Supplementary Material for the paper "Asymptotics for Constrained Dirichlet Distributions"

By
Charles J. Geyer and Glen D. Meeden

Technical Report No. 691

School of Statistics
University of Minnesota

March 20, 2012

Abstract

This document is supplementary material for a paper. It shows how to simulate the linear-equality-and-inequality-constrained normal distribution that is the large sample approximation to a similarly constrained Dirichlet posterior.

1 Introduction

Throughout this document "the paper" refers to Geyer and Meeden (submitted). We are interested in simulating a constrained Dirichlet distribution that is a posterior for multinomial data using conjugate priors. According to the discussion in Section 4.2 of the paper, if y is the vector of multinomial data, then the unconstrained posterior is Dirichlet with hyperparameter $\hat{\alpha}_n = y + \xi$, where ξ is the hyperparameter of the prior (which is also Dirichlet). Here we use $\xi = 0$, which gives an improper prior but a proper posterior so long as no component of y is zero. The constrained Dirichlet posterior restricts the Dirichlet probability density function to a constraint set determined by a finite set of linear equality and inequality constraints and renormalizes so it integrates to one over the constraint set.

According to Remark 6 in the paper, we approximate the posterior with a normal distribution having unnormalized probability density function

$$h(\lambda) = \exp\left(-\frac{n}{2}(\lambda - \hat{\lambda}_n)^T \widehat{\Lambda}_n^{-1}(\lambda - \hat{\lambda}_n)\right) \tag{1}$$

(this is equation (22) in the paper), where the scalar n is the multinomial sample size, the vector $\hat{\lambda}_n$ is the unconstrained posterior mean $\hat{\alpha}_n/n$, and the matrix $\widehat{\Lambda}_n$ is diagonal with diagonal entries that are the corresponding components of the vector $\hat{\lambda}_n$. The constrained asymptotic approximation of the posterior restricts (1) to the constraint set and renormalizes so it integrates to one over the constraint set.

We impose the constraints in two steps: equality constraints first, and then inequality constraints. The equality constraints constrain the parameter vector λ to an affine subspace.

Suppose the equality constraints have the form $B\lambda = a$ where B is a known matrix and a is a known vector, let V denote the vector subspace

$$V = \{\, w \in \mathbb{R}^d : Bw = 0 \,\}$$

(this is equation (19) in the paper), and let M be a matrix whose columns are a basis for V. Then every λ in the equality constraint set has the form

$$\lambda = \lambda_0 + M\beta \tag{2}$$

for some β, where λ_0 is any point in the equality constraint set, that is, any vector satisfying $B\lambda_0 = a$. We can think of β as a new parameter for the problem that, like λ has an approximate normal distribution, and, moreover, unlike λ, has a nondegenerate normal approximation. The mean and variance of this normal approximation are given in Remark 6 of the paper as

$$\beta_n^* = \left(M^T \widehat{\Lambda}_n^{-1} M\right)^{-1} M^T \widehat{\Lambda}_n^{-1}(\hat{\lambda}_n - \lambda_0) \tag{3}$$

(this is equation (25) in the paper) and

$$n^{-1}(M^T \widehat{\Lambda}_n^{-1} M)^{-1} \tag{4}$$

We simulate λ having the normal approximation to the equality constrained Dirichlet posterior by simulating β multivariate normal with mean vector (3) and variance matrix (4) and then transforming to the original parameter λ via (2).

We then impose the inequality constraints. In the simulation we reject any λ that do not satisfy the constraints.

2 R Package RCDD

We use the R statistical computing environment (R Development Core Team, 2012) in our analysis. It is free software and can be obtained from `http://cran.r-project.org`. Precompiled binaries are available for Windows, Macintosh, and popular Linux distributions. We use the contributed package `rcdd` (Geyer and Meeden, 2009). If R has been installed, but this package has not yet been installed, do

```
install.packages("rcdd")
```

from the R command line (or do the equivalent using the GUI menus if on Apple Macintosh or Microsoft Windows). This may require root or administrator privileges.

