

CDsampling: an R package for Constrained D-Optimal Sampling in Paid Research Studies

Yifei Huang¹, Liping Tong², Jie Yang¹

¹*University of Illinois at Chicago*, ²*Advocate Aurora Health*

Supplementary Material

S1 Examples of Fisher information matrix: Two examples of finding Fisher information matrix with [CDsampling](#);

S2 Comparison of lift-one algorithm and constrained lift-one algorithm: An example of comparison between lift-one algorithm and constrained lift-one algorithm;

S3 Comparison analysis of different sampling strategies: Detailed R codes for a simulation study to compare various sampling strategies mentioned in the paper;

S4 Template functions of constraint index set I : Detailed R codes for the example-specific functions of constraint index set I in [CDsampling](#);

S1 Examples of the Fisher information matrix

In Example [S1.1](#) below, we provide an example of calculating the Fisher information matrix for GLM through the [CDsampling](#) package.

Example S1.1. Consider a research study under a logistic regression model

$$\log\left(\frac{\mu_i}{1-\mu_i}\right) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} \quad (\text{S1.1})$$

where $\mu_i = E(Y_i | \mathbf{x}_i)$, $Y_i \in \{0, 1\}$, $\mathbf{x}_i = (x_{i1}, x_{i2})^\top \in \{(-1, -1), (-1, 1), (1, -1)\}$ and parameters $\beta = (\beta_0, \beta_1, \beta_2) = (0.5, 0.5, 0.5)$. The approximate allocation to the three possible design points, namely, \mathbf{x}_i 's, is $\mathbf{w} = (1/3, 1/3, 1/3)^\top$. To calculate this Fisher information matrix \mathbf{F} given the design \mathbf{w} , we may use `F_func_GLM()` function in the [CDsampling](#) package. Additionally, `W_func_GLM()` function can be used to find the diagonal elements of the matrix \mathbf{W} in the Fisher information matrix (1) of GLM.

```
> beta = c(0.5, 0.5, 0.5)
> X = matrix(data=c(1,-1,-1,1,-1,1,1,1,-1), byrow=TRUE, nrow=3)
> w = c(1/3,1/3,1/3)
> F_func_GLM(w=w, beta=beta, X=X, link='logit')
Dimensions: 3 x 3
Matrix:
-----
      [,1]      [,2]      [,3]
[1,] 0.23500371 -0.07833457 -0.07833457
[2,] -0.07833457 0.23500371 -0.07833457
[3,] -0.07833457 -0.07833457 0.23500371
-----
```

□

We also provide in Example [S1.2](#) with the computation of the Fisher information matrix for MLM through the [CDsampling](#) package.

Example S1.2. We revisit the `trauma_data` example previously analyzed in Section [3.2](#). Consider a research study under a cumulative logit npo model with $J = 5$ response levels and covariates

$(x_{i1}, x_{i2}) \in \{(1, 0), (2, 0), (3, 0), (4, 0), (1, 1), (2, 1), (3, 1), (4, 1)\}$. The model can be written as the following four equations:

$$\begin{aligned}\log\left(\frac{\pi_{i1}}{\pi_{i2} + \dots + \pi_{i5}}\right) &= \beta_{11} + \beta_{12}x_{i1} + \beta_{13}x_{i2} \\ \log\left(\frac{\pi_{i1} + \pi_{i2}}{\pi_{i3} + \pi_{i4} + \pi_{i5}}\right) &= \beta_{21} + \beta_{22}x_{i1} + \beta_{23}x_{i2} \\ \log\left(\frac{\pi_{i1} + \pi_{i2} + \pi_{i3}}{\pi_{i4} + \pi_{i5}}\right) &= \beta_{31} + \beta_{32}x_{i1} + \beta_{33}x_{i2} \\ \log\left(\frac{\pi_{i1} + \dots + \pi_{i4}}{\pi_{i5}}\right) &= \beta_{41} + \beta_{42}x_{i1} + \beta_{43}x_{i2}\end{aligned}$$

where $i = 1, \dots, 8$. We assume that the true parameters $\beta = (\hat{\beta}_{11}, \hat{\beta}_{12}, \hat{\beta}_{13}, \hat{\beta}_{21}, \hat{\beta}_{22}, \hat{\beta}_{23}, \hat{\beta}_{31}, \hat{\beta}_{32}, \hat{\beta}_{33}, \hat{\beta}_{41}, \hat{\beta}_{42}, \hat{\beta}_{43})^\top = (-4.047, -0.131, 4.214, -2.225, -0.376, 3.519, -0.302, -0.237, 2.420, 1.386, -0.120, 1.284)^\top$, and the approximate allocation for the eight design points is $\mathbf{w} = (1/8, \dots, 1/8)^\top$.

