Using the RCDD Package

Charles J. Geyer

April 22, 2019

1 The Name of the Game

We call the package `rcdd` which stands for "C Double Description in R," our name being copied from `cddlib`, the library we call to do the computations. This library was written by Komei Fukuda and is available at

`http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html`

Our `rcdd` package for R provides an interface to most of the functionality of the `cddlib` library.

The version of R used to make this document is 3.5.3. The version of the `rcdd` package used to make this document is 1.2.2.

2 Representations

The two descriptions in question are the descriptions of a convex polyhedron as either

- the intersection of a finite collection of closed half spaces or
- the convex hull of of a finite collection of points and directions.

A *direction* in $\mathbb{R}^d$ can be identified with either a nonzero point x or with the ray $\{\lambda x : \lambda \ge 0\}$ generated by such a point. The *convex hull* of a set of points x_1, ..., x_k and a set of directions represented as nonzero points x_{k+1}, ..., x_m is the set of linear combinations

$$x = \sum_{i=1}^{m} \lambda_i x_i$$

where the coefficients λ_i satisfy

$$\lambda_i \ge 0, \qquad i = 1, \ldots, m$$

and

$$\sum_{i=1}^{k} \lambda_i = 1$$

(note that only the λ_i for points, not directions, are in the latter sum). The fact that these two descriptions characterize the same class of convex sets (the *polyhedral* convex sets) is Theorem 19.1 in Rockafellar (1970). The points and directions are said to be *generators* of the convex polyhedron. Those who like eponyms call this the Minkowski-Weyl theorem

`http://www.ifor.math.ethz.ch/staff/fukuda/polyfaq/node14.html`

2.1 The H-representation

In the terminology of the `cddlib` documentation, the two descriptions are called the "H-representation" and the "V-representation" ("H" for half space and "V" for vertex, although, strictly speaking, generators are not always vertices).

For both efficiency and computational stability, the H-representation allows not only closed half spaces but hyperplanes (which are, of course, the intersection of two closed half spaces), or, what is equivalent, the H-representation characterizes the convex polyhedron as the solution set of a finite set of linear equalities and inequalities, that is, the set of points x satisfying

$$A_1 x \le b_1 \quad \text{and} \quad A_2 x = b_2$$

where A_1 and A_2 are matrices and b_1 and b_2 are vectors and the dimensions are such that these equations make sense.

In the representation used for our `rcdd` package for R, these parts of the specification are combined into one big matrix

$$M = \begin{pmatrix} 0 & b_1 & -A_1 \\ 1 & b_2 & -A_2 \end{pmatrix}$$

If the dimension of the space in which the polyhedron lives is d, then M has column dimension $d+2$ and the first two columns are special. The first column is an indicator vector, zero indicates an inequality constraint and one an equality constraint. The second column contains the "right hand side" vectors b_1 and b_2. Although we have given an example in which all the inequality rows are on top of all the equality rows, this is not required. The rows can be in any order.

If `m` is such a matrix and we let

```
l <- m[ , 1]
b <- m[ , 2]
v <- m[ , - c(1, 2)]
a <- (- v)
```

In mathematical notation

$$M = \begin{pmatrix} l & b & -A \end{pmatrix}$$

where l and b are column vectors and A is a matrix. Then the convex polyhedron described is the set of points `x` that satisfy

```
axb <- a %*% x - b
all(axb <= 0)
all(l * axb == 0)
```

In mathematical notation, if

$$w = Ax - b$$

then

$$\begin{aligned} w_i &\leq 0, \qquad \text{for all } i \\ w_i &= 0, \qquad \text{whenever } l_i = 1 \end{aligned}$$

2.2 The V-representation

For both efficiency and computational stability, the V-representation allows not only points and directions, but also lines and something I don't know the name of (perhaps "affine generators").

In R a V-representation is matrix with the same column dimension as the corresponding H-representation, and again the first two columns are special, but their interpretation is different. Now the first two columns are both indicators (zero or one valued). The rest of each row represents a point.

