01NAEX - Lecture 04
Latin Squares,
Graeco-Latin Square Design,
Balanced Incomplete Block Design

Jiri Franc

Czech Technical University
Faculty of Nuclear Sciences and Physical Engineering
Department of Mathematics

The Latin Square Design

Last lesson:
Randomized complete block design (**RCBD**) as a tool to reduce the residual error by removing variability due to a known and controllable nuisance variable.

Hint:
If the nuisance source of variability is **known and controllable**, use **blocking** to eliminate its effect on the statistical comparisons among treatments.

Other types of designs that utilize the blocking principle:

- Latin square design
- Graeco-Latin square design
- Balanced Incomplete Block Design (BIBD)

Recap: ANOVA, and Assumptions

Model (one treatment, two orthogonal blocks):
$y_{ijk} = \mu + \tau_i + \rho_j + \kappa_k + \varepsilon_{ijk}$, $i = 1..p$ (treatments), j (rows), k (columns),
$\sum \tau_i = \sum \rho_j = \sum \kappa_k = 0$.

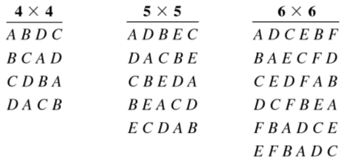
Source	df	SS	MS/Test
Treatments (τ)	$p-1$	SS_T	$MS_T = SS_T/(p-1)$
Rows (ρ)	$p-1$	SS_R	$MS_R = SS_R/(p-1)$
Columns (κ)	$p-1$	SS_C	$MS_C = SS_C/(p-1)$
Error	$(p-1)(p-2)$	SS_E	$MS_E = SS_E/[(p-1)(p-2)]$

Assumptions: additivity (no interactions), homoscedasticity, normality, independence within cells.

A The Latin Square Design Use-case: 1 treatment factor; 2 orthogonal nuisance sources (row/column)
Note: If interactions row$\times$treatment or col$\times$treatment are plausible, prefer a larger layout (Graeco-Latin or replicated squares).

The Latin Square Design

The $p \times p$ Latin square is a square containing p rows and p columns:



4×4	5×5	6×6
A B D C	*A D B E C*	*A D C E B F*
B C A D	*D A C B E*	*B A E C F D*
C D B A	*C B E D A*	*C E D F A B*
D A C B	*B E A C D*	*D C F B E A*
	E C D A B	*F B A D C E*
		E F B A D C

- Used to systematically handle two nuisance sources of variability
- Systematically blocks in two directions
- Each cell contains one of the p letters that correspond to the treatments
- Each letter occurs once and only once in each row and column

Note: The Latin square concept is also a mathematical basis for Sudoku puzzles, but with treatments instead of numbers.

Standard Latin Square

- A **Standard Latin Square** arranges the first row and column in alphabetical order.
- Each successive row is a left shift of the previous one.
- Analysis is nearly identical to randomized block designs but adapted for this specific format.

N: A single Latin square has no pure replication estimate error from within square deviations; replicate squares to stabilize MS_E.

Q: Would two smaller replicated squares be better than one big one for your process?

Q: Can I permute symbols, rows, columns independently?

Randomization & Construction

Randomization in practice

1. Structural randomization
   - Choose a standard $p \times p$ Latin square.
   - Randomly permute **treatment labels**, **rows**, and **columns** independently. It preserves orthogonality and balance.
2. Mapping
   - Map rows/cols to nuisance factors (e.g., day, operator) for logistics.
3. Run order randomization
   - If row = day: randomize day order, then randomize runs within day.
   - Otherwise: fully randomize all .

