Stat 430 – Design of Experiments. Lab 8, Nov 5 and Nov 19 2012

Agenda:

- Latin and Greco-Latin Squares
- Analysis of Latin Square Data (Exercise 4.22)
- How to fill in an incomplete Latin Square
- Balanced Incomplete Block Design - Theory
- Balanced Incomplete Block Design – Round Robins

Latin and Greco-Latin Squares

Last week we did a randomized complete block design (RCBD). It's called so because...

- **Block design:** Experimental units are assigned to batches or **blocks** , usually by some sort of **blocking factor** that we can't control across all the experimental units. A plot of land could be considered a block because we can only find so many units on one plot.

- **Randomized:** To keep the block effect from confounding with the treatment effect, each treatment is assigned to the units in a block in a (simple) random arrangement.

- **Complete:** **EVERY treatment** is found **in EVERY block** the same number of times. (Typically once)

If there are **two blocking factors**, every unit is in a placed in a unique combination of blocks (i.e. a row and a column). The treatments are assigned to the units so that no treatment appears more than once in a row block OR in a column block.

This design is called a **Latin Square** when there are an equal number of treatments and levels of each blocking factor. (Although, there are easy modifications if there are, say, half as many levels of one blocking factor as another)

When there are **two treatments and two blocking factors**, a Latin Square is build for EACH of the treatments. However, you can't use the same Latin Square for each because no combination of treatments appears more than once. This design is called a **Greco-Latin square**. It also works if there is one treatment and three blocking factors by treating the third blocking factor like a treatment.

Analysis of Latin Square Data (Exercise 4.22)

Consider the following data from Exercise 4.22 (8th Hardcover). We have 4 methods of assembling a device we want to test (Method = Treatment). We're going to see how long each method takes with each of 4 workers (Worker = Block). Also, since workers might tire or get more practiced, they might slow down or speed up near the end. Therefore the order that each worker tries a method also matters (Order = Block).

The treatments are assigned in a Latin Square (left) and the time to assemble (right) are below:

Order	Worker			
	1	2	3	4
1st	C	D	A	B
2nd	B	C	D	A
3rd	A	B	C	D
4th	D	A	B	C

Order	Worker			
	1	2	3	4
1st	10	14	7	8
2nd	7	18	11	8
3rd	5	10	11	9
4th	10	10	12	14

Which is entered into R with the following code:

```
time = c(10,14,7,8,7,18,11,8,5,10,11,9,10,10,12,14)
method = c("C","D","A","B","B","C","D","A","A","B","C","D","D","A","B","C")
order = as.factor( rep(1:4,4))
worker = as.factor(rep(1:4,each=4))
data422 = data.frame(time,method,order,worker)
```

As we did in Lab 7, to include an additional factor, just add it to the linear model.

```
mod = lm(time ~ method + order +  worker, data=data422)
anova(mod)
```

Notice that EACH of the blocking factors takes up (p-1) degrees of freedom. For this reason you can't do an ANOVA on a 2x2 Latin Square unless it's replicated. For such a case see crossover design on p. 165. At the moment we're treating the treatment and blocking factors as if their effects are additive. i.e. There are no interaction or multiplicative effects that we're missing.

One graphical method to check this assumption is the interaction plot. We can use it to check for an interaction between the treatment of method, and the block of worker. If the line segments are not approximately parallel, the effects may be non-additive or an effect is missing.

```
interaction.plot(  data422$method, data422$worker, data422$time)
```

We can also check for non-additivity for method vs. order, and order vs. worker.

```
interaction.plot(  data422$method, data422$order, data422$time)
interaction.plot(  data422$order, data422$worker, data422$time)
```

There is non-additivity in all cases, however:

1. We have three factors and interaction plots can only handle two.

2. We're comparing four sets of line segments in each picture. Crossovers and other non-parallel behaviour will happen by random chance (effectively we're doing multiple testing).

A solution to both problems comes from the Tukey Nonadditivity Test, in the alr3 package. Which is formal rather than graphic.

```
library(alr3)
tukey.nonadd.test(mod) # mod is a linear model
```

Filling in an incomplete Latin Square

A	B	C	D	E
B	C	D		
C				
D				

General strategy / algorithm:

1. Find the most complete row or column and fill that in. Often there's only one possible treatment for a spot.

A	B	C	D	E
B	C	D		
C				
D				

A	B	C	D	E
B	C	D	E	A
C				
D				
E				

A	B	C	D	E
B	C	D	***E***	***A***
C				
D				
E				

2. When there is no deterministic way solve a spot, choose from the ones that would work and continue from there.

A	B	C	D	E
B	C	D	E	A
C	*A,D, or E*			
D				
E				

A	B	C	D	E
B	C	D	E	A
C	***A***			
D	***E***			
E	***D***			

3. Remember that every letter appears in every column and row once. That's why the middle square in this case must be a E, otherwise there would be nowhere legal to put the E.

A	B	C	D	E		A	B	C	D	E		A	B	C	D	E
B	C	D	E	A		B	C	D	E	A		B	C	D	E	A
C	A	***E***				C	A	***E***				C	A	E	***B***	***D***
D	E	Not E				D	E	***B***				D	E	B		
E	D	Not E				E	D	***A***				E	D	A		

A	B	C	D	E		A	B	C	D	E
B	C	D	E	A		B	C	D	E	A
C	A	E	B	D		C	A	E	B	D
D	E	B	***A***	***C***		D	E	B	A	C
E	D	A				E	D	A	***C***	***B***

Balanced Incomplete Block Design – Theory

A BIBD is like an RCBD, but without the completeness requirement. BIBDs are used in situations where the blocks are not large enough to put one of each treatment into a block. However, they're designed in such a way that every treatment appears in the same number of blocks and every PAIR of treatments appear in the same number of blocks.