The convex polyhedron described is the set of linear combinations of these points such that the coefficients are (1) nonnegative if column one is zero and (2) sum to one where the sum runs over rows having a one in column two.

If `m` is such an object and we define `v`, `b`, and `l` as in the preceding section (`l` is column one, `b` is column two, and `v` is the rest), in mathematical notation

$$M = \begin{pmatrix} l & b & V \end{pmatrix}$$

where l and b are column vectors and V is a matrix, then the polyhedron in question is the set of points of the form

```
y <- t(lambda) %*% v
```

where `lambda` satisfies the constraints

```
all(lambda * (1 - l) >= 0)
sum(b * lambda) == max(b)
```

In mathematical notation, the polyhedron is the set of points of the form

$$y = \lambda^T V$$

where

$$\lambda_j \geq 0, \qquad \text{when } l_j = 0$$

and

$$\sum_j b_j \lambda_j = 1, \qquad \text{unless all } b_j \text{ are zero.}$$

2.3 Fukuda's Representations

Readers interested in comparing with Fukuda's documentation should be aware that `cddlib` uses different but mathematically equivalent representations. If our representation is a matrix `m`, then Fukuda's representation consists of a matrix, which is our `m[ , -1]` and a vector (which he calls the *linearity*), which is our `seq(1, nrow(m))[m[ , 1] == 1]` (that is the vector of indices of the rows having a one in our column one).

3 Trying it Out

3.1 A Unit Simplex

Let's try a really simple example, so we can see what's going on: the unit simplex in $\mathbb{R}^3$ (essentially copied from the `scdd` help page, never mind how `makeH` works, just look at the matrix `qux` that it produces, which is an H-representation).

```
> library(rcdd)
> d <- 3
> # unit simplex in H-representation
> qux <- makeH(- diag(d), rep(0, d), rep(1, d), 1)
> print(qux)

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1   -1   -1   -1
[2,]    0    0    1    0    0
[3,]    0    0    0    1    0
[4,]    0    0    0    0    1
attr(,"representation")
[1] "H"
```

The first row represents the equality constraint `sum(x) == 1` and the other three rows represent the inequality constraints `x[i] >= 0` for `i` in `1:d`.

```
> # unit simplex in V-representation
> out <- scdd(qux)
> print(out)

$output
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    0    0    1
[2,]    0    1    0    1    0
[3,]    0    1    1    0    0
attr(,"representation")
[1] "V"
```

The corresponding V-representation has 3 vertices, $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$.

```
> # unit simplex in H-representation
> # note: different from original, but equivalent
> out <- scdd(out$output)
> print(out)

$output
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1   -1   -1    0
[2,]    0    0    1    0    0
[3,]    0    0    0    1    0
[4,]    1   -1    1    1    1
attr(,"representation")
[1] "H"
```

Note that `scdd` goes both ways. If we toggle back, we get a different H-representation, but one that still represents the unit simplex.

3.2 Adding a Constraint

Now let us complicate the situation a bit. The unit simplex represents possible probability vectors. Let the corresponding sample space be `x <- 1:d`. So the expected value of the random variable X having probability vector `p` is `sum(p * x)`. Let us say we want to look at the subset of the simplex consisting of the probability vectors `p` and that satisfy the equality constraint `sum(p * x) == 2.2`.

```
> # add equality constraint
> quux <- addHeq(1:d, 2.2, qux)
> print(quux)

     [,1] [,2] [,3] [,4] [,5]
[1,]    1  1.0   -1   -1   -1
[2,]    0  0.0    1    0    0
[3,]    0  0.0    0    1    0
[4,]    0  0.0    0    0    1
[5,]    1  2.2   -1   -2   -3
attr(,"representation")
[1] "H"

> out <- scdd(quux)
> print(out)

$output
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1  0.4  0.0  0.6
[2,]    0    1  0.0  0.8  0.2
attr(,"representation")
[1] "V"
```

Adding the equality constraint takes us down a dimension. The unit simplex was two-dimensional (a triangle). Now the represented convex polyhedron is one-dimensional (a line segment).

