Using the R package quadprog for calibration

Let the $\hat{wt}_i$'s be a set of design based weights which could have been adjusted for non-response. Now in general, the $\hat{wt}_i$'s need not sum to the population size N. Suppose we wish to calibrate them using some auxiliary vaiable x. One thing to do is to find a new set of weights say $\gamma = \{\gamma_i : i \in s\}$ which is is a solution to the problem

$$\min_{\gamma} f(\gamma) = \sum_{i \in s} (x_i/\hat{wt}_i)(\gamma_i - \hat{wt}_i)^2$$

subject to the constraints

$$\sum_{i \in s} x_i \gamma_i = T_x \quad \text{and} \quad \sum_{i \in s_r} \gamma_i = N.$$

where we assume T_x, the population total of x is known. So we are simultaneously calibrating on x and making sure that the adjusted weights sum to N. This is a standard quadratic programing problem and the R package *quadprog* will find the solution. The R function, *calibrate* given just below does this.

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.

```
> library(quadprog)
>  calibrate<-function(wt,x,totx,N)
+     {
+         ns<-length(wt)
+         mxcnst<-cbind(sqrt(wt*x),sqrt(wt/x))
+         Dmat<-diag(ns)
+         dvec<-sqrt(wt*x)
+         Amat<-cbind(mxcnst,diag(ns))
+         bvec<-c(totx,N,rep(1,ns))  #rep(1,ns) is lower  bd for  final weights .
+         meq<-2  #this makes the first two constraints equality constraints
+  #       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$solution*sqrt(wt/x)
+ #       return(ans)
+         out<-try(solve.QP(Dmat,dvec,Amat,bvec=bvec,meq),silen=TRUE)
+          if(inherits(out,"try-error")){return(NULL)}
+         else{
+             nwt<-out$solution*sqrt(wt/x)
+          return(nwt)
+     }
+     }
```

```
> wt<-c(100,50,35,15,10)
> x<-c(3,7,10,12,15)
> totx<-1240
> N<-225
> ans<- calibrate(wt,x,totx,N)
> round(ans,digits=2)

[1] 133.47  44.89  27.98  11.42   7.23

> sum(ans*x)

[1] 1240

> sum(ans)

[1] 225
```

It is also possible to use this package to make a raking adjustment.

```
> altrake<-function(mxwt,rwtot,cltot)
+     {
+         #mxwt is a 2 by k matrix of weights
+         wt<-c(mxwt[1,],mxwt[2,])
+          ns<-length(wt)
+         nc<-ncol(mxwt)
+         nr<-nrow(mxwt)
+         foo<-rep(0,2*nc)
+         dd<-NULL
+         for(i in 1:nc){
+             dum<-foo
+             dum[c(i,i+nc)]<-1
+             dum<-dum*sqrt(wt)
+             dd<-cbind(dd,dum)
+         }
+         rw1<-c(rep(c(1,0),c(nc,nc)))*sqrt(wt)
+          mxcnst<-cbind(dd,rw1)
+         Dmat<-diag(ns)
+         dvec<-sqrt(wt)
+         Amat<-cbind(mxcnst,diag(ns))
+         bvec<-c(cltot,rwtot[1],rep(1,ns))
+         meq<-nc + nr-1
+         out<-solve.QP(Dmat,dvec,Amat,bvec=bvec,meq)
+         nwt<-out$solution*sqrt(wt)
+         ans<-rbind(nwt[1:nc],nwt[(nc+1):(2*nc)])
+        return(ans)
+ #      next few lines lets the function return NULL when constraints are
+ #      not consistent.
+ #        out<-try(solve.QP(Dmat,dvec,Amat,bvec=bvec,meq),silen=TRUE)
+ #         if(inherits(out,"try-error")){return(NULL)}
+ #        else{
+ #             nwt<-out$solution*sqrt(wt)
+ #            ans<-rbind(nwt[1:nc],nwt[(nc+1):(2*nc)])
+ #            return(ans)
+ #        }
+     }
> #mxwt<-rbind(c(17,16,13),c(7,22,25))
> mxwt<-rbind(c(20,25,10),c(10,15,20))
> rwtot<-c(40,60)
> cltot<-c(20,30,50)
> ans<-altrake(mxwt,rwtot,cltot)
> round(ans,digits=2)
      [,1]  [,2] [,3]
[1,] 10.76 15.14 14.1
```

```
[2,]  9.24 14.86 35.9

> apply(ans,1,sum)

[1] 40 60

> apply(ans,2,sum)

[1] 20 30 50

> rake<-function(mx,rt,ct)
+   {
+     d<-dim(mx)
+     dum<-mx
+     for(i in 1:d[1]){
+       dum[i,]<-dum[i,]*(rt[i]/sum(dum[i,]))
+     }
+     for(i in 1:d[2]){
+       dum[,i]<-dum[,i]*(ct[i]/sum(dum[,i]))
+     }
+     return(dum)
+   }
> rakelp<-function(mx,rt,ct,R)
+   {
+     dum<-mx
+     for(i in 1:R){
+       dum<-rake(dum,rt,ct)
+     }
+     return(dum)
+   }
> ans<-rakelp(mxwt,rwtot,cltot,8)
> round(ans,digits=2)

      [,1]  [,2]  [,3]
[1,] 11.49 15.89 12.62
[2,]  8.51 14.11 37.38

> apply(ans,1,sum)

[1] 40 60

> apply(ans,2,sum)

[1] 20 30 50

>
```