Math 420 HW 4 solution

Fall 2023

Washington University in St. Louis

Instruction:

Please type your answers clearly and show your work neatly. You are encouraged to use the Rmarkdown version of this assignment as a template to submit your work. Unless stated otherwise, all programming references in the assignment will be in R. For this assignment, problems roughly covers content from Split-plot Designs (Ch. 8 of the text) and a little bit of response surface methodology.

Problem 1

(a) Describe an example of split-plot design from a topic you are familiar with. Explain the process and relevant factors as well as their levels.

(b) Write the mathematical formula for the example from (a)

Problem 2

Modify the R code in Section 8.2.1 to create

(a) A randomized list for a split-plot experiment with completely randomized whole plots where there is one whole-plot factor A with 3 levels and two replicate whole plots for each level, and one split-plot factor B with three levels. Label the levels with `c(1,2,3)` rather than `c=("low","mid","high")` as shown in Section 8.2.1.

This design has two factors - A(whole-plot) and B(subplot). If we want to have two replicates for each factor-level combination, the full factorial design would have needed $2 \times 3^2 = 18$ separate sample. In this split-plot design, we can use 6 samples to achieve the same number of runs, by splitting each whole-plot into 3 randomized levels. Below is the code to build the design:

```
wp <- expand.grid(A = c(1,2,3),stringsAsFactors = T)
sp <- expand.grid(B = c(1,2,3),stringsAsFactors = T)

#two replicates for the whole-plot levels
wp <- rbind(wp, wp)
## set up model design using optBlock.
## There are a total of 6 whole-plot blocks, each has 3 levels of subplot factor
splitP <- optBlock( ~ A * B, withinData= sp,
blocksizes = rep(3,6), wholeBlockData = wp)
## original design
print(splitP$Blocks)
```

```
## $B1
##     A B
## 1   1 1
## 1.1 1 1
## 1.2 1 3
```

```
##
## $B2
##     A B
## 2   2 2
## 2.1 2 2
## 2.2 2 2
##
## $B3
##     A B
## 3   3 1
## 3.1 3 3
## 3.2 3 3
##
## $B4
##     A B
## 4   1 1
## 4.1 1 1
## 4.2 1 3
##
## $B5
##     A B
## 5   2 2
## 5.1 2 2
## 5.2 2 2
##
## $B6
##     A B
## 6   3 1
## 6.1 3 3
## 6.2 3 3
```

In here, we can create additional steps to randomize the runs, using a 2-step randomization scheme. Step 1: randomize whole plots and step 2: within each whole-plot, randomize the sub-plots:

```
## randomizing the plots -- in two stages
split_df<-data.frame(splitP$design)
## stage 1: randomize the whole plot
split_df[,'block']<-floor(as.numeric(row.names(split_df)))
rand_block<-sample(1:6,6,replace = FALSE)
split_df1<-data.frame()
for(i in rand_block){
  split_df2<-split_df[split_df$block==i,]
  ## stage 2: randomzie the subpplot levels within each whole plot
  split_df2$B<-sample(1:3,3,replace = FALSE)
  split_df1<-rbind(split_df2,split_df1)
}
```

The result is as follows:

```
print(split_df1)
```

```
##     A B block
## 4   1 1     4
## 4.1 1 3     4
## 4.2 1 2     4
## 3   3 3     3
```

```
## 3.1 3 2     3
## 3.2 3 1     3
## 2   2 3     2
## 2.1 2 1     2
## 2.2 2 2     2
## 5   2 2     5
## 5.1 2 1     5
## 5.2 2 3     5
## 6   3 2     6
## 6.1 3 1     6
## 6.2 3 3     6
## 1   1 1     1
## 1.1 1 2     1
## 1.2 1 3     1
```

(b) Write the model for the design you created in (a).

The model can be represented as follows:

$y_{ijk} = \mu + \alpha_i + w_{(i)j} + \beta_k + \alpha\beta_{ik} + \epsilon_{ijk}$

where: - μ is the overall effect - α_i is the fixed effect for factor A level i - $w_{(i)j}$ is the random effect for whole-plot j assigned to level i of factor A - β_k is the fixed effect for factor B level k - $\alpha\beta_{ik}$ is the interaction effect for A and B, at the sub-plot level - ϵ_{ijk} is the model error

(c) Create a randomized list for a split-plot experiment with completely randomized whole plots where there are two whole-plot factors A and B each with two levels and two replicate whole plots per treatment combination and two split-plot treatments C and D each with three levels.

```
wp <- expand.grid(A = c(1,2),B = c(1,2),stringsAsFactors = T)
sp <- expand.grid(C = c(1,2,3),D = c(1,2,3),stringsAsFactors = T)

#two replicates for the whole-plot levels
wp <- rbind(wp, wp)
## set up model design using optBlock:
## there are a total of 8 whole-plots for A+B,
## each with 9 factor-level combinations for C+D.
splitP_1c <- optBlock( ~ (A + B)*(C+D+C:D), withinData= sp,
blocksizes = rep(9,8), wholeBlockData = wp)
```

(d) Write the model for the design you created in (c).

The model can be represented as (note that some of the higher order interaction terms can be omitted if not significant) :

$y_{ijklm} = \mu + b_i + \alpha_j + \beta_k + \alpha\beta_{jk} + w_{(ij)k}$

$+\gamma_l + \delta_m + \gamma\delta_{lm} + \alpha\gamma_{jl} + \alpha\delta jm$

$+\beta\gamma_{kl} + \beta\delta_{km} + \alpha\beta\gamma_{jkl} + \alpha\beta\delta_{jkm}$

$\alpha\gamma_{jkl} + \beta\gamma\delta_{klm} + \alpha\beta\gamma\delta_{jklm} + \epsilon_{ijklm}$

Problem 3

Modify the R code using the `FrF2` function in Section 8.3.2 (for creating the design for the sausage-casing experiment with two whole-plot factors, two blocks of whole plots, and two split-plot factors) to create a design with two whole-plot factors, two blocks of whole plots, and three split-plot factors. Each factor has only two levels.

In this problem, there are a total of 5 factors and 2×32 runs (since there are 2 blocks). Since each whole-plot factor has 2 levels, and 2 blocks, in total we have 8 whole-plots. We can use the following R code to create the design:

```
des_3 <-FrF2(nruns = 32,
             nfactors = 5,
             WPs = 8,
             nfac.WP = 2,
             factor.names = c('A','B','C','D','E'))
```

```
## Warning in FrF2(nruns = 32, nfactors = 5, WPs = 8, nfac.WP = 2, factor.names =
## c("A", : There are fewer factors than needed for a full factorial whole plot
## design. 1 dummy splitting factor(s) have been introduced.
```

Problem 4

Kuehl (2000) reports the results of an experiment conducted at a large seafood company to investigate the effect of storage temperature and type of seafood upon bacterial growth on oysters and mussels. Three storage temperatures were studied (0C, 5C, and 10C). Three cold storage units were randomly assigned to be operated at each temperature. Within each storage unit, oysters and mussels were randomly assigned to be stored on one of the two shelves. The seafood was stored for 2 weeks at the assigned temperature, and at the end of the time the bacterial count was obtained from a sample on each shelf. The resulting data (log bacterial count) is shown in p.345 of the book.

(a) What is the experimental unit for temperature?

The experimental unit for temperature is a storage unit for the seafood.

(b) Why was it necessary to include nine storage units instead of three?

Using nine storage units allow for randomization and replicates of the treatment-level combinations, so that we can better address the variability of the effects with the experiment results.

(c) What is the experimental unit for seafood type?

The experimental unit for seafood type is the small shelf in the fridge where each different seafood type is placed

(d) Write the model for the data.

This experiment has one whole-plot factor (temperature, α) and one sub-plot factor (seafood type, β) with w representing the random effects of the whole-plot. The model can be represented mathematically as $y_{ijk} = \mu + \alpha_i + w_{(i)j} + \beta_k + \alpha\beta_{ik} + \epsilon_{ijk}$

(e) Analyze the data to determine what temperature and type of seafood have significant effects.

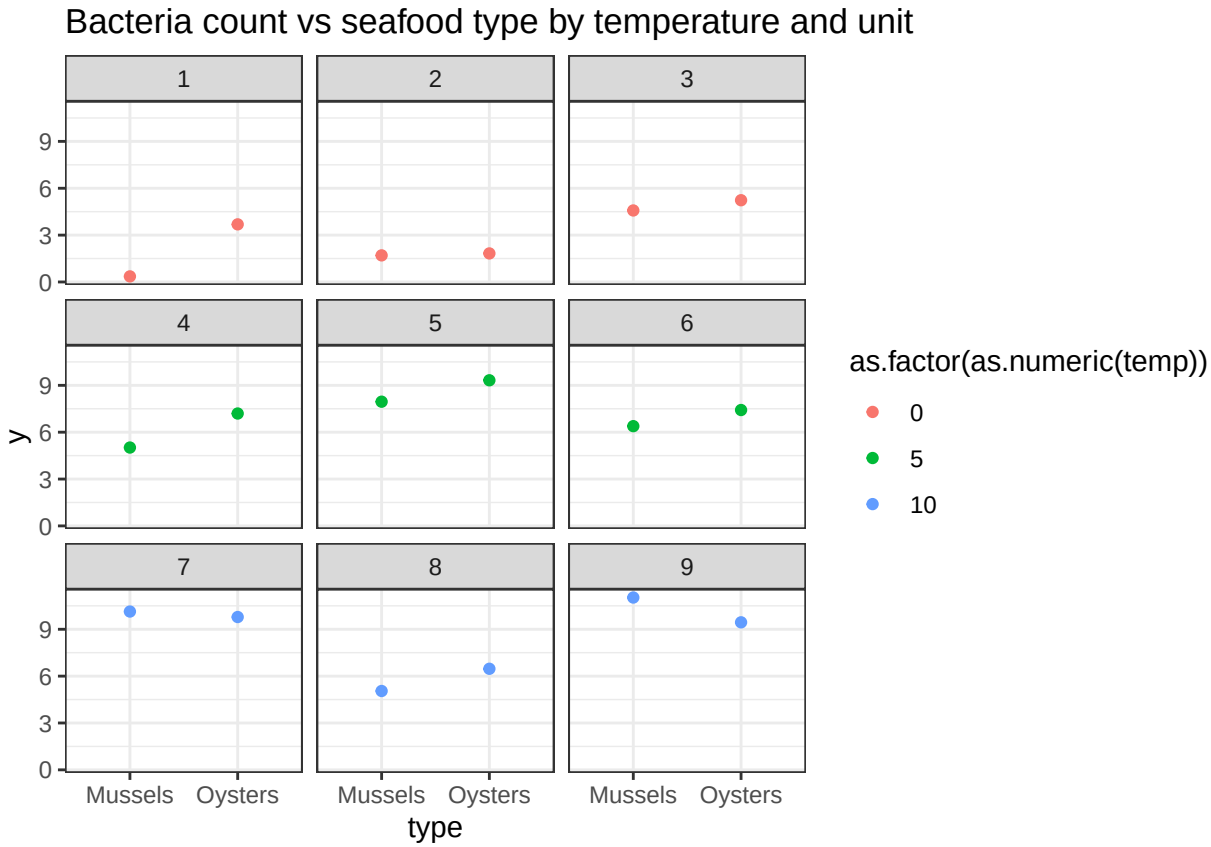
First let's create the dataset given by the book:

```
unit <- rep(seq(1, 9), each = 2)
temp <- rep(c('0','5','10'), each = 6)
type <-rep(c('Oysters', 'Mussels'),9)
y = c(3.6882,0.3565,1.8275,1.7023,5.2327,4.578,7.195,5.0169,9.3224,
7.9519,7.4195,6.3861,9.7842,10.1352,6.4703,5.0482,9.4442,11.0329)
seafood_data <- data.frame(unit=unit,temp=temp, type = type, y = y)
```

We can also visualize the data to get a feel of what the result looks like for different factor-level combination