Assuming the `rcdd` package has been installed, we load it

```
> suppressPackageStartupMessages(library(rcdd))
```

The version of the package used to make this document is 1.1-7. The version of R used to make this document is 2.15.0.

This entire document and all of the calculations shown were made using the R command `Sweave` and hence are exactly reproducible by anyone who has R and the R noweb (RNW) file from which it was created. Both the RNW file and and the PDF document produced from it will be made available at the University of Minnesota Digital Conservancy.

Not only can one exactly reproduce the results in the printable document, one can also modify the parameters of the simulation and get different results. Anything at all can be changed once one has the RNW file.

In particular, we set the "seed" of the random number generator

```
> set.seed(42)
```

so that every time this RNW file is run it produces the same results. Changing the argument of `set.seed` or removing this chunk of R code will produce different results.

```
> d <- 9
> n <- 1000
> if (d < 7) stop("need dimension at least 7")
```

These statements choose the dimension of the parameter space (9) and the multinomial sample size (1000). Modifying either or both of these statements will change the simulation accordingly.

3 The Constraints

We set up the constraints as an RCDD H-representation. First, the constraints defining the unit simplex.

```
> hrep <- makeH(- diag(d), rep(0, d), rep(1, d), 1)
```

These are the constraints that the components of the parameter are nonnegative and sum to one.

Second, a constraint on the mean of a certain random variable. Let X be a random variable taking values $1, \ldots, d$ with probabilities given by the vector λ.

```
> x <- d2q(1:d)
> mu <- qdq(qsum(x), d2q(d))
> hrep <- addHeq(x, mu, hrep)
```

This equality constraint requires that the mean of X be $\mu = (d + 1)/2 = 5$.

Third, constraints on the median of the same random variable.

```
> dev.from.mean <- qmq(x, rep(mu, d))
> dev.from.mean.plus.two <- qpq(dev.from.mean, rep("2", d))
> dev.from.mean.minus.two <- qmq(dev.from.mean, rep("2", d))
> hrep <- addHin(as.numeric(qsign(dev.from.mean.plus.two) < 0),
+     "1/2", hrep)
> hrep <- addHin(as.numeric(qsign(dev.from.mean.minus.two) > 0),
+     "1/2", hrep)
```

These inequality constraints require that the median of X be in the closed interval with endpoints $\mu - 2 = 3$ and $\mu + 2 = 7$.

We now want to add some constraints on the variance of X but do not know what is reasonable. Since any linear constraint is maximized or minimized at an extreme point of the constraint set, we check what are the extreme points of the current constraint set and what the variance of X is at each.

```
> vout <- scdd(hrep, incidence = TRUE)
> extremes <- vout$output[ , - c(1, 2)]
> squared.dev.from.mean <- qxq(dev.from.mean,
+     dev.from.mean)
> extreme.var <- qmatmult(extremes, cbind(squared.dev.from.mean))
> extreme.var <- as.vector(extreme.var)
> extreme.var[order(q2d(extreme.var))]

 [1] "0"    "1"    "2"    "2"    "3"    "3"    "4"    "4"    "4"
[10] "6"    "6"    "8"    "8"    "9"    "19/2" "19/2" "10"   "10"
[19] "21/2" "21/2" "11"   "11"   "23/2" "23/2" "16"

> more.extreme.var <- extreme.var[sapply(vout$incidence,
+     function(foo) nrow(hrep) %in% foo)]
> more.extreme.var[order(q2d(more.extreme.var))]

[1] "9"    "19/2" "10"   "21/2" "11"   "23/2" "16"
```

The points which are rows of the matrix `extremes` are the extreme points of the current constraint set (represented by `hrep`). The components of the vector `extreme.var` are the values of the variance of the random variable X at those extreme points. The components of the vector `more.extreme.var` are the values for the subset of those points at which the constraint determined by the last row of `hrep` is binding (holds with equality). This is the upper bound median constraint, so this makes the point $\mu + 2 = 7$ a median of the random variable X.

