Design of the RCDD Package

Charles J. Geyer

April 18, 2005

1 The Name of the Game

We have called 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.cs.mcgill.ca/~fukuda/soft/cdd_home/
```

Our `rcdd` package for R makes available some (by no means all) of the functionality of the `cddlib` library.

The "C" is for either of the C or C++ computer languages. This is bad terminology, making a mere implementation detail part of the name, but we keep Fukuda's name. And we make the terminology worse by tacking an "R" on the front for another irrelevant implementation detail (but at least we're consistent).

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.

For those readers who are not familiar with the second description, we give more detail. We assume the notion of a point in $\mathbb{R}^d$ is familiar. A *direction* in $\mathbb{R}^d$ can be identified with either a nonzero point x or with the ray $\{\lambda x : \lambda \geq 0\}$ generated by such a point. The *convex hull* of a set of points $x_1, \ldots, x_k$ and a set of directions represented as nonzero points $x_{k+1}, \ldots, x_m$ is the set of linear combinations

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

where the coefficients λ_i satisfy

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

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 (*Convex Analysis*, Princeton University Press, 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 Representations

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

then the convex polyhedron described is the set of points `x` that satisfy

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

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 `a`, `b`, and `l` as in the preceding section (`l` is column one, `b` is column two, and `a` is the rest), then the polyhedron in question is the set of points of the form

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

where `lambda` satisfies the constraints

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

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 Converting Between Representations

The R function `scdd` converts H-representations to V-representations and vice versa. The result is a list that always contains a component `output` which is the computed representation and may contain a component `input` which is the input representation (depending on an argument `keepinput`, about which see below).

Other options involve auxiliary computations, any of the arguments

```
adjacency = TRUE
incidence = TRUE
inputadjacency = TRUE
inputincidence = TRUE
```

(the defaults are `FALSE`) produce additional results, which are components of the list returned by `scdd` having the same name as the argument (`adjacency` and so forth). Each is a ragged array: `adjacency[[i]][j]` (note the brackets) says that the `i`-th and `j`-th rows of the `output` are "adjacent", and so forth. See

```
http://www.cs.mcgill.ca/~fukuda/soft/cddman/node4.html
```

for more about these.

The result contains a component `input` if

```
keepinput = "TRUE"
```

or if

```
keepinput = "maybe"
```

and the input is involved in an adjacency or incidence list (the default is `"maybe"`).

The last option involves the computation itself. The `roworder` option specifies the order in which the rows of M are processed which can have a considerable effect on the running time of the algorithm and, when using normal floating point arithmetic (see Section 4 below), on the numerical results of the algorithm or even on success or failure of the algorithm. This argument is a finite choice

```
rowoder = c("lexmin", "lexmax", "minindex", "maxindex",
    "mincutoff", "maxcutoff", "mixcutoff", "randomrow")
```

and `match.arg` is used for the argument matching, so (1) the argument may be abbreviated and (2) the default is `"lexmin"` if no argument is specified.

```
http://www.cs.mcgill.ca/~fukuda/soft/cddman/node4.html
http://www.cs.mcgill.ca/~fukuda/soft/cddlibman/node6.html
```

contain some discussion of which to use. The main bit of advice seems to be that `roworder = "maxcut"` might be useful when an input H-representation contains many redundant inequalities or an input V-representation contains many interior points.

4 Using GMP Rational Arithmetic

The `cddlib` code can also use the GMP (GNU Multiple Precision) Library to compute results using exact arithmetic with unlimited precision rational numbers.

In order to do this, the input problem must be in this form. Thus we need a way to specify rational numbers. We specify them as character objects of the following form: an optional minus sign followed by an integer (the *numerator*), followed by a slash, followed by another integer (the *denominator*). If the denominator is one, both it and the slash may be omitted. The string contains no whitespace.

All of

```
1/3
-5/7
2
1234567890123455678901234567890/33
```

are valid. Note that the last is not exactly representable as an ordinary floating point number (for that matter neither are $1/3$ and $-5/7$). The point of the long example is to point out that integers of any size are allowed. The numerator and denominator do not have to be representable as ordinary computer integers.

We have two functions `d2q` and `q2d` that convert from standard floating point (`storage.mode "double"` in R) to rational and vice versa. One can also construct rationals from numerators and denominators using the `z2q` function.