Code explanations

First file

1.1 solving quadratic optimization,py

The code formulates a quadratic programming problem with equality and inequality constraints using sparse matrices. It converts bound constraints to inequality constraints, combines all constraints, and then solves the problem using the **osqp** solver. The problem aims to minimize a quadratic objective function subject to linear equality and inequality constraints. Finally, it prints the optimal solution. This approach efficiently handles large-scale optimization problems while maintaining computational efficiency, particularly useful in scenarios where sparse matrices are prevalent, reducing memory consumption and computational burden.

Second file

1.2 operations with polyhedral.py

The code defines two polytopes in two-dimensional space represented by sets of linear inequalities. It checks if one polytope is a subset of another. Each polytope is visualized on a plot with their respective inequality lines. The function **plot_polytope** generates points along each inequality line and plots them. The first polytope is shown in blue, while the second is in red, distinguished by dashed lines. The plot axes are limited to focus on the region of interest, and labels and a title provide context. This visualization aids in understanding the geometric relationship between the two polytopes, showcasing their intersection and relative positions in the space.

Third file

1.3 controlsytems.py

The code implements Model Predictive Control (MPC) to compute optimal control inputs for a dynamical system defined by state-space matrices. It formulates an optimization problem where the objective is minimized (currently set to zero) subject to dynamic constraints and bounds on both states and inputs over a finite prediction horizon. State and input constraints, such as maximum angles, rates, and deflections, are defined. The problem is solved using CVXPY, a convex optimization modeling tool. The resulting optimal control sequence is extracted and printed. This approach ensures control inputs are computed considering system dynamics and constraints, facilitating stable and efficient control strategies for complex systems.

Forth file

Controlimplementation.py

The code formulates and solves a Quadratic Program (QP) for Model Predictive Control (MPC). It constructs QP matrices representing the cost function and constraint matrices based on system dynamics and constraints. Sparse matrices enhance computational efficiency. The QP is solved using the **osqp** solver. Optimal control inputs are extracted from the solution for the given prediction horizon and system dimensions. This approach enables real-time control of dynamical systems, ensuring adherence to constraints while optimizing performance. It's applicable across various domains, facilitating efficient and robust control strategies for complex systems, such as aerospace or industrial processes.

Fifth file

systemsimulations.py

The code simulates a dynamical system with given constraints for extreme initial states. It defines system dynamics and constraint parameters. The **check_constraints** function verifies if states satisfy constraints. The **simulate_system** function iteratively computes system states given initial conditions and control inputs, checking constraint adherence at each step. Extreme initial states from a defined set are simulated with zero control inputs. If constraints are violated during simulation, it prints the timestep where the violation occurs. This process facilitates analyzing system behavior under extreme conditions, crucial for robust system design and control strategy evaluation, ensuring safe operation within specified limits.