Now we find the upper and lower quartiles of those variance values.

```
> fred <- q2d(more.extreme.var)
> sally <- quantile(fred, type = 1)[c(2, 4)]
> varbound <- more.extreme.var[fred %in% sally]
> varbound <- unique(varbound)
> varbound <- varbound[order(q2d(varbound))]
> varbound
```

```
[1] "19/2" "23/2"
```

Finally, we constrain the variance of X to be between those bounds.

```
> hrep <- addHin(qneg(squared.dev.from.mean),
+     qneg(varbound[1]), hrep)
> hrep <- addHin(squared.dev.from.mean,
+     varbound[2], hrep)
```

This completes the construction of the constraint set. The R object `hrep` specifies it.

4 Faces

This is a fairly complicated convex polytope.

```
> fout <- allfaces(hrep)
> dims <- unlist(fout$dimension)
> length(dims)
```

```
[1] 1103
```

```
> f <- tabulate(dims + 1, nbins = d + 1)
> f
```

```
 [1]  46 178 308 305 185  67  13   1   0   0
```

There is one (improper) face of dimension $d - 2$, which is the whole constraint set. The dimension is $d - 2$ because there are two equality constraints: the components of λ sum to one and the mean of X is μ. There is also one (improper) face which is the empty set (and which is not counted above, since the function `allfaces` only lists nonempty faces). There are 46 vertices (faces consisting of a single point), 178 edges (faces which are line segments), 13 facets (proper faces of maximal dimension, in this case $d-3$), and 67 ridges (faces of dimension one less than the dimension of facets, in this case $d - 4$).

5 Simulation Truth

In order that our simulated data be a good test case, we want the inequality constraints to have effect. For this reason we choose the simulation truth parameter value (the λ we use to simulate multinomial data) to satisfy two of the four complicated inequality constraints with equality (where "complicated" means not the nonnegativity constraints, which are all satisfied with strict inequality). We choose the two upper bound constraints, which are the last row and the third to last row of `hrep`.

We make the simulation truth parameter value a relative interior point of the ridge satisfying these two upper bound constraints with equality as well as the equality constraints.

```
> ncons <- nrow(hrep)
> inies <- sapply(fout$active.set,
+     function(foo) all(c(ncons - 2, ncons) %in% foo))
> inies.max.dim <- max(unlist(fout$dimension[inies]))
> inies.of.max.dim <- inies &
+     unlist(fout$dimension) == inies.max.dim
> lambda.truth <- fout$relative.interior.point[inies.of.max.dim]
> length(lambda.truth) == 1

[1] TRUE

> lambda.truth <- lambda.truth[[1]]
> lambda.truth <- q2d(lambda.truth)
> lambda.truth

[1] 0.37946429 0.02008929 0.02008929 0.02008929 0.02008929 0.02008929
[7] 0.02008929 0.42187500 0.07812500
```

Clearly this λ satisfies the nonnegativity constraints. We check "by hand" using the most obvious code, that it satisfies the other constraints.

```
> x <- q2d(x)
> mu <- q2d(mu)
> sum(lambda.truth)

[1] 1

> sum(lambda.truth * x)

[1] 5

> sum(lambda.truth[x < mu - 2])

[1] 0.3995536

> sum(lambda.truth[x > mu + 2])

[1] 0.5

> sum(lambda.truth * (x - mu)^2)
```

```
[1] 11.5
```

```
> q2d(varbound)
```

```
[1]  9.5 11.5
```

We see that, indeed, all of the constraints are satisfied at this point, and two of the complicated inequality constraints are satisfied with equality, and, of course, this is all that are possible to satisfy with equality, because the median cannot simultaneously be at both its lower and upper bound and similarly for the variance.

6 Data

Make up data.

```
> y <- rmultinom(1, size = n, prob = lambda.truth)
> y <- as.vector(y)
> if (any(y <= 0)) stop("must have all y values strictly positive")
> y
```

```
[1] 366  22  21  23  16  22  23 426  81
```

7 Remark

None of the work done to this point in this document would need to be done for a real data analysis. The data vector `y` would be given (it would be the data). The equality and inequality constraints would be determined by prior knowledge (perhaps influenced by other data about the random variable X that is involved in the complicated constraints). A real data analysis would start here.