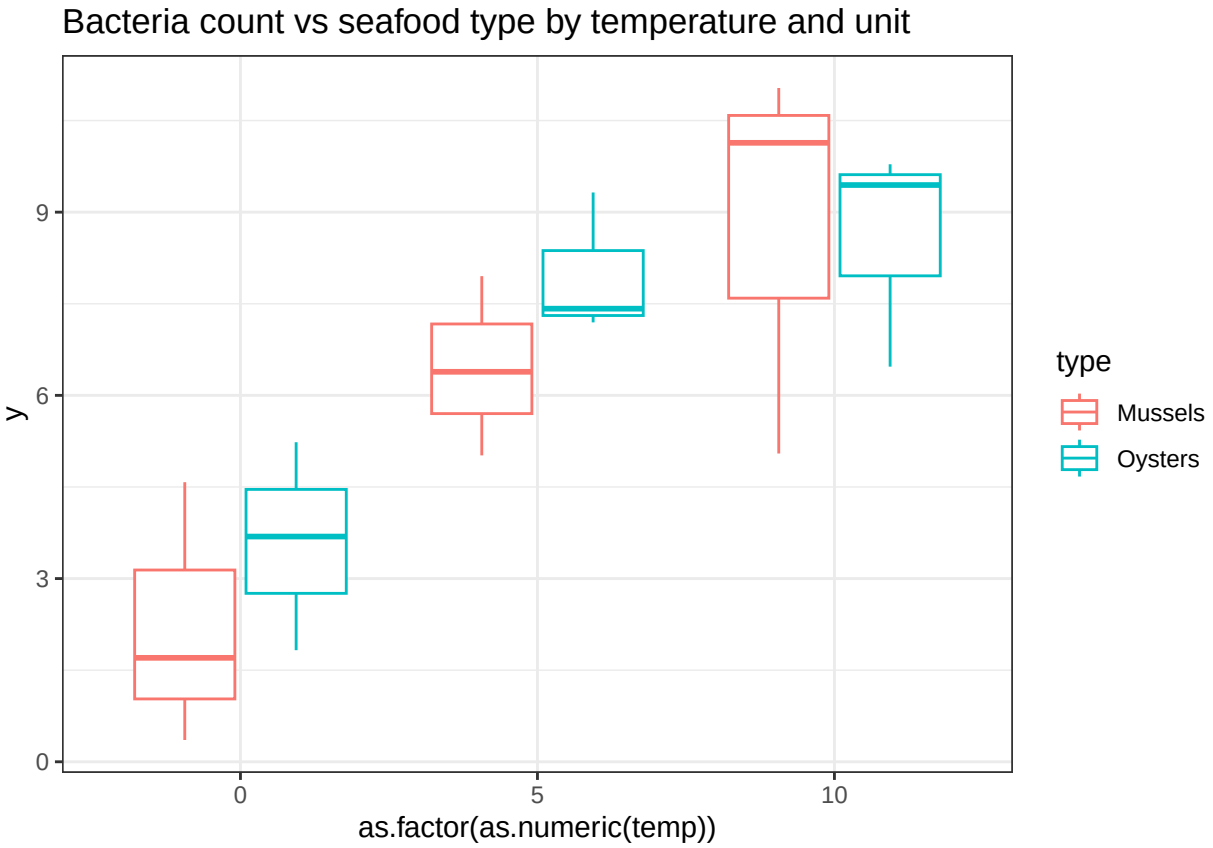
Below is a view in each storage unit :

```
ggplot(seafood_data)+geom_point(aes(x=type,y=y,col=as.factor(as.numeric(temp))))+facet_wrap(~unit,ncol=
```



And this is a view of overall bacteria count vs temperature, grouped by seafood type. There seems to be a pretty clear visual trend of temperature effect on bacteria growth, regardless of seafood type.

```
ggplot(seafood_data)+geom_boxplot(aes(x=as.factor(as.numeric(temp)),y=y,col=type))+theme_bw()+ggtitle('
```



Now, we can `lmer()` function to fit the model represented in d) to get a better view of the effects

```
lme_sf <- lmerTest::lmer(y~temp*type+ (1|unit)+ (1|temp:unit),
                data = seafood_data)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## unable to evaluate scaled gradient
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge: degenerate Hessian with 1 negative eigenvalues
```

```
## Warning: Model failed to converge with 1 negative eigenvalue: -2.9e-05
```

(f) Interpret any significant effects.

It does appear that the main significant effects are coming from temperature changes - for instance, storing the seafood at 5C and 10C both result in signficantly higher bacterial growth, regardless of seafood type and their interaction

```
summary(lme_sf)
```

