ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Algorithms Lab HS22
Department of Computer Science
Prof. Dr. A. Steger, Prof. Dr. E. Welzl
cadmo.ethz.ch/education/lectures/HS22/algolab

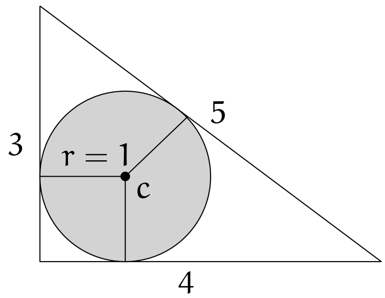
Solution — Inball

1 The problem in a nutshell

Given a cave, described by a set of linear inequalities in d dimensional space, find the maximum integral radius of a d dimensional ball that fits in the cave. A cave is described as:

$$C = \{x \in \mathbb{R}^d \colon a_i^T x \leqslant b_i \, , i = 1, \ldots, n\}.$$

$$-\mathbf{x} \leqslant 0$$
$$-\mathbf{y} \leqslant 0$$
$$3\mathbf{x} + 4\mathbf{y} \leqslant 12$$



The input consists of a set of linear inequalities, as on the left side of the figure above. On the right side we see a graphical description of the corresponding triangular cave, with the lengths of the sides. The largest ball that fits in that cave has radius $r = 1$ and the center of the ball is at $c = (1, 1)$.

2 Modeling

This problem has a geometric flavor and it comes with a collection of linear inequalities. Hence the obvious question: Can we solve the problem using linear programming?

In order to model a problem as a linear program (LP), we should first identify the roles of the various players in the game. In a linear program, we need to define variables and constraints. In the problem description there are two players: the cave, which is given, and the ball, which is sought. Usually, the variables in an LP correspond to the *unknowns* in the problem, whereas the constraints model the *known* aspects (in relation to the unknowns).

Therefore, a natural approach is to model the sought ball using variables, and to model the given cave using constraints. A natural representation of a ball is by center (a point in $\mathbb{R}^d$) and radius (a real number). So let us denote the sought ball by B, with center c and radius r.

In order to see how to model the cave as (hopefully linear) constraints, let us establish what it means for B to fit into the cave. Consider one of the given inequalities $a^T x \leqslant b$ and the halfspace $H = \{x \mid a^T x \leqslant b\}$ it defines. Let h be the hyperplane that defines H, that is, $h = \{x \mid a^T x = b\}$. The normal vector to h is a and the normal unit vector is $\frac{a}{||a||_2}$, where $||a||_2 = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$ is the Euclidean norm. When is B contained in H? Consider the 2-dimensional illustration in Figure 1.

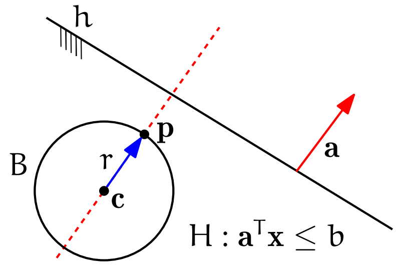


Figure 1: The line h defines a halfplane H with normal vector a (in red). The ball B has center c and radius r. The dashed red line is parallel to a and passes through c.

Let p denote the point of B that is furthest from H in direction of the normal vector a. Then

$$p = c + r\frac{a}{\|a\|_2}.$$

We have

$$B \subset H \iff p \in H \iff a^T p \leqslant b \iff a^T(c + r\frac{a}{\|a\|_2}) \leqslant b,$$

which can be simplified to

$$a^T c + \|a\|_2 r \leqslant b. \tag{1}$$

Indeed, this is a linear inequality: $a = (a_1, \dots, a_d) \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are given constants, and so $\|a\|_2 \in \mathbb{R}$ is a given constant. Setting $c = (c_1, \dots, c_d) \in \mathbb{R}^d$, we observe that

$$a^T c + \|a\|_2 r = \sum_{i=1}^{d} a_i c_i + \|a\|_2 r$$

is a linear function in the variables $c_1, \dots, c_d$ and r of our LP. The number of variables is $d+1$, which is sufficiently small because $d \leqslant 10$.

