Using the RCDD Package

Charles J. Geyer

January 4, 2008

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.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.

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 $\{\lambda x : \lambda \geq 0\}$ 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

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

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`

2.1 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 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 that the matrix `qux` that it produces, which is an H-representation).

```
> library(rcdd)
> d <- 3
> 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`.

```
> 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, $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$.

```
> 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 us say the points in the state space are `x <- 1:d` and we the elements of the unit simplex are probability vectors `p` and we want to add the equality constraint `sum(p * x) == 2.2`.

```
> 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).

3.3 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 <- cbind(0, cbind(1, x))
> outh <- scdd(foo, inputincidence = TRUE, representation = "V")
> inies <- sapply(outh$inputincidence, length) == 0
> sum(inies)
```

```
[1] 60
```

The points on the surface of the convex hull are the rows of `x[! inies, ]`. (The code in `cddlib` provides a faster way to do this, but `rcdd` does not currently provide an interface to it.)

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. We just use the representation of the rational as a character string, e. g., `"3/4"` or `"-15/32"`.

```
> 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 power 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 (better actually) 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 A More Complicated Example

Let's check our convex hull calculation.

```
> outhq <- scdd(d2q(foo), inputincidence = TRUE, representation = "V")
> iniesq <- sapply(outhq$inputincidence, length) == 0
> all.equal(inies, iniesq)

[1] TRUE

> nrow(outh$output) == nrow(outhq$output)

[1] TRUE

> outh <- scdd(foo, incidence = TRUE, representation = "V")
> outhq <- scdd(d2q(foo), incidence = TRUE, representation = "V")
> all.equal(outh$incidence, outhq$incidence)

[1] TRUE
```

So we happened to get the same number of points on the surface of the hull. And we got the same number of facets of the convex polyhedron, and they all had the same shape. The inexactness of floating point arithmetic didn't hurt us this time. But if you put this in a loop doing simulations, eventually you will get wrong answers with floating point. Or worse, `scdd` will just give up on the problem being unable to decide whether a point is inside or on the surface of the hull (or whatever). An example of this is in `rcdd/tests/oops.R*` where the problem in `oops.RData` does make `scdd` fail unless rational arithmetic is used. This problem arose in a simulation done by Glen Meeden.