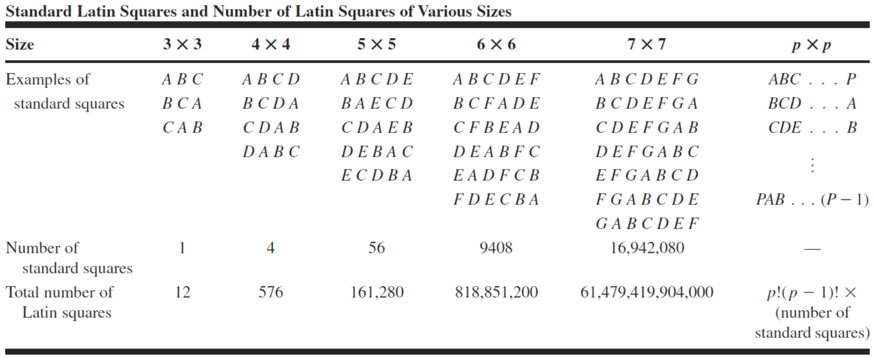
Why this works: orthogonality is preserved under permutations.

Q: Which mapping (row$\rightarrow$day, col$\rightarrow$operator vs row$\rightarrow$operator, col$\rightarrow$day) gives better logistics in your lab?

Number of Latin Squares

The number of distinct Latin squares increases exponentially with size *p*. For example:

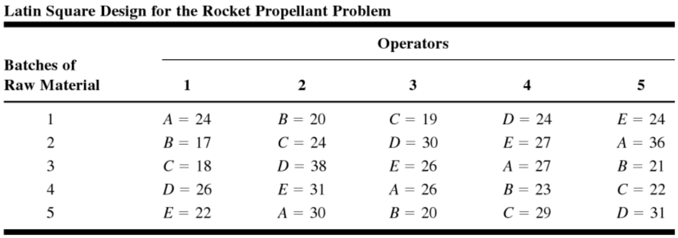
- There are 12 distinct 3×3 Latin squares.
- The number of 11×11 Latin squares is still unknown due to its complexity!



Standard Latin Squares and Number of Latin Squares of Various Sizes

Size	**3 × 3**	**4 × 4**	**5 × 5**	**6 × 6**	**7 × 7**	***p* × *p***
Examples of	*A B C*	*A B C D*	*A B C D E*	*A B C D E F*	*A B C D E F G*	*ABC . . . P*
standard squares	*B C A*	*B C D A*	*B A E C D*	*B C F A D E*	*B C D E F G A*	*BCD . . . A*
	C A B	*C D A B*	*C D A E B*	*C F B E A D*	*C D E F G A B*	*CDE . . . B*
		D A B C	*D E B A C*	*D E A B F C*	*D E F G A B C*	⋮
			E C D B A	*E A D F C B*	*E F G A B C D*	
				F D E C B A	*F G A B C D E*	*PAB . . .* $(P-1)$
					G A B C D E F	
Number of standard squares	1	4	56	9408	16,942,080	—
Total number of Latin squares	12	576	161,280	818,851,200	61,479,419,904,000	$p!(p-1)! \times$ (number of standard squares)

The Latin Square Design - Example



Latin Square Design for the Rocket Propellant Problem

Batches of Raw Material	**Operators**				
	1	**2**	**3**	**4**	**5**
1	$A = 24$	$B = 20$	$C = 19$	$D = 24$	$E = 24$
2	$B = 17$	$C = 24$	$D = 30$	$E = 27$	$A = 36$
3	$C = 18$	$D = 38$	$E = 26$	$A = 27$	$B = 21$
4	$D = 26$	$E = 31$	$A = 26$	$B = 23$	$C = 22$
5	$E = 22$	$A = 30$	$B = 20$	$C = 29$	$D = 31$

- A researcher studies the effects of five rocket propellant formulations on burn rate.
- Five batches of raw material are available, and each operator uses each formulation exactly once.

