Assignment #4
Experimental Design for Ordinary Least Squares based Linear Regression

Responsible: Mats Gustafsson

Contents

1 GOALS

After completing this homework you are expected to:

1. Explain and apply 2-level full factorial and fractional factorial designs of experiments using Python code.
2. Perform and explain ordinary least (OLS) squares fitting in the context of modeling multivariate linear relationships (linear regression) on the form $y = \mathbf{w}^T\mathbf{z}$.
3. Explain the difference between the use of a fully quadratic models for a *Response Surface Design* compared to the use of purely linear models, or models containing only linear and interaction terms used for a *Screening Design*.
4. Explain and apply computer based D-optimal design of experiments (using Python code).
5. Interpret resulting models on the form $y = \mathbf{w}^T\mathbf{z}$ obtained after parameter fitting.
6. