

Lab 17 Optimization II (Constrained)

This lab primarily focused on how to convert convex optimization problems from words to code.

1. Solving a Matrix Equation

There are many ways to solve the classic matrix equation $Ax = b$. In this problem, we explore how to solve it via optimizing a QP, specifically, we want to minimize $\|Ax - b\|_2$

However, the above expression is not differentiable anywhere. A common trick is to instead optimize the square of the norm, which results in the same solution since the norm is non-negative:

$$\text{minimize } \|Ax - b\|_2^2 = (Ax - b)^T (Ax - b)$$

Expanding that out to the canonical QP form we get

$$\begin{aligned} x^T A^T A x - x^T A^T b - b^T A x + b^T b &= x^T (A^T A) x + (-2b^T A) x + (b^T b) \\ D &= A^T A \\ c^T &= -2b^T A \\ c_0 &= b^T b \end{aligned}$$

Recall the QP canonical form. Note that the A matrix is often overloaded since it's the default name given to matrices and you should be aware of the context of its use. We do not actually have any constraints, so the A matrix of the canonical form is unfilled. Similarly, the b vector of the constraint in the canonical form is also unfilled. The variables to optimize, x , shares the same name as above, but this may not always be true.

$$\begin{aligned} \text{(QP) minimize } & \mathbf{x}^T D \mathbf{x} + \mathbf{c}^T \mathbf{x} + c_0 \\ \text{subject to } & A \mathbf{x} \preceq \mathbf{b}, \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned}$$

Let us set up the problem, first diving into the QP constructor:

```
Program qp(CGAL::SMALLER, false, 0, false, 0);
```

Since this is an unconstrained problem, we need to specify that there is no default finite bounds (the two `false` arguments). The other arguments do not matter since we are not specifying any constraints. Fill in the rest of the code below.

Note: Consulting the QP form above, for CGAL, you need to specify 2D instead of just D, and you need to specify only the upper triangular parts of D since it is symmetric. Concretely, if you have a matrix

$$D = \begin{bmatrix} D_{00} & D_{01} \\ D_{10} & D_{11} \end{bmatrix}$$

CGAL expects you to tell it $2D_{00}, 2D_{01}, 2D_{11}$

```
#include <math.h>
#include <iostream>
#include <eigen3/Eigen/Eigen>

#include <CGAL/QP_functions.h>
#include <CGAL/QP_models.h>

#include <CGAL/MP_Float.h>
typedef CGAL::MP_Float ET;

// program and solution types
typedef CGAL::Quadratic_program<double> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

int main() {
    // using QP
    Program qp(CGAL::SMALLER, false, 0, false, 0);

    Eigen::Matrix<double, 3,3> A;
    A << 1, 0.5, -0.5,
        0, 1, 2.1,
        -1, 0.2, -0.7;
    Eigen::Vector3d b;
    b << 0.7, 0.4, -0.6;

    // setup the QP
    // --- Your code here
    // ---

    Solution s = CGAL::solve_quadratic_program(qp, ET{});

    double optimal_value = CGAL::to_double(s.objective_value());

    auto it = s.variable_values_begin();
    Eigen::Vector3d x;
    x << CGAL::to_double(*it), CGAL::to_double(*(++it)), CGAL::to_double(*(++it));

    std::cout << s;
    std::cout << "optimal value: " << optimal_value << std::endl;
    std::cout << x << std::endl;
}
```

2. Constrained QP (Distance between polyhedra - 4.4.1 from Boyd)

The (Euclidean) distance between the polyhedra $\mathcal{P}_1 = \{x \mid A_1x \preceq b_1\}$ and $\mathcal{P}_2 = \{x \mid A_2x \preceq b_2\}$ in \mathbf{R}^n is defined as

$$\mathbf{dist}(\mathcal{P}_1, \mathcal{P}_2) = \inf\{\|x_1 - x_2\|_2 \mid x_1 \in \mathcal{P}_1, x_2 \in \mathcal{P}_2\}.$$