4 Using GMP Rational Arithmetic

4.1 A Simple Example

The `cddlib` code can also use the GMP (GNU Multiple Precision) Library to compute results using exact arithmetic with unlimited precision rational numbers and we bring this facility to `rcdd` as well.

In order to use rational arithmetic, we need a rational number format. Adding a new numeric type to R would be a job of horrendous complexity, so we don't even try (this has actually been done in the `gmp` package but that package was written long after the `rcdd` package). We just use the representation of the rational as a character string, e. g., `"3/4"` or `"-15/32"` (perhaps some day there will be a version of `rcdd` that uses objects of type `bigq` from the `gmp` package, but the current version cannot).

```
> quuxq <- d2q(quux)
> print(quuxq)

     [,1] [,2]                                    [,3] [,4] [,5]
[1,] "1"  "1"                                     "-1" "-1" "-1"
[2,] "0"  "0"                                     "1"  "0"  "0"
[3,] "0"  "0"                                     "0"  "1"  "0"
[4,] "0"  "0"                                     "0"  "0"  "1"
[5,] "1"  "2476979795053773/1125899906842624" "-1" "-2" "-3"
attr(,"representation")
[1] "H"
```

What is that? Well computers count in binary and 2.2 is *not* a round number to computers (because 1/10 is not a finite sum of powers of 2). We can see that the rational representation does make sense

```
> bar <- as.numeric(unlist(strsplit(quuxq[5,2], "/")))
> print(bar)

[1] 2.47698e+15 1.12590e+15

> bar[1] / bar[2]

[1] 2.2
```

But we don't want to check our rational approximations that way because (1) it's a pain and (2) big integers needn't be exactly represented either. So if you're willing to take `rcdd`'s word for it

```
> q2d(quuxq)

     [,1] [,2] [,3] [,4] [,5]
[1,]    1  1.0   -1   -1   -1
[2,]    0  0.0    1    0    0
[3,]    0  0.0    0    1    0
[4,]    0  0.0    0    0    1
[5,]    1  2.2   -1   -2   -3
attr(,"representation")
[1] "H"
```

But that was just a preliminary explanation. The point is that `scdd` uses rational representations like `quuxq` just as well as (actually better than) inexact floating point representations like `quux`.

```
> outq <- scdd(quuxq)
> print(outq)

$output
     [,1] [,2] [,3]
[1,] "0"  "1"  "900719925474099/2251799813685248"
[2,] "0"  "1"  "0"
     [,4]                                [,5]
[1,] "0"                                 "1351079888211149/2251799813685248"
[2,] "900719925474099/1125899906842624" "225179981368525/1125899906842624"
attr(,"representation")
[1] "V"
```

Oops! Excuse the verbose mess.

```
> print(q2d(outq$output))

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1  0.4  0.0  0.6
[2,]    0    1  0.0  0.8  0.2
attr(,"representation")
[1] "V"
```

But that too, was not exactly what I wanted to present. It's not rational arithmetic that is really messy here, but floating point! Let's make the rational approximation to be exactly what we wanted.

```
> quuxq <- z2q(round(quux * 10), rep(10, length(quux)))
> print(quuxq)

     [,1] [,2]   [,3] [,4] [,5]
[1,] "1"  "1"    "-1" "-1" "-1"
[2,] "0"  "0"    "1"  "0"  "0"
```

```
[3,] "0"  "0"    "0"  "1"  "0"
[4,] "0"  "0"    "0"  "0"  "1"
[5,] "1"  "11/5" "-1" "-2" "-3"
attr(,"representation")
[1] "H"
```

```
> outq <- scdd(quuxq)
> print(outq)

$output
     [,1] [,2] [,3]  [,4]  [,5]
[1,] "0"  "1"  "2/5" "0"   "3/5"
[2,] "0"  "1"  "0"   "4/5" "1/5"
attr(,"representation")
[1] "V"
```

Now we have a nice exact representation. It's the floating point stuff that is wrong.

```
> qmq(outq$output, out$output)

     [,1] [,2] [,3]                     [,4]
[1,] "0"  "0"  "13/90071992547409920" "0"
[2,] "0"  "0"  "0"                    "13/45035996273704960"
     [,5]
[1,] "-1/11258999068426240"
[2,] "-1/5629499534213120"
attr(,"representation")
[1] "V"
```

4.2 Warning

The discussion in the preceding section presents rational arithmetic as a way to get nicer answers, but its main purpose is to get correct answers. In version 1.1-4 of the package the following warning was added to help pages for functions that do computational geometry (including `scdd`).

