The Hit and Run Algorithm

Mike Flynn

June 25, 2013

To sample from the hull of a convex polytope, a random-walk algorithm is typically used. The "hit-and-run" type algorithm uses the following steps:

1. Start from an intial solution x_0
2. Pick a random direction in the polytope, u
3. Isolate the segment s connecting x_0 to a wall of the polytope in that direction u
4. Sample uniformly from the segment s

In Detail:

The polytope is defined as the set: $x|Ax = Ax_0, x0$ where x is a vector of length n, A is a mn matrix of constraints and x_0 an original solution. This set is geometrically an intersection of m n-planes defined by the rows of A and the half spaces x_i0. An easy case to picture is the 1-row A: $[1, 1, 1]$ with initial solution $(.3, .3, .4)$. This corresponds the the plane $x + y + z = 1$. The intersection of this plane with $x > 0$ leaves only the part in the first octant, a triangle.

Because we must at another solution to $Ax = Ax_0$ in the end, picking a "random" direction will not be a random direction in the space of x but rather a random direction in the k-plane that is $Ax = Ax_0$. This is done by finding an orthogonal basis of the null space of A: $Z_1, Z_2, \ldots, Z_k$, which will necessarily be orthogonal vectors in the k-plane. These basis vectors are weighted uniformly by sampling their weights from an exponential distribution and dividing by their sum. Therefore:

$$u = \sum_{j=0}^{k} Z_j r_j$$

where r_j is the normalized random weight.

We sample along the segment s by saying that $x_{i+1} = x_i + t * u$ where t is some scalar parameter, bounded by the limits of s. To sample uniformly on s, we merely must sample uniformly on t, bounded by the limits of s. To find the limits of t we must simply recognize that for each index i:

$$x_i + t * u_i 0$$

, of which there are only 2 important cases: when $u_i 0$ and when $u_i < 0$. This is because when we solve for the limits of t we simply divide by u_i to get 2 equations:

$$t_i - \frac{x_i}{u_i} for u_i > 0$$
$$and$$
$$t_i - \frac{x_i}{u_i} for u_i < 0$$

Therefore the largest t can be is large enough so that it is still less than the smallest such right hand side for the second equation, set by x_i, and likewise, must be greater than the largest right hand side for the first equation. Formally:

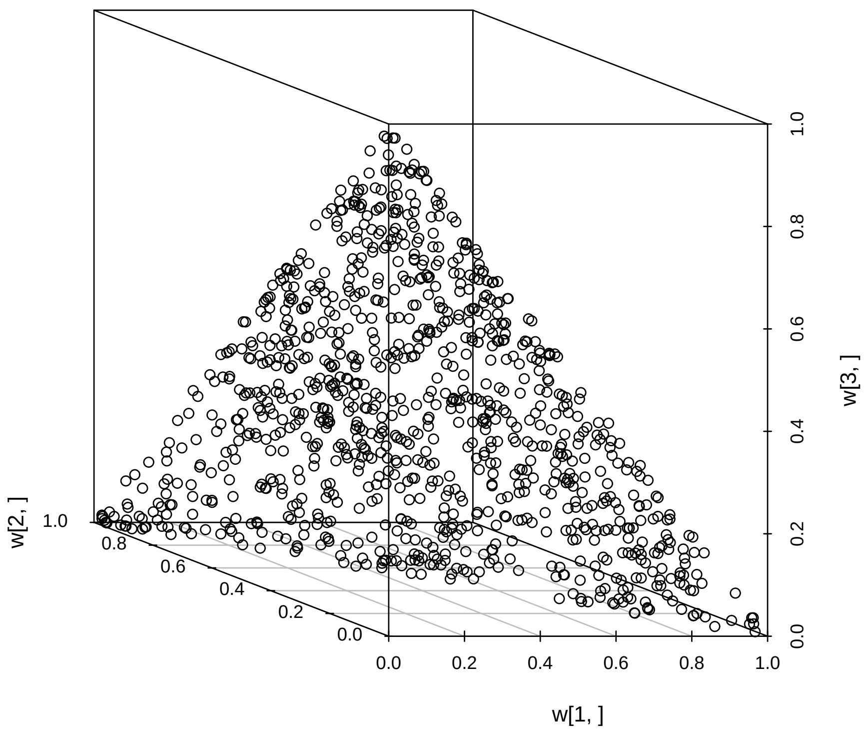
$$t_{max} = Min(-\frac{x_i}{u_i}) for u_i < 0$$
$$and$$
$$t_{min} = Max(-\frac{x_i}{u_i}) for u_i > 0$$

After these are figured out, t can be drawn from a uniform distribution between t_{min} and t_{max} to walk randomly on the simplex.

A demonstration:

```
require(MASS)
require(scatterplot3d)
#' Uniformly samples from {A*x=A*x0} U {x>0}
getWeights.hnr <- function(A, x0, n, discard) {

    y = x0
    ## resolve weird quirk in Null() function
    if (ncol(A) == 1) {
        Z = Null(A)
    } else {
        Z = Null(t(A))
    }
    X = matrix(0, nrow = length(x0), ncol = n + discard)
    for (i in 1:(n + discard)) {
        ## u is a random unit vector
        u = rexp(ncol(Z))
        u = u/sum(u)

        ## d is a unit vector in the appropriate k-plane pointing in a random
        ## direction
        d = Z %*% u
```

```
        c = y/d
        ## determine intersections of x + t*d with edges
        tmin = max(-c[d > 0])
        tmax = min(-c[d < 0])

        ## writeLines(paste('tmin: ', tmin, '\ntmax: ', tmax, '\n', sep = ''))
        ## chose a point on the line segment
        y = y + (tmin + (tmax - tmin) * runif(1)) * Z %*% u
        X[, i] = y
    }
    return(X[, (discard + 1):ncol(X)])
}


## Sample from the triangle {x+y+z =1} U {x>0}
Amat = matrix(c(1, 1, 1), ncol = 3, nrow = 1)
x0 = c(0.3, 0.2, 0.5)
w = getWeights.hnr(Amat, x0, 1000, 5)

scatterplot3d(x = w[1, ], y = w[2, ], z = w[3, ], angle = 160)
```



w[1,]
w[2,]
w[3,]
0.0
0.2
0.4
0.6
0.8
1.0