A ball fits into the given cave if it satisfies every such inequality, for all halfspaces that make up the cave. At the same time we want to maximize the radius of the ball, which yields the following LP formulation:

$$\begin{array}{ll} \text{maximize} & r \\ \text{subject to} & a_i^T x + \|a_i\| r \leqslant b_i, \quad \text{for } i = 1, \dots, n. \end{array}$$

There is three possible outcomes:

(i) The input constraints define a bounded polytope and thus there is an optimal value to be reported.

(ii) The input constraints define an unbounded polytope (that is, the cave is open). In this case an arbitrarily large ball fits into the cave.

(iii) The input constraints define an empty set.

The LP solver can tell us which of the three outcomes we have.

For this problem there is not really much choice in terms of algorithm design. Also there is only one level (that is, type of test sets) in the problem description. After all, this is just an introductory problem to practice modeling linear programs and working with the CGAL LP solver.

3 Implementation

Here are a few remarks about the implementation.

- By the problem description we have $|(a_i)_j|, |b_i| \leqslant 2^{10}$, for every i, j. In addition, it is guaranteed that $\|a_i\|_2$ is an integer, for every i. Thus, we have

$$\|a_i\|_2^2 = \sum_{j=1}^{d} (a_i)_j^2 \leqslant d \cdot (2^{10})^2 = 10 \cdot 2^{20} < 2^{24}.$$

  It follows that all the coefficients of our LP inequalities can be described by the type `int`, which we assume to be 32bit on Code Expert. Actually, note that only $\|a_i\|_2$ will be a coefficient (which, of course, is even smaller in bitsize). Therefore, we use `int` as an input type and `CGAL::Gmpz` as an exact type for the solution.

- The constraint inequalities for the LP have the $\leqslant$ relationship. In addition, remember that the variables are the position of the center of the ball and the size of the radius. We do not want to impose any lower or upper bounds for the position of the center. Hence, we call the LP constructor as `Program lp(CGAL::SMALLER, false, 0, false, 0)`.

- Recall that the CGAL LP solver always minimizes. So in order to maximize r, we minimize $-r$. In the code below, we make r the $(d+1)$-st variable, that is, it has index d. So we set its coefficient in the objective function to be -1 with `lp.set_c(d, -1)`. In addition, we add a lower bound of 0 for r because any solution with $r < 0$ is irrelevant.

- We want to round down the resulting radius to the nearest integer. Let `s` be the variable that holds the solution that the solver returns. The function `s.objective_value()` returns an object of type `CGAL::Quotient<ET>` where `ET` is the exact type. Note that this quotient is not necessarily simplified (that is, numerator and denominator may have common factors). In order to round it down, we just have to divide the numerator by the denominator using integer division. We can directly output the exact type because it is integral. Please refer to Line 50/51 of the complete solution on the next page.

4 A Complete Solution

```cpp
#include <iostream>
#include <stdexcept>
#include <cmath>
#include <CGAL/QP_models.h>
#include <CGAL/QP_functions.h>
#include <CGAL/Gmpz.h>

typedef int IT; // input type
typedef CGAL::Gmpz ET; // exact Type
typedef CGAL::Quadratic_program<IT> Program;
typedef CGAL::Quadratic_program_solution<ET> Solution;

int main()
{
  std::ios_base::sync_with_stdio(false);

  int n;
  for (std::cin >> n; n > 0; std::cin >> n) {
    int d;
    std::cin >> d;
    Program lp(CGAL::SMALLER, false, 0, false, 0);
    for (int i = 0; i < n; ++i) {
      int norm2 = 0;
      for (int j = 0; j < d; ++j) {
        IT ai;
        std::cin >> ai;
        norm2 += ai * ai;
        lp.set_a(j, i, ai);
      }

      // not needed, just to be sure: check that the norm is an integer
      int norm = std::floor(std::sqrt(norm2));
      if (norm2 != norm * norm)
        throw std::runtime_error("Error: norm2!= norm*norm.\n");

      lp.set_a(d, i, norm);
      IT bi;
      std::cin >> bi;
      lp.set_b(i, bi);
    }
    lp.set_c(d, -1);
    lp.set_l(d, true, 0);

    Solution s = CGAL::solve_linear_program(lp, ET());
    if (s.is_infeasible())
      std::cout << "none\n";
    else if (s.is_unbounded())
      std::cout << "inf\n";
    else
      std::cout << -(s.objective_value().numerator() /
                     s.objective_value().denominator())
                << "\n";
  }
}
```