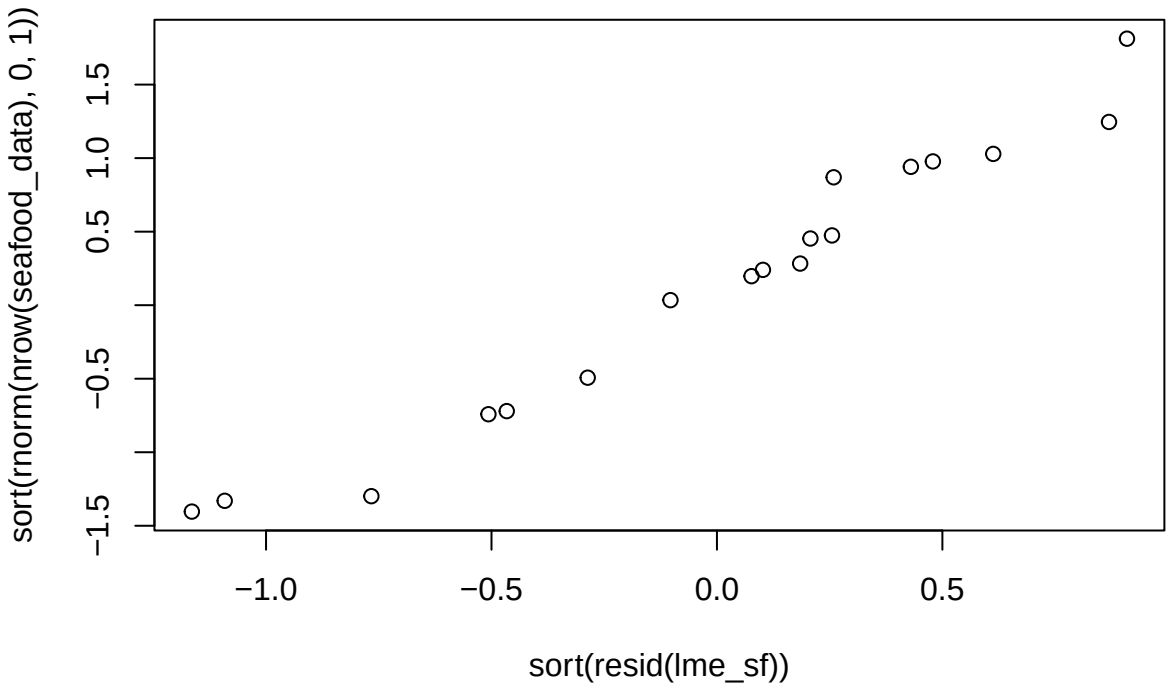
```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: y ~ temp * type + (1 | unit) + (1 | temp:unit)
##    Data: seafood_data
##
## REML criterion at convergence: 52.2
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -1.2064 -0.4364  0.1485  0.4012  0.9421
##
## Random effects:
##  Groups    Name        Variance Std.Dev.
##  unit      (Intercept) 0.8418   0.9175
##  temp:unit (Intercept) 2.3631   1.5372
##  Residual              0.9318   0.9653
## Number of obs: 18, groups:  unit, 9; temp:unit, 9
##
## Fixed effects:
##                     Estimate Std. Error      df t value Pr(>|t|)
## (Intercept)          2.2123     1.1743  7.4990   1.884  0.09878 .
## temp10               6.5265     1.6607  7.4990   3.930  0.00495 **
## temp5                4.2394     1.6607  7.4990   2.553  0.03585 *
## typeOysters          1.3705     0.7882  6.0000   1.739  0.13272
## temp10:typeOysters  -1.5431     1.1146  6.0000  -1.384  0.21554
## temp5:typeOysters    0.1568     1.1146  6.0000   0.141  0.89273
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) temp10 temp5  typOys tm10:O
## temp10      -0.707
## temp5       -0.707  0.500
## typeOysters -0.336  0.237  0.237
## tmp10:typOy  0.237 -0.336 -0.168 -0.707
## tmp5:typOys  0.237 -0.168 -0.336 -0.707  0.500
## optimizer (nloptwrap) convergence code: 0 (OK)
## unable to evaluate scaled gradient
## Model failed to converge: degenerate  Hessian with 1 negative eigenvalues
```

(g) Check the assumptions of the model you used for analysis.