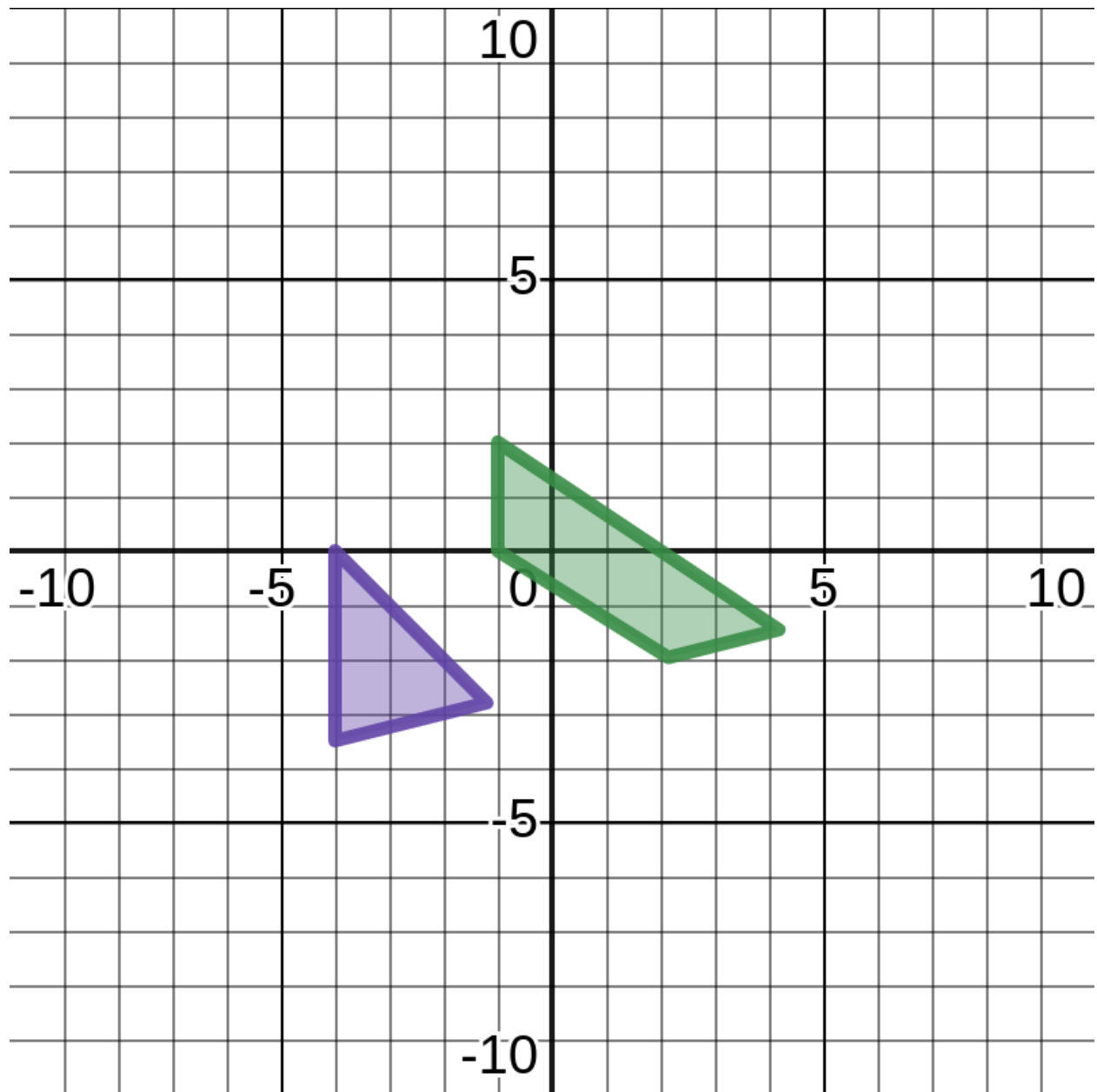
If the polyhedra intersect, the distance is zero.

To find the distance between \mathcal{P}_1 and \mathcal{P}_2 , we can solve the QP

$$\begin{array}{ll} \text{minimize} & \|x_1 - x_2\|_2^2 \\ \text{subject to} & A_1x_1 \preceq b_1, \quad A_2x_2 \preceq b_2, \end{array}$$

Suppose we are in 2D and have the polyhedra specified by:

<https://www.desmos.com/calculator/d5xcksn1mz>



$x + y \leq -4$	×
$-0.5x \leq 2$	×
$0.1x - 0.4y \leq 1$	×
$\text{polygon}((-4,0),(-4,-3.5),(-1.2,-2.8))$	×
$-x - 1.6y \leq 1$	×
$-x \leq 1$	×
$x + 1.5y \leq 2$	×
$0.1x - 0.4y \leq 1$	×
$\text{polygon}((-1,0),(2.143,-1.964),(4.182,-1.45)$	×

Where the polygons correspond to the equations above each `polygon` .

What are we optimizing over (what are the variables)?

Recall the canonical form expected from CGAL, what are the D, c^T, c_0, A, b parameters?

$$\begin{aligned}
 \text{(QP)} \text{ minimize } & \mathbf{x}^T D \mathbf{x} + \mathbf{c}^T \mathbf{x} + c_0 \\
 \text{subject to } & A \mathbf{x} \preceq \mathbf{b}, \\
 & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}
 \end{aligned}$$

Fill in the code below:

```

#include <math.h>
#include <iostream>
#include <eigen3/Eigen/Eigen>

#include <CGAL/QP_functions.h>
#include <CGAL/QP_models.h>

#include <CGAL/MP_Float.h>
typedef CGAL::MP_Float ET;

// program and solution types

```

```

typedef CGAL::Quadratic_program<double> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

int main() {
    Program qp(CGAL::SMALLER, false, 0, false, 0);

    // setup the QP
    // remember to specify 2D instead of just D
    // it also only expects the upper triangular part of D to be specified (since it's symmetric)
    // --- Your code here
    // ---

    Solution s = CGAL::solve_quadratic_program(qp, ET{});

    double optimal_value = CGAL::to_double(s.objective_value());

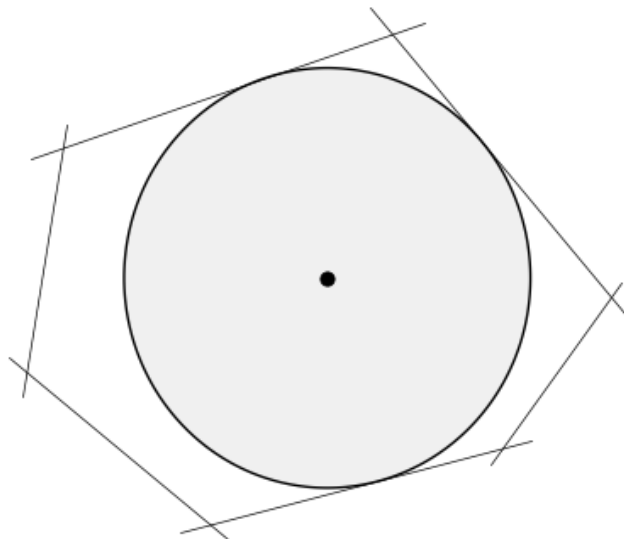
    auto it = s.variable_values_begin();
    Eigen::Matrix<double, n, 1> x;
    x << CGAL::to_double(*it), CGAL::to_double(*(++it)), CGAL::to_double(*(++it)), CGAL::to_double(*(++it));

    std::cout << s;
    std::cout << "optimal value: " << optimal_value << std::endl;
    std::cout << x << std::endl;
}

```

3. (Optional) Constrained LP (Chebyshev center of a polyhedron - 4.3.1 from Boyd)

Chebyshev center (fig 8.5)



We consider the problem of finding the largest Euclidean ball that lies in a polyhedron described by linear inequalities,

$$\mathcal{P} = \{x \in \mathbf{R}^n \mid a_i^T x \leq b_i, \ i = 1, \dots, m\}.$$

(The center of the optimal ball is called the *Chebyshev center* of the polyhedron; it is the point deepest inside the polyhedron, *i.e.*, farthest from the boundary; see §8.5.1.) We represent the ball as

$$\mathcal{B} = \{x_c + u \mid \|u\|_2 \leq r\}.$$

The variables in the problem are the center $x_c \in \mathbf{R}^n$ and the radius r ; we wish to maximize r subject to the constraint $\mathcal{B} \subseteq \mathcal{P}$.

We start by considering the simpler constraint that \mathcal{B} lies in one halfspace $a_i^T x \leq b_i$, *i.e.*,

$$\|u\|_2 \leq r \implies a_i^T(x_c + u) \leq b_i. \quad (4.30)$$

we can write (4.30) as

$$a_i^T x_c + r\|a_i\|_2 \leq b_i, \quad (4.31)$$

which is a linear inequality in x_c and r . In other words, the constraint that the ball lies in the halfspace determined by the inequality $a_i^T x \leq b_i$ can be written as a linear inequality.