To calculate the Fisher information matrix \mathbf{F} for the MLM in this example, we use the function `F_func_MLM()` in the `CDsampling` package. In the R code below, J is the number of levels of the response variable, p is the number of parameters in the model, m is the number of design points or subgroups, and X_i is the model matrix of the i th design point. The function `F_func_MLM()` returns the Fisher information matrix $\mathbf{F}(\mathbf{w})$ given \mathbf{w} .

```
> J=5; p=12; m=8;
> beta = c(-4.047, -0.131, 4.214, -2.225, -0.376, 3.519, -0.302, -0.237, 2.420, 1.386,
-0.120, 1.284)
> Xi=rep(0,J*p*m); dim(Xi)=c(J,p,m)
> Xi[, ,1] = rbind(c( 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1))
> Xi[, ,2] = rbind(c( 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2))
> Xi[, ,3] = rbind(c( 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 1, 3, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 1, 3, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3))
> Xi[, ,4] = rbind(c( 1, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 1, 4, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 1, 4, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4))
> Xi[, ,5] = rbind(c( 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1))
> Xi[, ,6] = rbind(c( 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 1, 2, 1, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 1),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1))
> Xi[, ,7] = rbind(c( 1, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 1, 3, 1, 0, 0, 0, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 1, 3, 1, 0, 0, 0),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1, 1),
+                  c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1))
```

```

> Xi[,8] = rbind(c( 1, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
+               c( 0, 0, 0, 1, 4, 1, 0, 0, 0, 0, 0, 0),
+               c( 0, 0, 0, 0, 0, 0, 1, 4, 1, 0, 0, 0),
+               c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 1),
+               c( 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))
> alloc = rep(1/8,m)
> F_func_MLM(w=rep(1/8,m), beta=beta, X=Xi, link='cumulative')
Dimensions: 12 x 12
Matrix:
-----
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.44505694 1.37915564 0.43609135 -0.37247296 -1.21252053 -0.36379349
[2,] 1.37915564 4.78410934 1.35694145 -1.21252053 -4.31436344 -1.19085831
[3,] 0.43609135 1.35694145 0.43609135 -0.36379349 -1.19085831 -0.36379349
[4,] -0.37247296 -1.21252053 -0.36379349 0.51192600 1.55177413 0.48018678
[5,] -1.21252053 -4.31436344 -1.19085831 1.55177413 5.31193908 1.48241981
[6,] -0.36379349 -1.19085831 -0.36379349 0.48018678 1.48241981 0.48018678
[7,] 0.00000000 0.00000000 0.00000000 -0.09154268 -0.22027625 -0.07471168
[8,] 0.00000000 0.00000000 0.00000000 -0.22027625 -0.64323991 -0.18445802
[9,] 0.00000000 0.00000000 0.00000000 -0.07471168 -0.18445802 -0.07471168
[10,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[11,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[12,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
      [,7]      [,8]      [,9]      [,10]      [,11]      [,12]
[1,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[2,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
[4,] -0.09154268 -0.22027625 -0.07471168 0.00000000 0.00000000 0.00000000
[5,] -0.22027625 -0.64323991 -0.18445802 0.00000000 0.00000000 0.00000000
[6,] -0.07471168 -0.18445802 -0.07471168 0.00000000 0.00000000 0.00000000
[7,] 0.29320484 0.71978889 0.16205254 -0.10435894 -0.25399393 -0.06194131
[8,] 0.71978889 2.13312603 0.41294396 -0.25399393 -0.74842743 -0.15305771
[9,] 0.16205254 0.41294396 0.16205254 -0.06194131 -0.15305771 -0.06194131
[10,] -0.10435894 -0.25399393 -0.06194131 0.17861575 0.45287619 0.06925715
[11,] -0.25399393 -0.74842743 -0.15305771 0.45287619 1.37180187 0.17395849
[12,] -0.06194131 -0.15305771 -0.06194131 0.06925715 0.17395849 0.06925715
-----