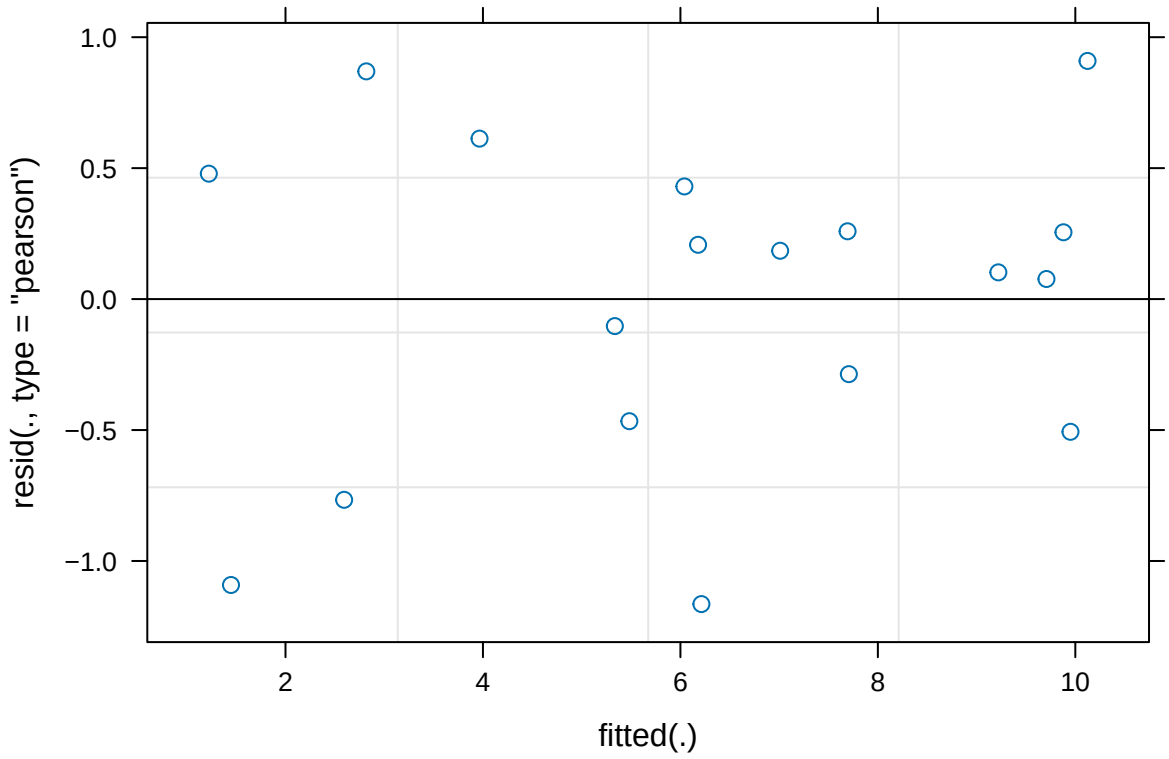
We can use quantile-quantile plot to check for the normality of the model residul - it appears to follow the distribution fairly well.

```
qqplot(sort(resid(lme_sf)),sort(rnorm(nrow(seafood_data),0,1)))
```



We can also plot the model residual directly to check for the constant variance assumption, which seems to be the case:

```
plot(lme_sf)
```



Problem 5

Create a central composite design for two factors using the `rsm` package.

(a) Create the uniform precision CCD and store the design along with random numbers (simulated response) in a data frame.