> If you want correct answers, use rational arithmetic. If you do not, this function may (1) produce approximately correct answers, (2) fail with an error, (3) give answers that are nowhere near correct with no error or warning, or (4) crash R losing all work done to that point. In large simulations (1) is most frequent, (2) occurs roughly one time in a thousand, (3) occurs roughly one time in ten thousand, and (4) has only occured once and only with the `redundant` function. So the R floating point arithmetic version does mostly work, but you cannot trust its results unless you can check them independently.

Using the computer's default floating point (inexact) arithmetic is more convenient, and usually — but not always — works. In this the `rcdd` package is no different from any other R package. The only difference is that some aspects of the objects (convex polyhedra) are discrete (how many generators), so a small error in arithmetic may cause a discrete error (integer sized) in the result. But this is no different from many other calculations in statistics. For example, in linear regression we need to deal with collinearity, and whether a model matrix is not full rank is an all-or-nothing proposition. The `lm` function uses QR decomposition with the default computer arithmetic to detect collinearity. This can produce incorrect results. Same issue as with `rcdd`.

To repeat, if you want correct (provably correct) answers, and you don't want to deal with cases (2), (3), and (4) in the warning, you must use rational arithmetic, despite its inconvenience. And it is merely inconvenient. You can use it, whatever you want to do.

4.3 Convex Hull

Let's try to find convex hulls in `d` dimensions.

```
> d <- 4
> n <- 100
> set.seed(42)
> x <- matrix(rnorm(d * n), nrow = n)
> foo <- makeV(d2q(x))
> out <- scdd(foo)
> l <- out$output[ , 1]
> b <- out$output[ , 2]
> v <- out$output[ , - c(1, 2)]
> a <- qneg(v)
```

This code generates a matrix `x`, each row of which represents a point in $\mathbb{R}^d$. The H-representation of the convex hull of these points is given by the column vectors l and b and the matrix A. Actually, the vector l is unnecessary, because since the convex hull is bounded we know $l_j = 0$ for all j. Note that we use exact arithmetic.

```
> axb <- qmatmult(a, t(x))
> axb <- sweep(axb, 1, b, FUN = qmq)
> fred <- apply(axb, 2, function(foo) max(qsign(foo)))
> all(fred <= 0)

[1] TRUE

> sum(fred < 0)

[1] 60

> sum(fred == 0)
```

```
[1] 40
```

Here `qmatmult(a, t(x))` is the matrix product ax^T, and `b` is the matrix resulting from subtracting b from each column of ax^T. Then `fred` is the vector that gives for each point -1, 0, or $+1$ if it is in the interior, boundary, or exterior of the convex hull, respectively.

The points on the boundary of the convex hull are the rows of `x[fred == 0, ]`. If one only wants to find the set of points on the boundary, then Section 6.2 discusses a more direct way to do this. If one only wants to check whether one point in the interior or exterior of the convex hull, then Section 5.4 discusses a more direct way to do this.

If we want to check whether other points are in the hull, this is easy

```
> y <- matrix(rnorm(2 * n * d), nrow = 2 * n)
> ayb <- qmatmult(a, t(d2q(y)))
> ayb <- sweep(ayb, 1, b, FUN = qmq)
> sally <- apply(ayb, 2, function(foo) max(qsign(foo)))
> sum(sally < 0)

[1] 95

> sum(sally == 0)

[1] 0

> sum(sally > 0)

[1] 105
```

There are 95 points (rows of y) in the interior of the hull, 0 points on the boundary, and 105 points in the exterior.