```

□

S2 Comparison of lift-one algorithm and constrained lift-one algorithm

Example S2.1. We consider the same logistic regression model (S1.1) as in Example S1.1. That is,

$$\log(\mu_i/(1-\mu_i)) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}$$

with $\mathbf{x}_i = (x_{i1}, x_{i2})^\top \in \{(-1, -1), (-1, 1), (1, -1)\}$ and $(\beta_0, \beta_1, \beta_2) = (0.5, 0.5, 0.5)$. To calculate the \mathbf{W} matrix in (1) for this model, we use the following R codes:

```

> beta = c(0.5, 0.5, 0.5)
> X = matrix(data=c(1,-1,-1,1,-1,1,1,1,-1), byrow=TRUE, nrow=3)
> W_matrix = W_func_GLM(X=X, b=beta)

```

If we don't consider any additional constraints on allocations, we may run the unconstrained lift-one algorithm using `liftone_GLM()` for GLMs or `liftone_MLM()` for MLMs.

```

> w00 = c(1/6, 1/6, 2/3) #an arbitrary starting allocation
> liftone_GLM(X=X, W=W_matrix, reltol=1e-10, maxit=100, random=FALSE, nram=3, w00=w00)

```

Optimal Sampling Results:

=====

Optimal approximate allocation:

	1	2	3
w	0.3333	0.3333	0.3333
w0	0.1667	0.1667	0.6667

Maximum :

0.0077

itmax :

7.0

convergence :

TRUE

This suggests that in the feasible allocation set $S_0 = \{(w_1, w_2, w_3)^\top \in \mathbb{R}^3 \mid w_i \geq 0, i = 1, \dots, 3; \sum_{i=1}^3 w_i = 1\}$, the D-optimal allocation is $\mathbf{w} = (w_1, w_2, w_3) = (1/3, 1/3, 1/3)$. \square

Example S2.2. We consider the same logistic regression model (S1.1) as in Examples S1.1 and S2.1. In this example, we aim for the constrained D-optimal allocation in $S = \{(w_1, w_2, w_3)^\top \in S_0 \mid w_1 \leq \frac{1}{6}, w_3 \geq \frac{8}{15}, 4w_1 \geq w_3\}$. The additional constraints on S_0 can be defined using the following R codes:

```
> g.con = matrix(,nrow=10, ncol=3)
> g.con[1,]=c(1,1,1)
> g.con[2:7, ]=rbind(diag(3),diag(3))
> g.con[8,]=c(1,0,0)
> g.con[9,]=c(0,0,1)
> g.con[10,]=c(4,0,-1)
> g.dir = c("==", rep(">=",3), rep("<=",3),"<=", ">=", ">=")
> g.rhs = c(1, rep(0,3), rep(1,3), 1/6, 8/15, 0)
```

We also need to find the upper boundary and lower boundary functions for $[r_{i1}, r_{i2}]$ in step 3^o of the constrained lift-one algorithm to meet the conditions in S . The r_{i1} and r_{i2} can be obtained as follows:

Case one: If $i = 1$, then $\mathbf{w}_i(z) = (z, \frac{1-z}{1-w_1}w_2, \frac{1-z}{1-w_1}w_3)^\top \in S$ if and only if

$$\begin{cases} 0 \leq z \leq 1/6 \\ 0 \leq \frac{1-z}{1-w_1}w_2 \leq 1 \\ 8/15 \leq \frac{1-z}{1-w_1}w_3 \leq 1 \\ 4z \geq \frac{1-z}{1-w_1}w_3 \end{cases}$$

which is equivalent to

$$\begin{cases} 0 \leq z \leq 1/6 \\ 1 - \frac{1-w_1}{w_2} \leq z \leq 1 \\ 1 - \frac{1-w_1}{w_3} \leq z \leq 1 - \frac{8(1-w_1)}{15w_3} \\ z \geq \frac{w_3}{4-4w_1+w_3} \end{cases}$$

Therefore,

$$\begin{cases} r_{i1} = \max\{1 - \frac{1-w_1}{w_2}, 1 - \frac{1-w_1}{w_3}, \frac{w_3}{4-4w_1+w_3}\} \\ r_{i2} = \min\{1/6, 1 - \frac{8(1-w_1)}{15w_3}\} \end{cases}$$