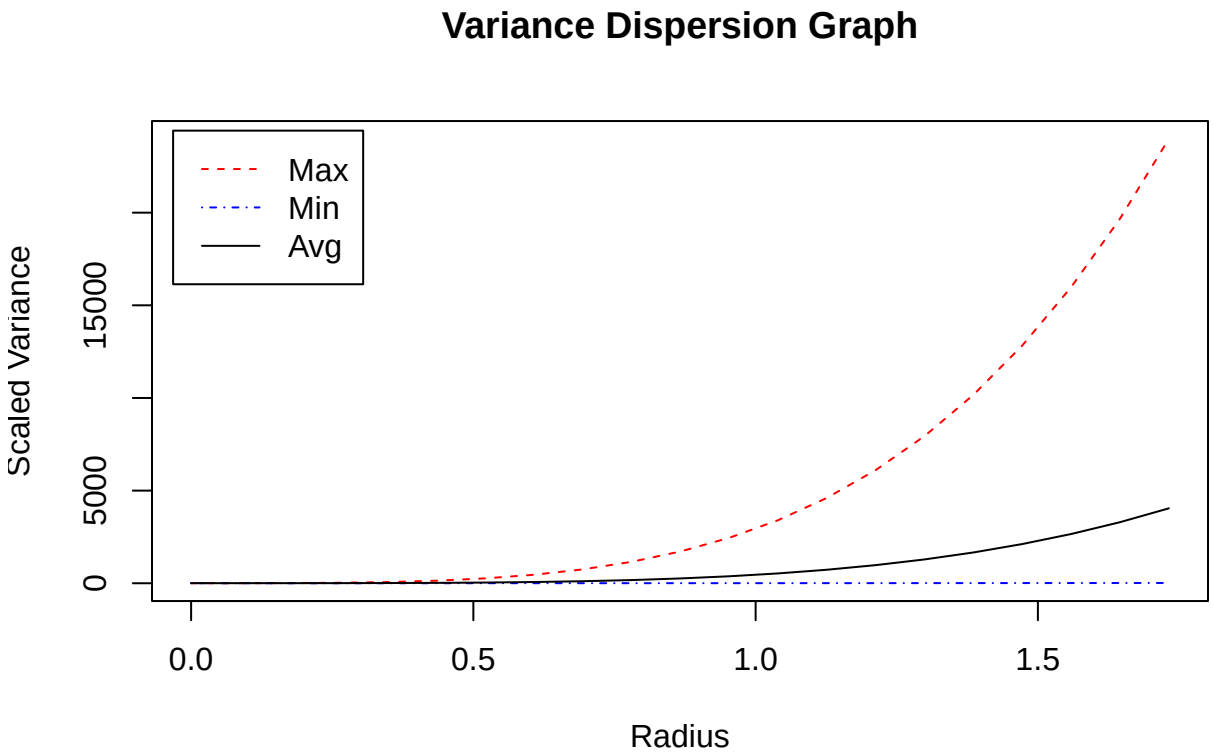
This can be done using the following code (also shown in lecture19):

```
rotd <- ccd(basis = 2, # number of factors
            alpha = "rotatable", #settings for the design
            randomize = FALSE)
## combine with simulated response (uniform distribution)
rotdm <- cbind(rotd[ , 3:4],runif(nrow(rotd)))
```

(b) Use the `Vdgraph` package to make a variance dispersion graph or a fraction of design space plot of the design you created.

```
Vdgraph(rotdm)
```

```
## number of design points= 16
## number of factors= 3
```




```
##           Radius      Maximum   Minimum     Average
##  [1,] 0.00000000     4.585890  4.585890    4.585890
##  [2,] 0.08660254     5.283237  4.115929    4.639202
##  [3,] 0.17320508    10.186020  2.844202    5.101273
##  [4,] 0.25980762    26.452569  2.145575    6.878510
##  [5,] 0.34641016    64.318931  2.238437   11.481588
##  [6,] 0.43301270   137.123130  2.463576   21.025457
##  [7,] 0.51961524   261.305374  2.730981   38.229334
##  [8,] 0.60621778   456.409126  3.036213   66.416710
##  [9,] 0.69282032   745.078796  3.376771  109.515343
## [10,] 0.77942286  1153.064749  3.758456  172.057264
## [11,] 0.86602540  1709.215254  4.194037  259.178777
## [12,] 0.95262794  2445.475651  4.699557  376.620452
## [13,] 1.03923048  3396.914172  5.295048  530.727133
```