5 Linear Programming

Version 0.8 of the `rcdd` package adds linear programming. One might think this is not particularly interesting, because there are already two other R contributed packages that do linear programming, but `rcdd` can solve linear programs using exact rational arithmetic and the others cannot.

Here are some simple examples taken from the help page for the lpcdd function.

5.1 A Problem Having a Solution

```
> hrep <- rbind(c("0", "0", "1", "1", "0", "0"),
+               c("0", "0", "0", "2", "0", "0"),
+               c("1", "3", "0", "-1", "0", "0"),
+               c("1", "9/2", "0", "0", "-1", "-1"))
> print(hrep)
```

```
     [,1] [,2]  [,3] [,4] [,5] [,6]
[1,] "0"  "0"   "1"  "1"  "0"  "0"
[2,] "0"  "0"   "0"  "2"  "0"  "0"
[3,] "1"  "3"   "0"  "-1" "0"  "0"
[4,] "1"  "9/2" "0"  "0"  "-1" "-1"
```

```
> a <- c("2", "3/5", "0", "0")
> out <- lpcdd(hrep, a)
> print(out)
```

```
$solution.type
[1] "Optimal"

$primal.solution
[1] "-3"  "3"   "9/2" "0"

$dual.solution
[1] "-2"   "0"    "-7/5" "0"

$optimal.value
[1] "-21/5"
```

The function `lpcdd` minimizes the linear function $x \mapsto a^T x$ subject to the abstract constraint that x lie in the polyhedral convex set having H-representation given by `hrep`.

In this problem, the linear program (LP) has a solution, which is given by the `out$primal.solution`. We can check that this does indeed give the stated optimal value

```
> qsum(qxq(a, out$primal.solution))
```

```
[1] "-21/5"
```

Moreover, we can check the Kuhn-Tucker conditions for optimality, one statement of which follows. For the problem

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \end{aligned}$$

where f is scalar-valued and g is vector-valued (so g represents a set of inequality constraints). If there are equality constraints (as in this problem), they can be represented as two inequality constraints.

Define the Lagrangian function

$$L(x, u) = f(x) + u^T g(x)$$

where u is a vector of Lagrange multipliers. Specialized to our problem where $f(x) = a^T x$ and $g(x) = Ax - b$, the Lagrangian is

$$L(x, u) = a^T x + u^T A x - u^T b$$

The a pair $(\bar{x}, \bar{u})$ is optimal if

(i) $\bar{x}$ minimizes $x \mapsto L(x, \bar{u})$,

(ii) $g(\bar{x}) \le 0$,

(iii) $\bar{u} \ge 0$,

(iv) $\bar{u}^T g(\bar{x}) = 0$.

Condition (ii) is called *primal feasibility*, condition (iii) *dual feasibility*, and condition (iv) *complementary slackness*.

Note that for an equality constraint, the corresponding Lagrange multiplier does not need to be nonnegative, because if the constraint is $g_i(x) = 0$, we could instead take it to be $-g_i(x) = 0$.

We claim that `- out$dual.solution` gives the Lagrange multipliers u. Then dual feasibility is clear. Let's check primal feasibility

```
> xbar <- out$primal.solution
> foo <- qmatmult(hrep[ , - c(1, 2)], cbind(xbar))
> foo <- qpq(hrep[ , 2], foo)
> print(foo)

[1] "0" "6" "0" "0"
```

We are supposed to check that the components of $A\bar{x} - b$ are ≤ 0 for the inequality constraints and $= 0$ for the equality constraints. Here `foo` is $-A\bar{x} + b$ so should check ≥ 0 for the inequality constraints. We do indeed have all components of `foo` nonnegative and the last two (which are for the equality constraints), zero.

