Sample size for the most important cause of heart valve disease

(or the brand of soap most consumers prefer)

$\boldsymbol{\theta} = (\theta_1, \ldots, \theta_5)$, X_1, ..., X_n multinomial

Try a power computation for a likelihood ratio test of θ_1 vs θ_2 using the non-central chisquare distribution.

```
theta <- c(0.4,0.2,0.1,0.25,0.05); n <- 100
t1 <- theta[1] ; t2 <- theta[2]
alpha <- 0.05
ncp <- 2*n* (  t1*log(t1)+t2*log(t2)-(t1+t2)*log((t1+t2)/2)  )
cval <- qchisq(1-alpha,1)
pow1 <- 1-pchisq(cval,1,ncp) ; pow1
# pow1 = 0.7411679
```

Compare pow2 by simulation. First illustrate how the `rmultinom` and `apply` functions work.

```
> testdat <- rmultinom(10,1,theta); testdat
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    0    0    0    0    0    0    0     0
[2,]    0    0    0    1    0    0    0    1    0     0
[3,]    0    0    0    0    0    0    0    0    0     1
[4,]    0    0    1    0    1    0    1    0    1     0
[5,]    0    0    0    0    0    1    0    0    0     0
> apply(testdat,1,mean) # Apply the mean function to dimension 1 (rows) of testdat
[1] 0.2 0.2 0.1 0.4 0.1
```

```
# Now compare pow2 by simulation.
set.seed(32448)
nsim <- 10000; pow2 <- 0

for(i in 1:nsim)
    {
    rdat <- rmultinom(n,1,theta)
    xbar <- apply(rdat,1,mean); x1  <- xbar[1]; x2 <- xbar[2]
    G <- 2*n* (  x1*log(x1)+x2*log(x2)-(x1+x2)*log((x1+x2)/2)  )
    if(G > cval) pow2 <- pow2+1
    } # Next simulation
pow2 <- pow2/nsim
pow1; pow2
```

```
> pow1; pow2
[1] 0.7411679
[1] 0.7457
```

So I believe it. Restart. First proceeding analytically with the Bonferroni approximation,

```
theta <- c(0.4,0.2,0.1,0.25,0.05)

# Define function apow: Approximate power
apow <- function(n,truth,alpha=0.05)
    {
    k <- length(truth); p <- numeric(k-1)
    bcval <- qchisq(1-alpha/choose(k,2),1) # Bonferroni critical value
    t1 <- truth[1]
    for(j in 2:k)
        {
        t2 <- truth[j]
        ncp <- 2*n* (  t1*log(t1)+t2*log(t2)-(t1+t2)*log((t1+t2)/2)  )
        p[j-1] <- 1-pchisq(bcval,1,ncp)
        }
        # print(p)
        apow <- 1 - sum(1-p)
        apow
    } # End function apow

####### Some output #######
> apow(200,truth=theta)
[1] 0.2450348
> apow(300,truth=theta)
[1] 0.622631
> apow(400,truth=theta)
[1] 0.8159864
> apow(380,truth=theta)
[1] 0.7870189
> apow(390,truth=theta)
[1] 0.801994
> for(nn in 380:390) cat(nn,"  ",apow(nn,truth=theta),"\n")
380    0.7870189
381    0.7885625
382    0.7900957
383    0.7916185
384    0.793131
385    0.7946334
386    0.7961255
387    0.7976076
388    0.7990797
389    0.8005418
390    0.801994
```

So it LOOKS like n=389. My guess is that this is conservative. I expect that for these parameter values, results of the tests are VERY dependent. If 1vs4 is significant, the others probably are too (except maybe 1vs2). Better get it by simulation.

But first, simulate the multinomial data more efficiently. I was simulating the generalization of Bernoullis; simulate a general multinomial, which is the sum of n such, like simulating a binomial instead of just a Bernoulli. For example:

```
> theta <- c(0.4,0.2,0.1,0.25,0.05)
> rmultinom(10,1,theta)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    0    0    1    0    1    0    0    0     1
[2,]    0    1    0    0    0    0    0    1    0     0
[3,]    0    0    0    0    0    0    0    0    0     0
[4,]    0    0    1    0    1    0    1    0    1     0
[5,]    0    0    0    0    0    0    0    0    0     0

> rmultinom(1,10,theta)
     [,1]
[1,]    7
[2,]    0
[3,]    1
[4,]    2
[5,]    0
```

Simulate 6 of these, with n=100:

```
> rmultinom(6,100,theta)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   35   42   42   37   43   32
[2,]   18   19   15   20   24   22
[3,]   13   10   16   10    8   10
[4,]   30   21   21   30   23   28
[5,]    4    8    6    3    2    8
```