8 Affine Hull

We determine the affine hull of the constraint set.

```
> equalities <- hrep[ , 1] == "1"
> equalities
```

```
 [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[12] FALSE FALSE FALSE FALSE
```

```
> av <- scdd(hrep[equalities, ])$output
> av.point <- av[av[ , 1] == "0", ]
> av.lines <- av[av[ , 1] == "1", ]
> av.point <- as.vector(av.point[- c(1, 2)])
> av.lines <- av.lines[ , - c(1, 2)]
> av.point
```

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

```
> av.lines
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,] "1"  "-2" "1"  "0"  "0"  "0"  "0"  "0"  "0"
[2,] "2"  "-3" "0"  "1"  "0"  "0"  "0"  "0"  "0"
[3,] "3"  "-4" "0"  "0"  "1"  "0"  "0"  "0"  "0"
[4,] "4"  "-5" "0"  "0"  "0"  "1"  "0"  "0"  "0"
[5,] "5"  "-6" "0"  "0"  "0"  "0"  "1"  "0"  "0"
[6,] "6"  "-7" "0"  "0"  "0"  "0"  "0"  "1"  "0"
[7,] "7"  "-8" "0"  "0"  "0"  "0"  "0"  "0"  "1"
```

The vector `av.point` is a point in the affine hull of the constraint set. The rows of the matrix `av.lines` are a basis for the tangent space of the affine hull of the constraint set (the vector space parallel to it).

We now change notation to follow Remark 6 of the paper.

```
> m <- t(q2d(av.lines))
> lambda.zero <- q2d(av.point)
```

Our `m` is M in the paper, and our `lambda.zero` is λ_0 in the paper.

We check to see that these objects behave well with respect to the constraints (as they must if the work above is correct). First we check that λ_0 satisfies the equality constraints (it does not satisfy the inequality constraints, and does not need to).

```
> sum(lambda.zero)
```

```
[1] 1
```

```
> sum(x * lambda.zero)
```

```
[1] 5
```

Second we check that the columns of M satisfy the equality constraints with the right-hand side changed to zero (because movement along these vectors should go from λ_0 to other points satisfying the equality constraints).

```
> as.vector(rbind(rep(1, d)) %*% m)
```

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

```
> as.vector(rbind(x) %*% m)
```

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

9 Point Estimates

```
> alpha.hat <- y
> lambda.hat <- alpha.hat / n
```

In notation of Section 4.2 of the paper, we are using the improper prior determined by setting the hyperparameter ξ to the zero vector. Our `alpha.hat` is $\hat{\alpha}_n$ in the paper, and our `lambda.hat` is $\hat{\lambda}_n$ in the paper.

10 Beta

Now we find the vector (3) and the matrix (4) that are the mean and variance of the normal distribution for the new parameter β that lives on the affine hull.

First, we find β_n^* given by (3).

```
> lout <- lm(lambda.hat - lambda.zero ~ 0 + m,
+     weights = 1 / lambda.hat)
> beta.star <- lout$coefficients
```

We can also do this by literally implementing (3).

```
> beta.star.too <- solve(t(m) %*% diag(1 / lambda.hat) %*% m) %*%
+     t(m) %*% diag(1 / lambda.hat) %*% cbind(lambda.hat - lambda.zero)
> all.equal(as.vector(beta.star), as.vector(beta.star.too))
```

```
[1] TRUE
```

Second, we find the variance matrix given by (4).

```
> varmat <- solve(t(m) %*% diag(1 / lambda.hat) %*% m) / n
```

11 Simulate Equality Constrained Posterior

```
> library(MASS)
> nboot <- 1e5
> beta <- mvrnorm(nboot, beta.star, varmat)
> lambda <- beta %*% t(m)
> lambda <- sweep(lambda, 2, lambda.zero, "+")
```

This code simulates β vectors (each row of the matrix `beta` is one such simulation. Then we transform to λ using (2), except since `beta` is actually a matrix, must use sweep instead of add.