Statistical model for a simple Latin square ($\epsilon \sim \mathcal{N}(0, \sigma^2)$):

$$y_{ijk} = \mu + \alpha_i + \tau_j + \beta_k + \epsilon_{ijk} \begin{cases} i = 1, 2, \ldots, p \\ j = 1, 2, \ldots, p \\ k = 1, 2, \ldots, p \end{cases}$$

The Latin Square Design - ANOVA

In ANOVA, partition the total sum of squares into components for rows, columns, treatments, and error:

$$SS_{Total} = SS_{Rows} + SS_{Columns} + SS_{Treatments} + SS_{Error}$$

Python Code:

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

data = {’operator’: [’A’, ’B’, ’C’, ’D’, ’E’],
        ’batch’: [’1’, ’2’, ’3’, ’4’, ’5’],
        ’burn_rate’: [30, 40, 50, 20, 60],
        ’treatment’: [’T1’, ’T2’, ’T3’, ’T4’, ’T5’]}

model = ols(’burn_rate ~ C(operator) + C(batch) + C(treatment)
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

Blocking in the Latin Square Design

Blocking in two directions (rows and columns) helps reduce residual error. This method controls for two nuisance variables, such as time and location in an agricultural experiment, or operator and batch in industrial processes.

N: Always ensure that the blocking factors are independent and do not interact with treatments.

Sir Fisher Example:

- In an agricultural study, the Latin square can be used to test the effects of fertilizers (treatments) while blocking for both different fields and times of planting.

Replication in Latin Squares

Latin squares of small sizes provide relatively few degrees of freedom for error (3×3 has only 2, 4×4 only 6). It is often desirable to replicate them.

Three replication methods:

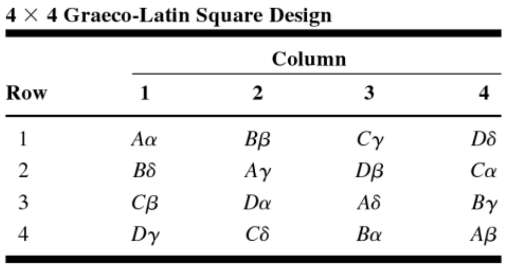
- **Case 1:** Use the same batches and operators in each replicate.
- **Case 2:** Use the same batches but different operators in each replicate (or vice versa).
- **Case 3:** Use different batches and different operators in each replicate.

N: The choice of replication method depends on the availability of resources and the variability between the blocks.

The Graeco-Latin Square Design

A Graeco-Latin square design superimposes two $p \times p$ Latin squares, one using Latin letters and the other using Greek letters:

- Each Greek letter appears exactly once with each Latin letter.
- Orthogonal designs allow control of three sources of variability.



4 × 4 Graeco-Latin Square Design

	Column			
Row	**1**	**2**	**3**	**4**
1	$A\alpha$	$B\beta$	$C\gamma$	$D\delta$
2	$B\delta$	$A\gamma$	$D\beta$	$C\alpha$
3	$C\beta$	$D\alpha$	$A\delta$	$B\gamma$
4	$D\gamma$	$C\delta$	$B\alpha$	$A\beta$

Fun Facts:

- Sir R. Fisher applied this design in agricultural experiments to control multiple nuisance variables.
- Euler referred to this as the "36 officers problem," an unsolved mathematical puzzle for many years.

Graeco-Latin Square

Idea: Superimpose two orthogonal Latin squares ($p \times p$) to test *two treatment factors* (Latin and Greek letters), while also blocking by *rows* and *columns*.

- Each Latin letter occurs once per row and column; same for each Greek letter.
- Each Latin-Greek pair occurs exactly once (orthogonality).

Existence: No Graeco-Latin squares for $p = 2$ or $p = 6$; they exist for all other p ($p \geq 3$, $p \neq 6$).

Usage: 2 treatment factors + 2 orthogonal nuisance sources; interactions among any of these assumed negligible.

Q: If you fear interaction between the two treatments, is Graeco-Latin still a good idea?