Make the columns into means:

```
> Xbar <- rmultinom(6,100,theta)/100 ; Xbar

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] 0.40 0.41 0.31 0.29 0.40 0.42
[2,] 0.17 0.23 0.21 0.28 0.22 0.22
[3,] 0.14 0.11 0.12 0.11 0.11 0.06
[4,] 0.26 0.19 0.30 0.28 0.23 0.24
[5,] 0.03 0.06 0.06 0.04 0.04 0.06
```

Now make a vector of test statistics, one for each column

```
n <- 100
x1  <- Xbar[1,]; x2 <- Xbar[2,]
G <- 2*n* (  x1*log(x1)+x2*log(x2)-(x1+x2)*log((x1+x2)/2)  )
```

```
> n <- 100
> x1  <- Xbar[1,]; x2 <- Xbar[2,]
> G <- 2*n* (  x1*log(x1)+x2*log(x2)-(x1+x2)*log((x1+x2)/2)  )
> rbind(Xbar,G)
      [,1]     [,2]     [,3]       [,4]     [,5]    [,6]
  0.400000 0.410000 0.310000 0.29000000 0.400000 0.42000
  0.170000 0.230000 0.210000 0.28000000 0.220000 0.22000
  0.140000 0.110000 0.120000 0.11000000 0.110000 0.06000
  0.260000 0.190000 0.300000 0.28000000 0.230000 0.24000
  0.030000 0.060000 0.060000 0.04000000 0.040000 0.06000
G 9.550544 5.131448 1.935109 0.01754476 5.301811 6.35592
```

This is good. Restart.

```
theta <- c(0.4,0.2,0.1,0.25,0.05)
set.seed(32448)

spow <- function(n,truth,alpha=0.05,M=1000)
    {
    k <- length(truth); p <- numeric(k-1); G <- NULL; spow <- 0
    bcval <- qchisq(1-alpha/choose(k,2),1) # Bonferroni critical value
    Xbar <- rmultinom(M,n,theta)/n
    x1  <- Xbar[1,]
    for(j in 2:k)
        {
        x2 <- Xbar[j,]
        G <- rbind(G, 2*n* (x1*log(x1)+x2*log(x2)-(x1+x2)*log((x1+x2)/2)) )
        }
        # print(rbind(Xbar,G))
        # Now loop
        for(i in 1:M) if( (min(G[,i])>bcval) && (Xbar[1,i]==max(Xbar[,i])) )
                        spow <- spow+1
        spow <- spow/M
        spow
    } # End function spow

# Start with the approximate answer of n = 389 using the default 1000 simulations
```

```
> spow(n=389,truth=theta)
[1] 0.81
>
> # wow
```

Some more output:

```
> for(nn in 385:395) cat("n = ",nn,"  Power = ", spow(n=nn,truth=theta,M=5000),"\n")
n =  385   Power =  0.8078
n =  386   Power =  0.7956
n =  387   Power =  0.8086
n =  388   Power =  0.814
n =  389   Power =  0.8108
n =  390   Power =  0.8136
n =  391   Power =  0.8088
n =  392   Power =  0.813
n =  393   Power =  0.817
n =  394   Power =  0.8118
n =  395   Power =  0.829
> for(nn in 385:387) cat("n = ",nn,"  Power = ", spow(n=nn,truth=theta,M=10000),"\n")
n =  385   Power =  0.8091
n =  386   Power =  0.8077
n =  387   Power =  0.805
> for(nn in 380:390) cat("n = ",nn,"  Power = ", spow(n=nn,truth=theta,M=10000),"\n")
n =  380   Power =  0.7949
n =  381   Power =  0.8016
n =  382   Power =  0.8044
n =  383   Power =  0.8116
n =  384   Power =  0.8154
n =  385   Power =  0.8115
n =  386   Power =  0.81
n =  387   Power =  0.8118
n =  388   Power =  0.8072
n =  389   Power =  0.8162
n =  390   Power =  0.8114
> for(nn in 380:382) cat("n = ",nn,"  Power = ", spow(n=nn,truth=theta,M=50000),"\n")
n =  380   Power =  0.80144
n =  381   Power =  0.80014
n =  382   Power =  0.80298
> for(nn in 379:382) cat("n = ",nn,"  Power = ", spow(n=nn,truth=theta,M=50000),"\n")
n =  379   Power =  0.79622
n =  380   Power =  0.79976
n =  381   Power =  0.79868
n =  382   Power =  0.80288
```

Some comments:

1. I guess n=382 is okay.

2. The approximate calculation yielding n = 389 was very good -- only a little conservative.

3. Any good function for power calculation by simulation should include a 99% margin of error.

4. There would be great value in an algorithm for efficiently choosing a sequence of (n,M) values. This is always an issue when choosing sample size according to power when you are estimating the power by simulation.