Case two: If $i = 2$, then $\mathbf{w}_i(z) = (\frac{1-z}{1-w_2}w_1, z, \frac{1-z}{1-w_2}w_3)^\top \in S$ if and only if

$$\begin{cases} 0 \leq \frac{1-z}{1-w_2}w_1 \leq 1/6 \\ 0 \leq z \leq 1 \\ 8/15 \leq \frac{1-z}{1-w_2}w_3 \leq 1 \\ 4\frac{1-z}{1-w_2}w_1 \geq \frac{1-z}{1-w_2}w_3 \end{cases}$$

Similarly, we obtain

$$\begin{cases} r_{i1} = \max\{0, 1 - \frac{1-w_2}{6w_1}, 1 - \frac{1-w_2}{w_3}\} \\ r_{i2} = 1 - \frac{8(1-w_2)}{15w_3} \end{cases}$$

Case three: If $i = 3$, then $\mathbf{w}_i(z) = (\frac{1-z}{1-w_3}w_1, \frac{1-z}{1-w_3}w_2, z)^\top \in S$ if and only if

$$\begin{cases} 0 \leq \frac{1-z}{1-w_3}w_1 \leq 1/6 \\ 0 \leq \frac{1-z}{1-w_3}w_2 \leq 1 \\ 8/15 \leq z \leq 1 \\ 4\frac{1-z}{1-w_3}w_1 \geq z \end{cases}$$

Similarly, we obtain

$$\begin{cases} r_{i1} = \max\{8/15, 1 - \frac{1-w_3}{6w_1}, 1 - \frac{1-w_3}{w_2}\} \\ r_{i2} = \frac{4w_1}{1+4w_1-w_3} \end{cases}$$

We may code r_{i1}, r_{i2} as functions in R as follows:

```
> lower.bound = function(i, w){
+   if(i == 1){
+     return(max(1-(1-w[i])/w[2], 1-(1-w[i])/w[3], (w[3])/(4-4*w[i]+w[3]))))
+   }
+   if(i == 2){
+     return(max(0, 1-((1-w[i])/(6*w[1])), 1-(1-w[i])/w[3])))
+   }
+   if(i == 3){
+     return(max(8/15, (1-(1-w[i])/(6*w[1])), 1-(1-w[i])/w[2])))
+   }
+ }

> upper.bound = function(i, w){
+   if(i == 1){
+     return(min(1/6, 1-(8*(1-w[i])/(15*w[3]))))
+   }
+   if(i == 2){
+     return(1-(8*(1-w[i])/(15*w[3])))
+   }
+   if(i == 3){
+     return((4*w[1])/(1+4*w[1]-w[i]))
+   }
+ }
```

Now we are ready to find the constrained D-optimal allocation in S using the constrained lift-one function `liftone_constrained_GLM()`.

```
> set.seed(123)
> liftone_constrained_GLM(X=X, W=W_matrix, g.con=g.con, g.dir=g.dir, g.rhs=g.rhs,
+ lower.bound=lower.bound, upper.bound=upper.bound, reltol=1e-10, maxit=100,
+ random=FALSE, nram=3, w00=w00, epsilon=1e-8)
```

Optimal Sampling Results:

=====

Optimal approximate allocation:

	1	2	3
w	0.1667	0.3	0.5333
w0	0.1667	0.1667	0.6667

maximum :

0.0055

convergence :

TRUE

itmax :

1.0

deriv.ans :

0.0199, 0.0026, -0.0133

```

-----
gmax :
0.0
-----

reason :
"gmax <= 0"
-----

```

The result indicates that with the feasible allocation set S , our constrained D-optimal allocation converge to a different point with the D-optimal approximate allocation $(w_1, w_2, w_3) = (\frac{1}{6}, \frac{3}{10}, \frac{8}{15})$ and the lift-one loop breaks because $\max g(\mathbf{w}) \leq 0$, where $g(\mathbf{w}) = \sum_{i=1}^m w_i(1 - w_i^*)f'_i(w_i^*)$ in step 8° of the constrained lift-one algorithm (Figure 2). \square

S3 Comparison analysis of different sampling strategies

In this section, we present the detailed simulation codes for comparing the simple random sample without replacement (SRSWOR), constrained D-optimal allocation, constrained EW D-optimal allocation, and constrained uniform allocation as discussed in Figure 5 of Section 3.1.

