_polytope**(*Q*)

Check if the set is a polytope

Parameters
Q (*object*) – Set to check

Returns
Returns True if the set is a polytope, False otherwise

Return type
bool

pycvxset.common.**prune_and_round_vertices**(*V*, *decimal_precision=3*)

Filter through the vertices to skip any point that has another point (down in the list) that is close to it in the list.

Parameters

- **V** (*array_like*) – Matrix of vertices (N times self.dim)
- **decimal_precision** (*int, optional*) – The decimal precision for rounding the vertices. Defaults to PLOTTING_DECIMAL_PRECISION_CDD from pycvxset.common.constants.

Returns
The pruned and rounded array of vertices.

Return type
numpy.ndarray

pycvxset.common.**spread_points_on_a_unit_sphere**(*n_dim*, *n_points=None*, *cvxpy_socp_args=None*, *verbose=False*)

Spread points on a unit sphere in n-dimensional space.

Parameters

- **n_dim** (*int*) – The dimension of the sphere.
- **n_points** (*int*) – The number of points to be spread on the unit sphere. Defaults to None, in which case, we choose
- **=** (*n_points*)
- **cvxpy_socp_args** (*dict*) – Additional arguments to be passed to the CVXPY solver. Defaults to None, in which case the function uses DEFAULT_CVXPY_ARGS_SOCP from pycvxset.common.constants.
- **verbose** (*bool, optional*) – Whether to print verbose output. Defaults to False.

Returns

A tuple containing three items:
opt_locations (ndarray): The spread points on the unit sphere. # minimum_separation (float): The minimum separation between the points. # opt_locations_first_quad (ndarray): The spread points in the first quadrant.

Return type
tuple

Raises

- **ValueError** – If n_points is less than 2 * n_dim.
- **UserWarning** – If n_points - 2 * n_dim is not a multiple of 2^n_dim.
- **NotImplementedError** – Unable to solve the convexified problem using CVXPY
- **NotImplementedError** – Convex-concave procedure did not converge!

Notes

This function uses the CVXPY library to solve a convex optimization problem to spread the points on the unit sphere. The spread points are returned as opt_locations, the separation between the points is returned as separation, and the spread points in the first quadrant are returned as opt_locations_first_quad.

For n_dim in [1, 2], the points are available in closed-form. For n_dim >= 3, we solve the following

non-convex optimization problem using convex-concave procedure:

$$
\begin{array}{ll}
\text{maximize} & R \\
\text{subject to} & R \geq 0 \\
& x \geq R/2 \\
& \|x_i - x_j\| \geq R, \qquad 1 \leq i < j \leq n_{\text{points}} \\
& \|x_i - e_j\| \geq R, \qquad 1 \leq i \leq n_{\text{points}}, 1 \leq j \leq n_{\text{dim}} \\
& \|x_i\| \leq 1, \qquad 1 \leq i \leq n_{\text{points}} \\
& \|x_i\| \geq 0.8, \qquad 1 \leq i \leq n_{\text{points}}
\end{array}
$$

The optimization problem seeks to spread points (apart from the standard axes) in the first quadrant so that their pairwise separation is maximized, while they have a norm close to 1. The second constraint is motivated to ensure that the reflections of the points about the quadrant plane is also separated by R.

CHAPTER

FIFTEEN

PYCVXSET.COMMON.CONSTANTS

The following provides the default constants used in pycvxset.

```python
# Copyright (C) 2020-2024 Mitsubishi Electric Research Laboratories (MERL)
#
# SPDX-License-Identifier: AGPL-3.0-or-later

# Code purpose:  Specify the constants to be used with cvxpy when optimization problems as well as testing workflows

PYCVXSET_ZERO = 1e-6  # Zero threshold for numerical stability
PYCVXSET_ZERO_GUROBI = 1e-5  # Zero threshold for numerical stability when using GUROBI
# Zero threshold for vertex clustering when computing polytopic inner-approximation of constrained zonotopes
PYCVXSET_ZERO_CDD = 1e-5
PLOTTING_DECIMAL_PRECISION_CDD = 3

# Solvers used by default
DEFAULT_LP_SOLVER_STR = "CLARABEL"  # CLARABEL, MOSEK, CVXOPT, SCS, ECOS, GUROBI, OSQP
DEFAULT_SOCP_SOLVER_STR = "CLARABEL"  # CLARABEL, MOSEK, CVXOPT, SCS, ECOS, GUROBI
DEFAULT_SDP_SOLVER_STR = "SCS"  # CLARABEL, MOSEK, CVXOPT, SCS

# CVXPY args used by default (use "reoptimize": True when using GUROBI)
DEFAULT_CVXPY_ARGS_LP = {"solver": DEFAULT_LP_SOLVER_STR}
DEFAULT_CVXPY_ARGS_SOCP = {"solver": DEFAULT_SOCP_SOLVER_STR}
DEFAULT_CVXPY_ARGS_SDP = {"solver": DEFAULT_SDP_SOLVER_STR}

# Time limit for GUROBI when checking for containment of a constrained zonotope in another constrained zonotope
TIME_LIMIT_FOR_GUROBI_NON_CONVEX = 60.0

# Constants for spread_points_on_a_unit_sphere (SPOAUS)
SPOAUS_SLACK_TOLERANCE = 1e-8
SPOAUS_COST_TOLERANCE = 1e-5
SPOAUS_INITIAL_TAU = 1.0
SPOAUS_SCALING_TAU = 1.1
SPOAUS_TAU_MAX = 1e4
SPOAUS_ITERATIONS_AT_TAU_MAX = 20
SPOAUS_MINIMUM_NORM_VALUE_SQR = 0.8**2
# For SPOAUS_DIRECTIONS_PER_QUADRANT=20, we have 2D = 84, 3D = 166, 4D = 328, 5D = 650
SPOAUS_DIRECTIONS_PER_QUADRANT = 20

# Testing workflow constants | You could also do "GUROBI" in cvxpy.installed_solvers()
TESTING_CONTAINMENT_STATEMENTS_INVOLVING_GUROBI = False
TESTING_SHOW_PLOTS = False
```