The Graeco-Latin Square Design - ANOVA

Python Code: ANOVA for Graeco-Latin design in Python:

```
import pandas as pd

data = pd.DataFrame({
    'operator': ['A', 'B', 'C', 'D', 'E'],
    'batch': ['1', '2', '3', '4', '5'],
    'assembly': ['X', 'Y', 'Z', 'W', 'V'],
    'burn_rate': [35, 45, 50, 30, 55],
    'treatment': ['T1', 'T2', 'T3', 'T4', 'T5']
})

model = ols('burn_rate ~ C(operator) + C(batch) + C(treatment)
                       + C(assembly)', data).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

Graeco-Latin Square - Considerations

- A Graeco-Latin square can block for three sources of variability (e.g., operator, batch, and test assembly).
- Orthogonality ensures no overlap between the blocking factors.
- It is important to account for the degrees of freedom lost by adding more blocking factors.

N: The third blocking factor in Graeco-Latin designs increases the complexity of the model, but often results in more accurate treatment estimates.

Balanced Incomplete Block Design (BIBD)

Balanced Incomplete Block Design (BIBD):

- Not all treatments can be run in each block.
- Use BIBD when treatment comparisons are equally important, and treatments appear together in a balanced manner.

For *a* treatments, *b* blocks, *k* treatments per block, and *r* replications:

$$\lambda = \frac{r(k-1)}{a-1}$$

N: BIBD is used in clinical trials to minimize the number of patients subjected to different treatments while maintaining comparability.

BIBD - Example (Python)

Python Code: Balanced Incomplete Block Design (BIBD):

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols

data = pd.DataFrame({
    'catalyst': ['C1', 'C2', 'C3', 'C4'],
    'batch': ['B1', 'B2', 'B3', 'B4'],
    'reaction_time': [120, 135, 140, 128]
})

model = ols('reaction_time ~ C(catalyst) +
                             C(batch)', data).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

Applications of BIBD in Real-World Scenarios

Clinical Trials:

- BIBD is useful when it's impossible to test all treatments on all subjects. For example, when patients are scarce or expensive to recruit, but there are many treatments to evaluate.

Agricultural Experiments:

- In crop testing, when not every combination of treatments can be applied to every plot, BIBD is an efficient way to compare different crop varieties or fertilizers.

N: BIBD helps manage logistical constraints while preserving the validity of statistical comparisons.

Today Exercise (1)

Problem 4.23 from D. C. Montgomery:
An industrial engineer investigates how four assembly methods affect assembly time. Four operators participate in a Latin square design. Analyze the data:

Order	Operator 1	Operator 2	Operator 3	Operator 4
1	C = 10	D = 14	A = 7	B = 8
2	B = 7	C = 18	D = 11	A = 8
3	A = 5	B = 10	C = 11	D = 9
4	D = 10	A = 10	B = 12	C = 14

Today Exercise (2)

Problem 4.40 from D. C. Montgomery:
An engineer studies mileage performance of five types of gasoline additives. The incomplete block design is used with five cars. Analyze the data:

Additive	**Car 1**	**Car 2**	**Car 3**	**Car 4**	**Car 5**
1	-	17	14	13	12
2	14	14	-	13	10
3	12	-	13	12	9
4	13	11	11	12	-
5	11	12	10	-	8

Today Exercise (3)

Problem 4.42 from D. C. Montgomery:
Seven hardwood concentrations are being studied to determine their effect on the strength of paper. Due to constraints, only three runs are allowed per day, resulting in a BIBD:

Concen. (%)	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
2	114	-	-	-	120	-	117
4	126	120	-	-	-	119	-
6	-	137	117	-	-	-	134
8	141	-	129	149	-	-	-
10	-	145	-	150	143	-	-
12	-	-	120	-	118	123	-
14	-	-	-	136	-	130	127

Analyze: Use ANOVA for BIBD and a linear model to evaluate concentration effects.

Conclusion

In this lecture, we covered:

- The Latin Square Design for two-way blocking
- The Graeco-Latin Square Design for three-way blocking
- The Balanced Incomplete Block Design (BIBD) for experiments with incomplete blocks

These designs help control variability in experiments and improve the precision of treatment effect estimates.