We begin by defining the model matrix, coefficients, and constraints, consistent with the specifications in Section 3.1, using the following codes:

```

> beta = c(0, 3, 3, 3)
> X=matrix(data=c(1,0,0,0,1,0,1,0,1,0,0,1,1,1,0,0,1,1,1,0,1,1,0,1), ncol=4, byrow=TRUE)
> W=CDsampling::W_func_GLM(X=X, b=beta, link="logit")
> rc = c(50, 40, 10, 200, 150, 50)/200 #available volunteers/sample size
> m = 6
> g.con = matrix(0,nrow=(2*m+1), ncol=m)
> g.con[1,] = rep(1, m)
> g.con[2:(m+1),] = diag(m)
> g.con[(m+2):(2*m+1), ] = diag(m)
> g.dir = c("==", rep("<=", m), rep(">=", m))
> g.rhs = c(1, rc, rep(0, m))

> lower.bound=function(i, w){
+   nsample = 200
+   rc = c(50, 40, 10, 200, 150, 50)/nsample
+   m=length(w) #num of categories
+   temp = rep(0,m)
+   temp[w>0]=1-pmin(1,rc[w>0])*(1-w[i])/w[w>0];
+   temp[i]=0;
+   max(0,temp);
+ }

> upper.bound=function(i, w){
+   nsample = 200
+   rc = c(50, 40, 10, 200, 150, 50)/nsample
+   m=length(w) #num of categories
+   rc[i];
+   min(1,rc[i]);
+ }

```

To find the constrained D-optimal allocation, we use the following codes:

```

> set.seed(123)
> approximate_design = liftone_constrained_GLM(X=X, W=W, g.con=g.con,
+ g.dir=g.dir, g.rhs=g.rhs, lower.bound=lower.bound, upper.bound=upper.bound,
+ reltol=1e-10, maxit=100, random=TRUE, nram=4, w00=NULL, epsilon=1e-8)

```

```
> exact_design = approxtoexact_constrained_func(n=200, w=approximate_design$w,
+ m=6, beta=beta, link='logit', X=X, Fdet_func=Fdet_func_GLM,
+ iset_func=iset_func_trial)
```

To determine the EW D-optimal allocation with the independent uniform priors $\beta_0 \sim \text{uniform}(-2, 2)$, $\beta_1 \sim \text{uniform}(-1, 5)$, $\beta_{21} \sim \text{uniform}(-1, 5)$, and $\beta_{22} \sim \text{uniform}(-1, 5)$, as outlined in Section 3.1, we utilize the same codes provided in Section 3.1:

```
> library(cubature)
> unif.prior <- rbind(c(-2, -1, -1, -1), c(2, 5, 5, 5))
> W.EW.unif = matrix(rep(0,6))
> for (i in 1:6){
+   x = matrix((cbind(1, unique(X))))[i,]
+   W.EW.unif[i] = hcubature(function(beta) dunif(beta[1], min=unif.prior[1,1],
max=unif.prior[2,1])*dunif(beta[2], min=unif.prior[1,2],
max=unif.prior[2,2])*dunif(beta[3], min=unif.prior[1,3],
max=unif.prior[2,3])*dunif(beta[4], min=unif.prior[1,4], max=unif.prior[2,4])*
(exp(x[1]*beta[1]+x[2]*beta[2]+x[3]*beta[3]+x[4]*beta[4]))/(1+exp(x[1]*beta[1]+
x[2]*beta[2]+x[3]*beta[3]+x[4]*beta[4]))^2), lowerLimit = unif.prior[1,],
upperLimit = unif.prior[2,])$integral
+ }

> set.seed(602)
> approximate_design_EW = liftone_constrained_GLM(X=X, W=W.EW.unif, g.con=g.con,
+ g.dir=g.dir, g.rhs=g.rhs, lower.bound=lower.bound, upper.bound=upper.bound,
+ reltol=1e-12, maxit=100, random=TRUE, nram=12, w00=NULL, epsilon=1e-12)

> exact_design_EW = approxtoexact_constrained_func(n=200, w=approximate_design_EW$w,
+ m=6, beta=beta, link='logit', X=X, Fdet_func=Fdet_func_GLM, iset_func=iset_func_trial)
```