(continues on next page)

(continued from previous page)

```
TEST_3D_PLOTTING = False

# Plotting constants for polytopes
DEFAULT_PATCH_ARGS_2D = {"edgecolor": "k", "facecolor": "skyblue"}
DEFAULT_PATCH_ARGS_3D = {"edgecolor": "k", "facecolor": None}
DEFAULT_VERTEX_ARGS = {"visible": False, "s": 30, "marker": "o", "color": "k"}
```

CHAPTER

SIXTEEN

TUTORIALS

We provide several tutorials to help you get started with *pycvxset*. The tutorials are designed to be run in a Jupyter notebook, which allows you to interact with the code and experiment with the library. The tutorials are available in the *docs/examples/* directory of *pycvxset*.

For your convenience, we provide a link to the tutorials below:

1. Polytope
2. ConstrainedZonotope
3. Application: Reachability using Polytope/ConstrainedZonotope classes

In the above tutorials, we demonstrate how to use *pycvxset* to perform various operations on polytopes, ellipsoids, and constrained zonotopes. We also provide an application example that demonstrates the use of *pycvxset* in reachability analysis. The tutorials are designed to be self-contained, and we encourage you to run them in a Jupyter notebook to understand the code and experiment with the library.

CHAPTER

SEVENTEEN

REFERENCES

BIBLIOGRAPHY

[BV04] Boyd, Stephen, and Lieven Vandenberghe. Convex optimization. Cambridge University Press, 2004.

[HJ13rank] Horn, Roger A., and Charles R. Johnson. Matrix analysis. Cambridge university press, 2012, p. 431, Observation 7.1.8.

[KG98] Kolmanovsky, Ilya, and Elmer G. Gilbert. Theory and computation of disturbance invariant sets for discretetime linear systems. Mathematical problems in engineering 4, no. 4 (1998): 317-367.

[RK22] Raghuraman, Vignesh, and Koeln, Justin P.. Set operations and order reductions for constrained zonotopes. Automatica 139 (2022): 110204.

[SDGR16] Scott, Joseph K., Davide M. Raimondo, Giuseppe Roberto Marseglia, and Richard D. Braatz. Constrained zonotopes: A new tool for set-based estimation and fault detection. Automatica 69 (2016): 126-136.

[VWD24] Vinod, Abraham P., Avishai Weiss, and Stefano Di Cairano. Projection-free computation of robust controllable sets with constrained zonotopes. arXiv preprint arXiv:2403.13730 (2024).

[VWS24] Vinod, Abraham P., Avishai Weiss, and Stefano Di Cairano. Inscribing and separating an ellipsoid and a constrained zonotope: Applications in stochastic control and centering IEEE Conference on Decision and Control, 2024 (accepted).

[Matplotlib-Line3DCollection] Matplotlib documentation for Line3DCollection (used in 3D plotting) <https://matplotlib.org/stable/api/_as_gen/mpl_toolkits.mplot3d.art3d.Line3DCollection.html#mpl_toolkits.mplot3d.art3d.Line3DCollection.set>

[Matplotlib-Axes3D.scatter] Matplotlib documentation for Axes3D.scatter (used in 3D plotting) <https://matplotlib.org/stable/api/_as_gen/mpl_toolkits.mplot3d.axes3d.Axes3D.scatter.html#mpl_toolkits.mplot3d.axes3d.Axes3D.scatter>

[Matplotlib-patch] Matplotlib documentation for options to pass with Axis.add_patch (used in 2D plotting) <https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.Patch.html#matplotlib-patches-patch>

[Matplotlib-scatter] Matplotlib documentation for Axis.scatter (used in 2D plotting) <https://matplotlib.org/stable/api/_as_gen/mpl_toolkits.mplot3d.art3d.Line3DCollection.html#mpl_toolkits.mplot3d.art3d.Line3DCollection.set>

[EllipsoidalTbx-MinVolEll] Documentation for minimum-volume-ellipsoids in Ellipsoidal Toolbox. <https://systemanalysisdpt-cmc-msu.github.io/ellipsoids/doc/chap_ellcalc.html#minimum-volume-ellipsoids>.

[EllipsoidalTbx-Min_verticesolEll] Documentation for minimum-volume-ellipsoids in Ellipsoidal Toolbox. <https://systemanalysisdpt-cmc-msu.github.io/ellipsoids/doc/chap_ellcalc.html#minimum-volume-ellipsoids>.

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

G

H

I

L

M

N

P

Q

R

S

V