Now complementary slackness is also clear. For each row, either the corresponding component of `foo` should be zero, or the corresponding component of `out$dual.solution`.

```
> qxq(foo, out$dual.solution)

[1] "0" "0" "0" "0"
```

Finally, we need to check that $\bar{x}$ minimizes the Lagrangian function, condition (i). Since the objective function is affine, and the constraints are linear, this can only happen if the Lagrangian is a constant function of x, in which case its derivative is zero

```
> qpq(a, qmatmult(rbind(out$dual.solution), hrep[ , -c(1, 2)]))

[1] "0" "0" "0" "0"
```

The derivative of the Lagrangian is

$$\nabla L(x, \bar{u}) = a^T + u^T A$$

This is what is calculated above (recall that `out$dual.solution` is $-u$ and `hrep[ , -c(1, 2)]` is $-A$).

5.2 A Problem with Empty Feasible Region

```
> hrep <- rbind(c("0", "0", "1", "0"),
+               c("0", "0", "0", "1"),
+               c("0", "-2", "-1", "-1"))
> print(hrep)
```

```
     [,1] [,2] [,3] [,4]
[1,] "0"  "0"  "1"  "0"
[2,] "0"  "0"  "0"  "1"
[3,] "0"  "-2" "-1" "-1"
```

```
> a <- c("1", "1")
> out <- lpcdd(hrep, a)
> print(out)
```

```
$solution.type
[1] "Inconsistent"

$dual.direction
[1] "1" "1" "1"
```

The dual direction $(1, 1, 1)$ indicates that the sum of the three inequalities

$$\begin{aligned} -x_1 - 0 &\le 0 \\ -x_2 - 0 &\le 0 \\ x_1 + x_2 - (-2) &\le 0 \end{aligned}$$

is $2 \le 0$, which is false (hence no point can satisfy the inequalities and the feasible region is empty).

5.3 A Problem with Unbounded Objective Function

```
> hrep <- rbind(c("0", "0", "1", "0"),
+               c("0", "0", "0", "1"))
> print(hrep)
```

```
     [,1] [,2] [,3] [,4]
[1,] "0"  "0"  "1"  "0"
[2,] "0"  "0"  "0"  "1"
```

```
> a <- c("1", "1")
> out <- lpcdd(hrep, a, minimize = FALSE)
> print(out)
```

```
$solution.type
[1] "DualInconsistent"

$primal.direction
[1] "1" "0"
```

Here the problem is to maximize $f(x) = x_1 + x_2$ subject to the constraints

$$x_1 \geq 0$$
$$x_2 \geq 0$$

The vector `out$primal.direction` is a direction of recession of the feasible region C, that is, if we call this direction y, then

$$x + sy \in C, \qquad x \in C,\ s \geq 0.$$

To verify that, we need to check that

$$A(x + sy) \leq b, \qquad x \in C,\ s \geq 0,$$

which, if we assume C is nonempty, is equivalent to

$$Ay \leq 0.$$

```
> qmatmult(hrep[ , - c(1, 2)], cbind(out$primal.direction))
```

```
     [,1]
[1,] "1"
[2,] "0"
```

It checks (recall that `href[ , - c(1, 2)]` is $-A$).

We also need to verify that the objective function increases without bound in this direction

```
> qsum(qxq(a, out$primal.direction))
```

```
[1] "1"
```

It does.

5.4 Convex Hull Revisited

If one wants to verify whether a single point q is in or out of the convex hull of many other points p_i, then the "Polyhedral Computation FAQ"

`http://www.ifor.math.ethz.ch/~fukuda/polyfaq/polyfaq.html`

says there is a more efficient way to do it than the than the calculations in Section 4.3.

We can prove a point q in in the exterior of the convex hull of a set of points $\{ p_i : i \in I \}$ by finding a strongly separating hyperplane, which is determined by a vector z and a scalar z_0 satisfying

$$z^T p_i < z_0, \qquad i \in I$$
$$z^T q > z_0$$

We can do this by solving the following LP

$$\begin{aligned} &\text{minimize } f(z_0, z) = z^T q - z_0 \\ &\text{subject to } z^T p_i - z_0 \le 0, \qquad i \in I \\ &\qquad\qquad z^T q - z_0 \le 1 \end{aligned}$$

(the last inequality being inserted to make the LP have a bounded solution). Note that the variable in the LP is the vector (z_0, z) which has dimension one more than q and p_i.