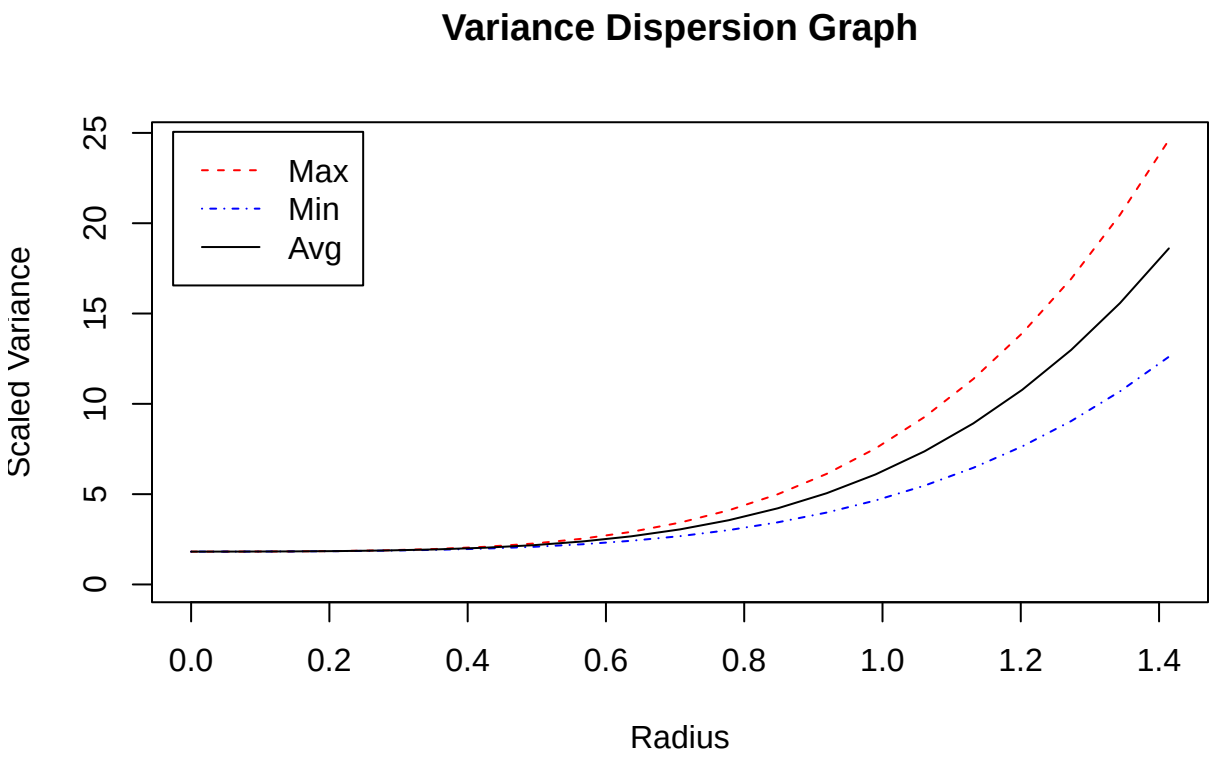
```
## [14,] 1.12583302  4601.719790  6.002285  728.447934
## [15,] 1.21243557  6101.094401  6.842973  977.336240
## [16,] 1.29903811  7939.443823  7.840742 1285.549706
## [17,] 1.38564065 10164.215775  9.026118 1661.850258
## [18,] 1.47224319 12825.735653 10.419189 2115.604094
## [19,] 1.55884573 15978.426270 12.055191 2656.781682
## [20,] 1.64544827 19678.220888 13.966001 3295.957760
## [21,] 1.73205081 23985.549880 16.181442 4044.311338
```

(c) Repeat (a) through (b) for a face-centered cube design (i.e., CCD with axial points at ± 1).

```
faced <- ccd(basis = 2, # number of factors
             alpha = "faces", #settings for the design
             randomize = FALSE)

##taking just the coded factor levels
facedm<- cbind(faced[ , 3:4],runif(nrow(faced)))
## distribution of variance
Vdgraph(faced[ , 3:4])
```

```
## number of design points= 16
## number of factors= 2
```




```
##          Radius   Maximum   Minimum   Average
##  [1,] 0.00000000  1.818182  1.818182  1.818182
##  [2,] 0.07071068  1.820742  1.820667  1.820705
##  [3,] 0.14142136  1.830083  1.828861  1.829461
##  [4,] 0.21213203  1.851045  1.844970  1.848008
##  [5,] 0.28284271  1.891884  1.872679  1.882279
##  [6,] 0.35355339  1.964017  1.917140  1.940578
##  [7,] 0.42426407  2.082184  1.984982  2.033582
##  [8,] 0.49497475  2.264380  2.084304  2.174341
```

```
##  [9,] 0.56568542  2.531880  2.224679  2.378279
## [10,] 0.63639610  2.909227  2.417152  2.663190
## [11,] 0.70710678  3.424242  2.674242  3.049242
## [12,] 0.77781746  4.108015  3.009940  3.558978
## [13,] 0.84852814  4.994909  3.439709  4.217309
## [14,] 0.91923882  6.122561  3.980486  5.051523
## [15,] 0.98994949  7.531879  4.650679  6.091279
## [16,] 1.06066017  9.267045  5.470170  7.368608
## [17,] 1.13137085 11.375515  6.460315  8.917915
## [18,] 1.20208153 13.908015  7.643940 10.775978
## [19,] 1.27279221 16.918545  9.045345 12.981945
## [20,] 1.34350288 20.464379 10.690304 15.577341
## [21,] 1.41421356 24.606061 12.606061 18.606061
```

(d) Based on the graphs you made, which design do you prefer? State the reason.

The face-centered design is preferred as the range of the variance for the second design appears to be narrower compared to the CCD in part a). This means that the responses collected at these design point should theoretically have more consistent values as well.