To find the constrained uniform allocation, we use the following codes:

```
> w00 = rep(1/200, 6)
> unif_design = approxtoexact_constrained_func(n=200, w=w00, m=6, beta=NULL, link=NULL,
+ X=NULL, Fdet_func=Fdet_func_unif, iset_func=iset_func_trial)
```

Next, we conduct 100 simulations and record the corresponding Root Mean Square Error (RMSE) from both the full data (500 patients) fitted model and the models fitted using the aforementioned samplings with 200 patients. The R codes used for this process are as follows:

```
> nsimu=100
> p=3 # 3 covariates (excluding intercept)
> set.seed(666)
> seeds <- sample(1:10000, nsimu) #random seeds

> # matrix for recording beta estimates
> beta.estimate <- rep(NA, nsimu*(p+1)*(5))
> dim(beta.estimate) = c(nsimu, p+1, 5) # [1] 100 4 8
> dimnames(beta.estimate)[[2]] = paste0("beta", 0:p, seq="")
> dimnames(beta.estimate)[[3]] <- c('full', "SRSWOR", "Unif", "local_Dopt", "EW_Unif")

> # generate X for original 500 patients
> # gender group (0 for female and 1 for male)
> gender = c(rep(0,50), rep(0,40), rep(0,10), rep(1,200), rep(1,150), rep(1,50))
> # age group (0 for 18~25, 1 for 26~64, and 2 for 65 or above)
> age1 = c(rep(0,50), rep(1,40), rep(0,10), rep(0,200), rep(1,150), rep(0,50))
> age2 = c(rep(0,50), rep(0,40), rep(1,10), rep(0,200), rep(0,150), rep(1,50))
> X.original = cbind(gender, age1, age2)
> #probability of Y = 1 given X using coefficient beta under logistic model
```

```

> prob = exp(beta %*% t(cbind(1,X.original))) / (1+exp(beta %*% t(cbind(1,X.original))))

> label = rep(0, length(X.original[,1])) #create label to store category of the population data
> for(k in 1:length(X.original[,1])){
+   for (m in 1:m){
+     if((X.original[k,]==unique(X.original)[m,])[1] & (X.original[k,]==unique(X.original)[m,])[2]
& (X.original[k,]==unique(X.original)[m,])[3]){
+       label[k] = m
+     }
+   }
+ }

> n = tabulate(label) #number of subjects available in each category
> sampling_func = function(label, n.sample, X, s, n, m){
+   J = 0
+   j = rep(0, m)
+   c = rep(0, m)
+   isample = vector()
+   while(J < n.sample){
+     for(k in 1:length(X[,1])){
+       i = label[k]
+       c[i] = c[i] + 1
+       if(j[i]<s[i]){
+         if(runif(1) < (s[i] - j[i])/(n[i] - c[i] + 1)){
+           j[i]=j[i]+1
+           J = J + 1
+           isample = c(isample, k)
+         }
+       }
+     }
+   }
+   return(isample)
+ }

> s_unif = unif_design$allocation #num to sample in constrained uniform from each category
> s_localD = exact_design$allocation #num to sample in local D-optimal from each category
> s_EWD = exact_design_EW$allocation #num to sample in EW D-optimal from each category

> for(isimu in 1:nsimu){
+   set.seed(seeds[isimu])
+   ##Generate Y response under logistic model
+   Y = rbinom(n=length(X.original[,1]), size=1, prob = prob)

+   #Estimate beta with full data
+   beta.estimate[isimu,,"full"] <- glm(Y~X.original,family = "binomial")$coefficients

+   #SRSWOR sample
+   isample1 <- sample(x=1:500, size=200);
+   X1=X.original[isample1,];
+   Y1=Y[isample1];

+   #Estimate beta with SRSWOR sampling
+   beta.estimate[isimu,,"SRSWOR"] <- glm(Y1~X1,family = "binomial")$coefficients

+   #Uniform sample
+   #stratified function returns s[k] (num of samples plan to collect for ith category)
+   isample2 = sampling_func(label=label, n.sample=200, X=X.original, s=s_unif, n=n, m=m)
+   X2=X.original[isample2,];
+   Y2=Y[isample2];