Each row of the matrix `lambda` is a simulated normal random vector having the required mean vector and variance matrix. We check that all of these simulations satisfy the equality constraints.

```
> range(lambda %*% cbind(x))
```

```
[1] 5 5
```

```
> range(apply(lambda, 1, sum))
```

```
[1] 1 1
```

12 Apply Inequality Constraints to Simulation

Now we apply the inequality constraints to the simulated realizations of the unconstrained posterior. To do this we need the constraints in the form $B\lambda \le a$.

```
> inequalities <- hrep[ , 1] == "0"
> bmat <- q2d(hrep[inequalities, ])
> avec <- as.vector(bmat[ , 2])
> bmat <- bmat[ , - c(1, 2)]
> bmat <- (- bmat)
```

This code makes matrix `bmat` and vector `avec`, which are B and a in mathematical notation, that determine the inequality constraints as described.

Since `lambda` is actually a matrix, we must sweep instead of add

```
> goodies <- lambda %*% t(bmat)
> goodies <- sweep(goodies, 2, avec)
> goodies <- apply(goodies < 0, 1, all)
> mean(goodies)

[1] 0.53917
```

Our Monte Carlo (MC) sample is the "goodies."

```
> lambda <- lambda[goodies, ]
```

Now we check — the dumb way to be safe — that our MC sample does indeed satisfy the constraints.

```
> dim(lambda)

[1] 53917     9

> range(apply(lambda, 1, min))

[1] 0.001703099 0.025639433

> range(apply(lambda, 1, sum))

[1] 1 1

> range(apply(lambda, 1, function(lambda) sum(lambda[x < mu - 2])))

[1] 0.3776154 0.4096745

> range(apply(lambda, 1, function(lambda) sum(lambda[x > mu + 2])))

[1] 0.4676482 0.4999998

> range(apply(lambda, 1, function(lambda)
+     sum(q2d(squared.dev.from.mean) * lambda)))

[1] 10.96291 11.50000

> q2d(varbound)

[1]  9.5 11.5
```

13 Remark

That ends the simulation. The rows of the matrix `lambda` are independent and identically distributed Monte Carlo simulations of the (approximate, large sample, asymptotic) constrained posterior distribution.

14 Importance Weights

Suppose we were to use the asymptotic approximation as an importance sampling distribution to sample the correct (finite n) distribution.

The Dirichlet posterior distribution has unnormalized probability density function

$$g(\lambda) = \prod_{i=1}^{n} \lambda_i^{\hat{\alpha}_i - 1} \tag{5}$$

and the asymptotic approximation has unnormalized probability density function (1). Their ratio is the unnormalized importance weights.

```
> log.g.lambda <- apply(lambda, 1, function(lambda)
+     sum((alpha.hat - 1) * log(lambda)))
> log.h.lambda <- apply(lambda, 1, function(lambda) {
+     foo <- lambda - lambda.hat
+     bar <- sum(foo^2 / lambda.hat)
+     return(- n * bar / 2)})
> weigh <- log.g.lambda - log.h.lambda
```

The vector `weigh` now contains the log unnormalized importance weights. Because of the constraints, we do not know the normalizing constants for the distributions. Hence these unnormalized importance weights are useless for importance sampling. By normalizing them, however, we do make them useful (Geyer, 2011, Section 11.1).

```
> weigh <- weigh - max(weigh)
> weigh <- exp(weigh)
> weigh <- weigh / sum(weigh)
```

If w_i are these importance weights (components of our R vector `weigh`) and λ_i are the samples (rows of our R matrix `lambda`), then for any function f the weighted average

$$\sum_{i=1}^{m} w_i f(\lambda_i)$$

is an unbiased estimator of the true posterior expectation $E\{f(\lambda) \mid y\}$, assuming this posterior expectation exists (that is, that f is integrable with respect to the posterior distribution).

