Computing constrained Polya posterior estimates when using judgment sampling

This is a companion to the paper "More efficient inferences using ranking information obtained from judgment sampling" by Glen Meeden and Bo Ra Lee that appeared in the *Journal of Survey Statistics and Methodology* in 2014. Given a sample the following R code computes their point estimate of the population mean and finds its posterior variance. We recall some of the notation from the paper.

- *elds* is a vector of length n and contains the elders in the sample in increasing order.
- *sibs* is a list of length n. $sibs[[i]]$ contains the information about the sibs for $elds[i]$. The value of -1 indicates a sib smaller than its elder and the value 1 indicates a sib larger than its elder.
- R is the the number of simulated copies of the population to be generated.

Here is a simple assignment of *elds* and *sibs*

```
elds<-1:5
sibs<-list(c(-1,1,1),c(1,1),numeric(),c(-1,-1,-1,1),c(-1,-1))
```

Note, for this example, the elder with the value 1 has one sib which is smaller than it and two which are larger, the elder with the value 2 has two sibs which are larger than it, the elder with value 3 has no sibs and so on.

The estimator is based on a constrained version of the Polya posterior. Given a sample the Polya posterior assumes that the only values units in the population can have are those that appeared in the sample. It then uses Polya sampling from an urn containing the observed sample to simulate many complete copies of the population. With a judgment sample the simulation is broken into two steps. First, a constrained Polya sample is drawn from the *elds* to simulate possible values for the *sibs* consistent with the sample information. Then from this combined set of values Polya sampling is used to simulated possible values for all the units not in the sample.

In practice drawing a sample from the constrained Polya posterior can be difficult. So instead we use an importance sampling distribution to simulate possible values for the sibs. This is done using the function *importsmpall*.

```
> elds<-1:5
> sibs<-list(c(-1,1,1),c(1,1),numeric(),c(-1,-1,-1,1),c(-1,-1))
> set.seed(9876643)
> importsmpall<-function (elds, sibs)
+ {
+     n <- length(elds)
+     ans <- rep(0, n)
+     for (i in 1:n) {
```

```
+         ans <- ans + importsmp(elds, i, sibs[[i]])
+     }
+     return(ans)
+ }
> importsmp<-function (elds, ind, sibinfo)
+ {
+     k <- length(sibinfo)
+     n <- length(elds)
+     ans <- rep(0, n)
+     if (k == 0)
+         return(ans)
+     for (j in 1:k) {
+         dum <- importsmpone(elds, ind, sibinfo[j])
+         ans[dum] <- ans[dum] + 1
+     }
+     return(ans)
+ }
> importsmpone<-function (elds, ind, sibinfo)
+ {
+     n <- length(elds)
+     if (ind == 1) {
+         if (sibinfo < 0)
+             ans <- 1
+         else ans <- sample(2:n, 1)
+     }
+     else if (ind == n) {
+         if (sibinfo > 0)
+             ans <- n
+         else ans <- sample(1:(n - 1), 1)
+     }
+     else if (ind == n - 1) {
+         if (sibinfo < 0)
+             ans <- sample(1:(ind - 1), 1)
+         else ans <- n
+     }
+     else {
+         if (sibinfo < 0)
+             ans <- sample(1:(ind - 1), 1)
+         else ans <- sample((ind + 1):n, 1)
+     }
+     return(ans)
+ }
> importsmpall (elds, sibs)

[1] 2 2 1 3 3
```

Note there were a total of 11 sibs in our sample and we see that in this case

2 were assigned to $y = 1$ and 2 more to $y = 2$ and so on.

Once we have assigned values to the sibs we can then simulated a completed copy of the population. Let $p_{i,j}$ denote the proportion of the jth simulated population that takes on the value $eld[i]$. Now if we were generating values for the sibs using their true distribution instead of our importance sampling distribution our estimate for q_i, the true proportion of units in the population with value $eld[i]$, would just be the average of the observed $p_{i,j}$'s over our R simulated copies. Instead for each simulation we need to find its importance sampling weight (up to a constant) and then renormalize them so that they sum to one. Our estimate for q_i is then the weighted average of the $q_{i,j}$'s using these weights instead of taking the straight average. When estimating the population mean we do not need to carry out the second part of the simulation because conditional on a given assignment of values to the sibs we know approximately the expected value of the q_i's, and then we use the fact that an expectation is equal to the expectation of a conditional expectation. Then we just use these expected values and the renormalized importance weights to find the $E(q_i)$'s approximately. Once we have these posterior expectations we then can find our estimate of the population mean.

The next chunk of code finds the $E(q_i)$'s.

```
> findwts<-function (elds, sibs, R)
+ {
+     n <- length(elds)
+     imprtsmpwts <- rep(0, R)
+     dirwts <- matrix(0, R, n)
+      imptsib<-numeric(0)
+     for (i in 1:R) {
+         imptsib <- importsmpall(elds, sibs)
+         imprtsmpwts[i]<-sum(lgamma(rep(1,n) + imptsib) - lgamma(rep(1,n)))
+         dirwts[i, ] <- imptsib + 1
+     }
+     truimprtsmpwts<-rep(0,R)
+     for(i in 1:R){
+       truimprtsmpwts[i]<-1/sum(exp(imprtsmpwts - imprtsmpwts[i]))
+     }
+     avdirwts <- apply(dirwts, 2, function(x) sum(truimprtsmpwts *  x))
+     normedwts<-avdirwts/sum(avdirwts)
+     return(normedwts)
+ }
> R<-10000
> wts<-findwts(elds,sibs,R)
> wts

[1] 0.2248815 0.1588560 0.2789801 0.1341396 0.2031429

> est<-sum(elds*wts)
> est
```

```
[1] 2.931806

>
```

In order to get an interval estimate we need to find the effective sample size which is defined in the paper.

```
> sibscores<-function(sibs)
+   {
+     dd<-sibs
+      n <- length(dd)
+     ans <- rep(0, n)
+     lw <- rep(0, n)
+     up <- rep(0, n)
+     for (i in 1:n) {
+         dum <- dd[[i]]
+         lw[i] <- lw[i] + length(dum[dum < 0])
+         up[i] <- up[i] + length(dum[dum > 0])
+     }
+     ans[1]<-lw[1] + (1/(n-1))*up[1]
+     ans[n]<-(1/(n-1))*lw[n] + up[n]
+     for(i in 2:(n-1)){
+       ans[i]<-(1/(i-1))*lw[i] + (1/(n-i))*up[i]
+     }
+     return(ans)
+   }
> sibwt<-sum(sibscores(sibs))
> n<-length(sibs)
>  effsmpsz<-n + (1/2 - 2/n)*sibwt
> effsmpsz

[1] 5.466667

>
```

The next chunk of code finds the variance of our point estimate of the population mean under the weighted Dirichlet posterior where y is the value of the $elds$ and a are their respective weights based on the $effsmpsz$.

```
> varwtdir<-function(y,a)
+ {
+         a0<-sum(a)
+         den<-a0*a0*(a0 + 1)
+         varp<-(a*(a0-a))/den
+
+         ans<-sum(y*y*varp)
+         n<-length(y)
```

```
+         for(i in 1:(n-1)){
+                 for(j in (i+1):n){
+                         ans<-ans -2*y[i]*y[j]*a[i]*a[j]/den
+                 }
+         }
+         return(ans)
+ }
> a<-effsmpsz*wts
> estvar<-varwtdir(elds,a)
> estvar

[1] 0.3093468
```