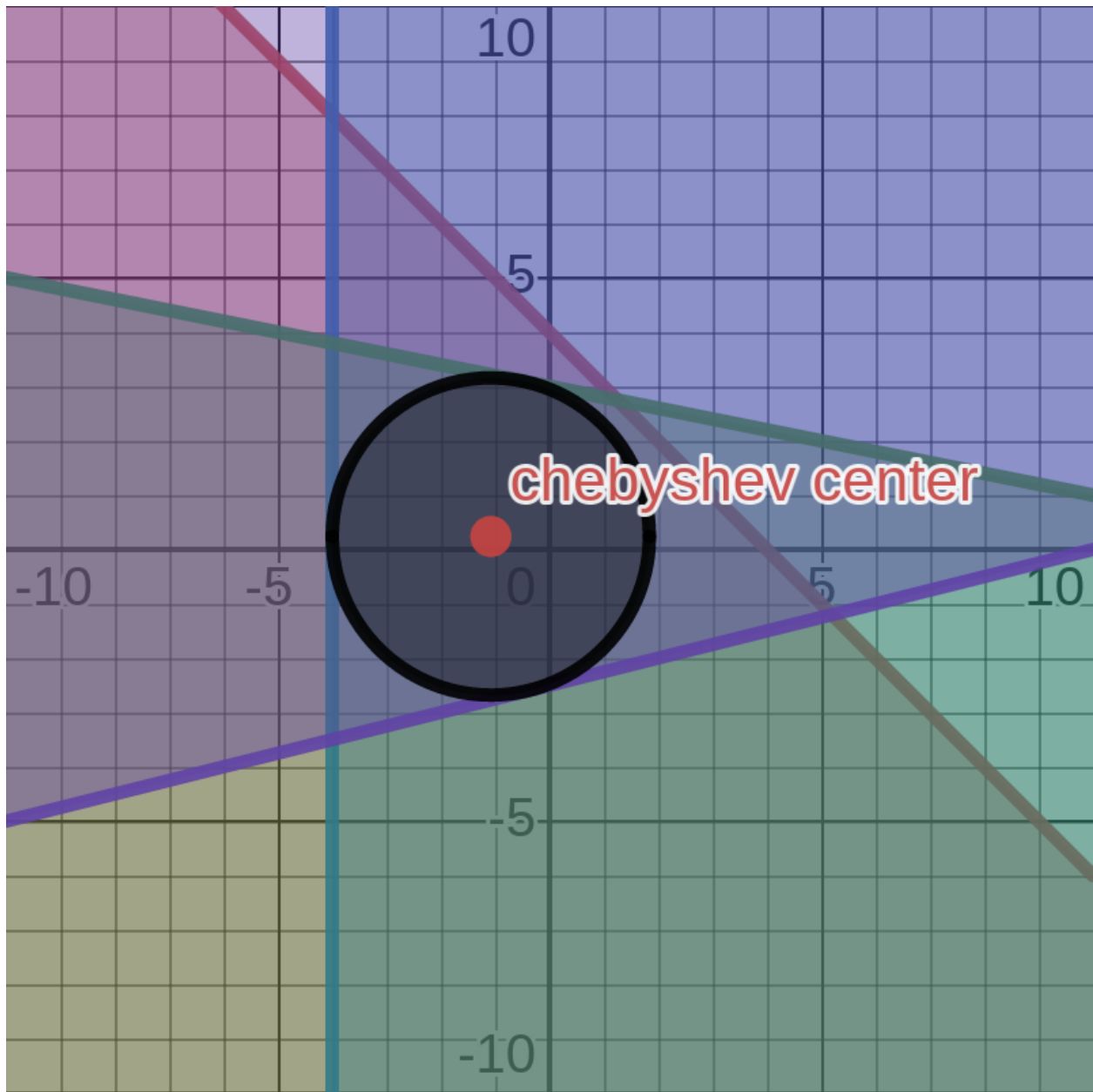
Therefore $\mathcal{B} \subseteq \mathcal{P}$ if and only if (4.31) holds for all $i = 1, \dots, m$. Hence the Chebyshev center can be determined by solving the LP





$$\begin{array}{ll} \text{maximize} & r \\ \text{subject to} & a_i^T x_c + r\|a_i\|_2 \leq b_i, \quad i = 1, \dots, m, \end{array}$$

What are we optimizing over (what are the variables)?

To make this problem concrete, suppose we are in 2D and have the polyhedron specified by:

<https://www.desmos.com/calculator/psd9n4i4ld>



1		$x + y \leq 4$
2		$-0.5x \leq 2$
3		$0.1x + 0.5y \leq 1.5$
4		$0.1x - 0.4y \leq 1$

Recall the canonical form expected from CGAL, what are the D, c^T, c_0, A, b parameters?

$$\begin{aligned}
 \text{(QP)} \text{ minimize } & \mathbf{x}^T D \mathbf{x} + \mathbf{c}^T \mathbf{x} + c_0 \\
 \text{subject to } & A \mathbf{x} \preceq \mathbf{b}, \\
 & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}
 \end{aligned}$$

Fill in the code below:

```

#include <math.h>
#include <iostream>
#include <eigen3/Eigen/Eigen>

#include <CGAL/QP_functions.h>
#include <CGAL/QP_models.h>

#include <CGAL/MP_Float.h>
typedef CGAL::MP_Float ET;

// program and solution types
typedef CGAL::Quadratic_program<double> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

int main() {
    Program lp(CGAL::SMALLER, false, 0, false, 0);

    // setup the LP
    // --- Your code here
    // ---

    Solution s = CGAL::solve_linear_program(lp, ET{});

    double optimal_value = CGAL::to_double(s.objective_value());

    auto it = s.variable_values_begin();
    Eigen::Vector3d x;
    x << CGAL::to_double(*it), CGAL::to_double(*(++it)), CGAL::to_double(*(++it));

    std::cout << s;
    std::cout << "optimal value: " << optimal_value << std::endl;
    std::cout << x << std::endl;
}

```