It will be a good estimator so long as the importance weights are not too uneven. So let us look at them. The following R statements make the plot Figure 1.

```
> par(mar = c(5, 4, 1, 1) + 0.1)
> plot(log.h.lambda, weigh, log = "y")
```

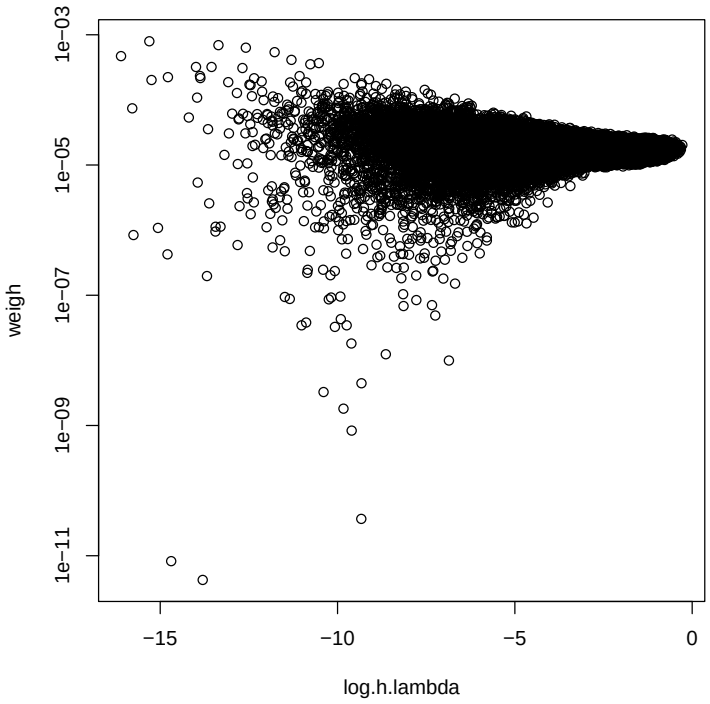

Figure 1: Normalized Importance Weights versus Log Unnormalized Density of Asymptotic Normal Distribution (vertical axis is log scale).

We see from the figure that the normalized importance weights are not too unevenly distributed. One indication of this is Figure 1. Another is the ratio of the maximum importance weight to the average.

```
> max(weigh) * length(weigh)
```

```
[1] 42.84962
```

Hence we conclude that importance sampling will work well (for these simulated data `y`), and we can use our asymptotic approximation to calculate expectations with respect to the exact posterior distribution.

15 Discussion

It is somewhat surprising that we went to all that trouble to simulate the constrained normal approximation to the constrained exact posterior and then didn't use the constrained normal approximation except as an importance sampling distribution. But this does make sense. We need the asymptotic approximation because we do not know how to simulate an equality constrained Dirichlet distribution (we do know how to simulate an unconstrained Dirichlet distribution because it is a product of beta conditionals and marginals, but that is no help), whereas we do know how to simulate an equality constrained normal distribution. Thus the usefulness of the normal approximation is that it allows us to impose the equality constraints. This cannot be done by rejection sampling because the equality constrained distribution lies in a lower-dimensional affine subspace that has probability zero under the unconstrained distribution. In contrast the inequality constraints can be imposed (as we did above) by rejection sampling (simply reject the points that do not satisfy the inequality constraints).

This importance sampling scheme will work well when the normal approximation is good (when n is large) and will not work well when it isn't (when n is small). In this respect, it is no different from any other use of asymptotic approximation.

Rather than produce a massive simulation study (repeat the above with lots of different random number generator seeds, different sample sizes `n` and different dimensions `d`), we have produced a "simulation schema" that allows interested readers to redo this document with different choices of these quantities. Industrious readers can with a bit more work also change the constraints.

References

Geyer, C. J. (2011). Importance sampling, simulated tempering, and umbrella sampling. In *Handbook of Markov Chain Monte Carlo*. Edited by S. P. Brooks, A. E. Gelman, G. L. Jones, and X. L. Meng. Chapman & Hall/CRC, Boca Raton, pp. 295–311.

Geyer, C. J. and Meeden, G. D. (2009). R package `rcdd` (C Double Description for R). Current version 1.1-3 (14 Jul 2009). `http://www.stat.umn.edu/geyer/rcdd/`.

Geyer, C. J. and Meeden, G. D. (submitted). Asymptotics for constrained Dirichlet distributions. Revised and resubmitted to *Bayesian Analysis*.

R Development Core Team (2012). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. `http://www.R-project.org`.