```



```

+ #Estimate beta with Unif sampling
+ beta.estimate[isimu,,"Unif"] <- glm(Y2~X2,family = "binomial")$coefficients

+ #Local D-opt Sample
+ isample3 = sampling_func(label=label, n.sample=200, X=X.original, s=s_localD, n=n,m=m)
+ X3=X.original[isample3,];
+ Y3=Y[isample3];

+ #Estimate beta with local D-optimal sampling
+ beta.estimate[isimu,,"local_Dopt"] <- glm(Y3~X3,family = "binomial")$coefficients

+ #EW D-opt Sample
+ isample4 = sampling_func(label=label, n.sample=200, X=X.original, s=s_EWD, n=n, m=m)
+ X4=X.original[isample4,];
+ Y4=Y[isample4];
+ #Estimate beta with EW D-optimal sampling
+ beta.estimate[isimu,,"EW_Unif"] <- glm(Y4~X4,family = "binomial")$coefficients
+ }

```

We prepare the dataset for figure creation using the following R codes:

```

> beta.noint = beta[2:(p+1)]
> ## Estimating beta0 (intercept) compared to true
> btemp0 = abs(beta.estimate[,1,]) #abs(beta0_est - beta0), beta0 = 0
> mse.b0.mean=apply(btemp0,02,mean, na.rm = TRUE)
> mse.b0.sd=apply(btemp0,2,sd) #sd of beta0 mse

> ## Estimating beta1-beta4 compared to true
> btemp=1/3*(beta.estimate[,2:(p+1,)]-beta.noint)^2 #square of beta's error
> btemp=sqrt(apply(btemp,c(1,3),sum, na.rm = TRUE))
> mse.b5all.mean=apply(btemp,2,mean, na.rm = TRUE) #apply mean over column
> mse.b5all.sd=apply(btemp,2,sd, na.rm = TRUE)

> ## separated mse sd for each beta parameter
> btemp1 = sqrt((beta.estimate[,2,] - beta[2])^2)
> mse.b1.mean=apply(btemp1,2,mean, na.rm = TRUE)
> mse.b1.sd=apply(btemp1,2,sd, na.rm = TRUE) #sd of mse

> btemp2 = sqrt((beta.estimate[,3,] - beta[3])^2)
> mse.b2.mean=apply(btemp2,2,mean, na.rm = TRUE)
> mse.b2.sd=apply(btemp2,2,sd, na.rm = TRUE) #sd of mse

> btemp3 = sqrt((beta.estimate[,4,] - beta[4])^2)
> mse.b3.mean=apply(btemp3,2,mean, na.rm = TRUE)
> mse.b3.sd=apply(btemp3,2,sd, na.rm = TRUE) #sd of mse

> output_simulation_rmse = as.data.frame(rbind(cbind(btemp0, category="beta0"),
cbind(btemp, category="all_coef"), cbind(btemp1, category="beta1"), cbind(btemp2,
category="beta21"), cbind(btemp3, category="beta22")))

> library(data.table)
> output_simulation_rmse.long=melt(setDT(output_simulation_rmse), id.vars=c("category"),
variable.name='Method')
> colnames(output_simulation_rmse.long)[3]='RMSE'
> colnames(output_simulation_rmse.long)[1]='Coef'

```

Finally, Figure 5 is generated using the following R codes:

```

> library(ggplot2)
> library(dplyr)

```

```
> output_simulation_rmse.long %>%
+ ggplot(aes(x=Coef, y=as.numeric(RMSE), fill=factor(Method))) +
+ geom_boxplot(alpha=0.7) +
+ stat_summary(fun=mean, color = "black", position = position_dodge(0.75),
+ geom = "point", shape = 18, size = 2, show.legend = FALSE)+
+ scale_fill_grey(start = 0, end = 1, name="Design Method") +
+ xlab("Coefficient")+ylab("RMSE")+ theme_bw()+
+ facet_wrap(~Coef, scales='free')
```

S4 Template functions of constraint index set I

```
> iset_func_trauma <- function(allocation){
+   iset = rep(TRUE,8)
+   if(sum(allocation[1:4])>=392){iset[1:4]=FALSE}
+   if(sum(allocation[5:8])>=410){iset[5:8]=FALSE}
+   return(iset)
+ }turn(iset)
+ }

> iset_func_trial <- function(allocation){
+   Ni = c(50, 40, 10, 200, 150, 50)
+   return(allocation < Ni)
+ }
```