Rules for a BIBD:

- There is only one treatment factor and one blocking factor.
- There are ***b*** blocks.
- Every block has one unit each of ***k*** levels of treatments (block size = k).
- There are ***a*** treatments in total
- Each treatment occurs ***r*** times.

And...

- Each pair of treatments appears in ***λ*** blocks.

Some key equations: $\lambda = \frac{\mathrm{r(k-1)}}{\mathrm{a-1}}$ and $N = ar = bk$

Where N is the total number of experimental or sampling units.

BIBDs have some design challenges similar to those of Latin Squares. You can't just make blocks haphazardly. For example, it's impossible to make a BIBD for an experiment with a = 6 treatment levels each which appears in r = 4 blocks because...

N = ar = 24, so bk = 24.

You need at least b=4 blocks because each treatment appears in r=4 blocks, and N is an integer, so b = 6, 12, or 24, and k = 4, 2, or 1 respectively. None of the possible k values produce a λ which is an integer, so no BIBD can be constructed from these parameters. There are other restrictions besides making sure all the values of a,b,k,r, and λ are whole numbers, but they're usually covered in a course in applied abstract algebra.

A BIBD with a=b=7 and k=r=3 and λ =1 can be constructed by taking three rows of a standard 7x7 Latin Square as long as those rows aren't consecutive. (example: The 1st,2nd, and 4th rows work; the 1st,2nd,and 3rd don't)

Aside from their construction, there is another complication: The means of treatments aren't directly comparable because they don't appear in the same set of blocks. To analyze the results of a BIBD we need adjusted treatment means and an adjusted $SS_{treatment}$. Q_i is the adjusted treatment total and

$$SS_{treatment(adj.)} = \frac{\sum_{i=1}^{a} Q_i^2}{\lambda a}$$

$$Q_i = y_{i.} - \frac{1}{k}\sum_{j=1}^{b} n_{ij} y_{.j}$$

Where nij is the number of treatments i are in block j. There is also an adjustment for block totals, but for ANOVA you only need to adjust for treatment totals OR block totals, but NOT both. All formulae on Page 169-171.

Balanced Incomplete Block Design Example – Round Robin Hockey

Consider the results from a pool of teams at the 2010 World Juniors Hockey Tournament**.

Team	Game 1	Game 2	Game 3	Game 4	Game 5	Game 6	Game 7	Game 8	Game 9	Game 10
Sweden		7			2		6		6	
Canada	6			7		10			5	
Russia	3				0			8		8
C.R.*			2	2			3			3
Norway		1	0			1		2		

*Stands for Czech Republic

** 2010 IIHF World U20 Juniors, to be exact.

First, some definitions:
A sample unit is a Team`s Score
A treatment is a Team
A block is a Game

Next, find the design parameters (not to be confused with the distribution parameters):
Number of treatments (teams), a = 5
Number of blocks (games), b = 10
Size of a block (teams/game), k = 2
Blocks for each treatment (games/team), r = 4

There are N = ar = bk = 20 scores recorded.
Each pair of teams is in $\lambda = r(k-1)/(a-1) = 4(2-1)/(5-1) = 1$ game together.

Null hypothesis to test: All the team's mean scores are the same.
Solution: Do an ANOVA with adjusted treatment totals.

Put the data into R

```
team =
c("B_Canada","C_Russia","A_Sweden","E_Norway","D_CzechRepub","E_Norway","B_Canada
","D_CzechRepub","A_Sweden","C_Russia","B_Canada","E_Norway","A_Sweden","D_CzechR
epub","C_Russia","E_Norway","A_Sweden","B_Canada","C_Russia","D_CzechRepub")
goals = c(6,3,7,1,2,0,7,2,2,0,10,1,6,3,8,2,6,5,8,3)
game = as.factor(rep(1:10,each=2))
hockey = data.frame(game,team,goals)
```

Input the basics and design parameters

```
y = hockey$goals
trt = hockey$team
blk = hockey$game
k = 2
lambda = 1
a = 5
```

Get the SS.Total and SS.Blocks the usual way

```
SS.total = sum ( (y - mean(y))^2)

y.j = as.numeric(by(y,blk,sum))
SS.blocks = 1/k * sum(y.j^2) - sum(y)^2/length(y)
```

Get the adjusted means

```
yi. = as.numeric(by(y,trt,sum))
nij = matrix( c(0,1,0,0,1,0,1,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,0,1,
0,0,1,0,1,0,0,1,1,0,0,1,0,0,1,0,1,1,0,0,1,0,1,0,0), byrow=TRUE, nrow=5, ncol=10)

Qi = c(0,0,0,0,0)
for(i in 1:5)
{
     Qi[i] = yi.[i] - 1/k * sum( nij[i,]*y.j)
}
```

Get the adjusted SS treatment and SS error

```
SS.trt.adj = k*sum(Qi^2)/(lambda*a)
SS.error = SS.total - SS.blocks - SS.trt.adj
```

Perform an ANOVA

```
MS.trt = SS.trt.adj / 4   ## 5 teams, 4 df
df.error = 19 - 4 - 9 ## N = 20, so 19 total, - 4 for teams, - 9 for games
MS.error = SS.error / df.error
F = MS.trt / MS.error
1 - pf(F, df1=4, df2=6)
```