If the optimal value is strictly positive, then we have a strongly separating hyperplane and q is in the exterior of the convex hull. Otherwise q is on the boundary or in the interior.

Let's try it. First a point in the interior.

```
> xin <- x[fred < 0, , drop = FALSE]
> qin <- xin[sample(nrow(xin), 1), ]
> qin

[1]  0.5809965 -0.1507760  1.3099782 -0.6905602

> hrep <- cbind(0, 0, 1, - x)
> hrep <- rbind(hrep, c(0, 1, 1, - qin))
> out <- lpcdd(d2q(hrep), d2q(c(-1, qin)), minimize = FALSE)
> out$optimal.value

[1] "0"
```

So q is not in the exterior.

Now a point in the exterior.

```
> yout <- y[sally > 0, , drop = FALSE]
> qout <- yout[sample(nrow(yout), 1), ]
> qout

[1]  2.0948051 -0.1392716  0.4199467 -0.4526308

> hrep <- cbind(0, 0, 1, - x)
> hrep <- rbind(hrep, c(0, 1, 1, - qout))
> out <- lpcdd(d2q(hrep), d2q(c(-1, qout)), minimize = FALSE)
> out$optimal.value

[1] "1"
```

So q is in the exterior.

6 Redundant Row Elimination

In the section title "row" refers to a row of the matrix that specifies an H-representation or a V-representation. For an H-representation a row represents a linear equality or inequality constraint, so this is redundant constraint elimination. For a V-representation a row represents a generator (point, ray, line, or affine generator), so this is redundant generator elimination.

6.1 Redundant Constraints

Here is a toy problem from the on-line help for the `redundant` function.

```
> hrep <- rbind(c(0, 0,  1,  1,  0),
+               c(0, 0, -1,  0,  0),
+               c(0, 0,  0, -1,  0),
+               c(0, 0,  0,  0, -1),
+               c(0, 0, -1, -1, -1))
> print(hrep)

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    0
[2,]    0    0   -1    0    0
[3,]    0    0    0   -1    0
[4,]    0    0    0    0   -1
[5,]    0    0   -1   -1   -1

> redundant(hrep, representation = "H")

$output
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    1    1    0
[2,]    1    0   -1    0    0
[3,]    0    0   -1   -1   -1
attr(,"representation")
[1] "H"

$implied.linearity
[1] 1 2 3

$redundant
[1] 4

$new.position
[1] 1 2 0 0 3
```

The `output` component of the result gives another representation having no redundant rows of the convex polytope of the same type (H or V) as the input.

In this example, the constraints are

$$\begin{aligned} -x_1 - x_2 &\le 0 \\ x_1 &\le 0 \\ x_2 &\le 0 \\ x_3 &\le 0 \\ x_1 + x_2 + x_3 &\le 0 \end{aligned}$$

The first three of these imply equality constraints $x_1 = x_2 = 0$. This is indicated by the `implied linearity` component of the result. These three inequality constraints are replaced by two equality constraints

$$\begin{aligned} -x_1 - x_2 - 0 &= 0 \\ x_1 &= 0 \end{aligned}$$

which also are equivalent to $x_1 = x_2 = 0$. That these are now equality constraints is indicated by the 1 in the first column of the `output` matrix. The forth row of the input implies $x_3 \le 0$, and we now see that the fifth row of the input is redundant, since $x_1 = x_2 = 0$, the fifth row implies $x_3 \le 0$, which is already implied by the forth row.

The `redundant` argument of the result is what is returned by the `cddlib` library function (`dd_MatrixCanonicalize`) that does the work for the R function `redundant`. This function has decided to take the fourth row rather than the fifth as redundant. It does not seem to count rows involved in the "implied linearity" as redundant here, nor from other examples does it seem to count any equality constraints as redundant even as it drops them.

However, the `new.position` component of the result shows which rows of the input are kept in the output and which not. So we can always tell which rows of the input were actually dropped from this component.

