More on calibration in R

Let the wt be a vector of design based weights which could have been adjusted for non-response and which may or may not sum to the population size N. Let n be the the length of wt. Let mxv be matrix with n rows where each column is a variable for which we want to use to calibrate our weights. Let $totv$ be the vector of population totals for the columns of mxv. We assume that $totv$ is known.

Our goal is to find a new set of weights say $\gamma = (\gamma_1, \gamma_2 \ldots, \gamma_n)$ which is is a solution to the problem

$$\min_{\gamma} f(\gamma) = \sum_{i=1}^{n} (z_i/wt_i)(\gamma_i - wt_i)^2$$

subject to the constraints

$$\gamma * mxv = totv$$

where the vector z needs to be specfied. Often a good choice for z is just a vector of one's.

This is a standard quadratic programing problem and the R package *quadprog*, as we have seen, will find the solution. The R function, *gencalibrate* given just below does this. This is a more general version of the R function *calibrate* give in the handout "Calibration using quadprog".

The function can be written so that when the constraints are not consistent and there is no solution the function returns $NULL$. One may also include the additional constraints so that no weight gets too big but the following code does not do that.

```
> set.seed(33445566)
> library(quadprog)
> gencalibrate<-function(wt,z,mxv,totv)
+     {
+         dvec<-sqrt(wt*z)
+         cvec<-sqrt(wt/z)
+         n<-length(wt)
+         bvec<-c(totv,rep(1,n))  #rep(1,n) is lower  bd for  final weights .
+         nc<-ncol(mxv)
+         mxcnst<-NULL
+         for(i in 1:nc){
+             mxcnst<-cbind(mxcnst,cvec*mxv[,i])
+         }
+         Dmat<-diag(n)
+         Amat<-cbind(mxcnst,diag(n))
+          meq<-nc   #this makes everything an equality constraint
+  #       Use the next four lines when debugging so you can see the error message.
+ #        out<-solve.QP(Dmat,dvec,Amat,bvec=bvec,meq)
+ #        return(out)
```

```
+ #        ans<-out$solution*sqrt(wt/x)
+ #       return(ans)
+         out<-try(solve.QP(Dmat,dvec,Amat,bvec=bvec,meq),silent=TRUE)
+          if(inherits(out,"try-error")){return(NULL)}
+         else{
+             nwt<-out$solution*sqrt(wt/z)
+          return(nwt)
+         }
+     }
> #wt<-c(15,10,8,9,12,6,4,4)
> #mxv<-cbind(rep(1,8),c(rep(1,3),rep(0,5)),c(10:17))
> #totv<-c(75,46,135)
> wt<-rep(5,8)
> mxv<-cbind(rep(1,8),c(rep(1,4),rep(0,4)),1:8)
> wt%*%mxv

     [,1] [,2] [,3]
[1,]   40   20  180

> totv<-c(42,18,177)
> z<-rep(1,8)
> ans<- gencalibrate(wt,z,mxv,totv)
> ans%*%mxv

     [,1] [,2] [,3]
[1,]   42   18  177

> round(ans,digits=2)

[1]  8.24  5.25  2.27  2.24 10.24  7.25  4.27  2.24
```

In class we discussed another way to adjust a set of weights so that they satisfy a set of known equality constraints. This involved using the function *hitrun* in the *R* package "polyapost". More information about this package is found in the handout "Using polyapost". The next bit of code does this for the setup given here.

```
> library(polyapost)
>  N<-42
>  n<-length(wt)
>  da2<-t(mxv[,-1])
>  db2<-totv[-1]/N
>  da1<-NULL
>  db1<-NULL
>  alpha<-n*wt/sum(wt)
>  dum<-hitrun(alpha,a1=da1,b1=db1,a2=da2,b2=db2,nbatch=1,blen=2000)
>  out<-colMeans(dum$batch) # nbatch=1 so just getting the overall mean
>  weights<-N*out
>  weights%*%mxv

     [,1] [,2] [,3]
[1,]   42   18  177

>  round(weights,digits=2)

[1]  6.78  5.07  3.54  2.60 13.02  5.50  2.90  2.58
```