6.2 Convex Hull Revisited Again

Eliminating redundant generators from a set of points gives the points that are the *vertices* or *extreme points* of the hull.

```
> foo <- makeV(points = d2q(x))
> out <- redundant(foo)
> nrow(out$output)

[1] 40

> all((out$new.position == 0) == (fred < 0))

[1] TRUE
```

7 Faces

A nonempty *face* of a convex polyhedron P (Rockafellar, 1970, Chapter 18) is the subset of P where some affine function achieves its maximum over P. Note that P itself is a face (the set where constant functions achieve their maxima). By definition the empty set is also a face. The empty set and P are *improper* faces of P. All other faces are proper. Proper faces of the highest dimension (here 2) are called *facets*. Proper faces of dimension zero (single points) are called *vertices*. Proper faces of dimension one (line segments) are called *edges*.

Given an H-representation for a convex polyhedron, the function `allfaces` produces a list of faces. Here is a toy problem from the on-line help for the `allfaces` function.

```
> vrep <- rbind(c(0, 1,  1,  1, 0),
+               c(0, 1,  1, -1, 0),
+               c(0, 1, -1,  1, 0),
+               c(0, 1, -1, -1, 0),
+               c(0, 1,  0,  0, 1))
> print(vrep)

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    1    0
[2,]    0    1    1   -1    0
[3,]    0    1   -1    1    0
[4,]    0    1   -1   -1    0
[5,]    0    1    0    0    1

> hrep <- scdd(vrep, rep = "V")$output
> print(hrep)

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    0   -1
[2,]    0    1    0    1   -1
[3,]    0    1   -1    0   -1
[4,]    0    1    0   -1   -1
[5,]    0    0    0    0    1
attr(,"representation")
[1] "H"
```

Here the convex polytope in question is a pyramid with a square base. The base is in the x-y plane, the square centered at the origin, with sides parallel to the x and y axes, and vertices of the form $(\pm 1, \pm 1, 0)$. The fifth vertex (the *apex*) is above the base on the z axis $(0, 0, 1)$.

After conversion, we see that is convex polytope is also characterized by five inequalities, two of the form $z \pm x \leq 1$, two of the form $z \pm y \leq 1$, and $z \geq 0$. If this representation is nonredundant (which is obviously is — we could check with the `redundant` function, but won't), then there will be five faces of

dimension 2 (the base and four sides of the pyramid), eight faces of dimension 1 (the four sides of the base, and the four edges that connect vertices of the base with the apex), and five faces of dimension zero. Plus there is one face of dimension 3 (the pyramid itself) and one face of dimension -1 (by convention), the empty set.

```
> out <- allfaces(hrep)
> d <- unlist(out$dimension)
> nd <- tabulate(d + 1)
> names(nd) <- seq(0, 3)
> print(nd)
```

```
0 1 2 3
5 8 5 1
```

The empty set is omitted from the list of faces produced by `allfaces` (it is always a face but you don't need a computer to tell you that).

```
> asl <- sapply(out$active.set, paste, collapse = " ")
> names(asl) <- d
> asl <- asl[order(d)]
> print(asl)
```

```
      0       0       0       0       0       1       1       1
"3 4 5" "2 3 5" "1 4 5" "1 2 5" "1 2 3 4"   "4 5"   "3 5"   "3 4"
      1       1       1       1       1       2       2       2
  "2 5"   "2 3"   "1 5"   "1 4"   "1 2"     "5"     "4"     "3"
      2       2       3
    "2"     "1"      ""
```

The component `active.set` of the result of `allfaces` gives the row numbers of the active set of constraints for a face (the set of inequalities that are satisfied with equality on the face). The active set characterizes the face. Here we see that there are indeed five facets, all of which have one constraint active. Also there are five vertices, four of which have three constraints active (the vertices of the base) and one having four constraints active (the apex). There are eight edges, all of which have two constraints active.

References

Rockafellar, R. T. (1970). *Convex Analysis*. Princeton University Press.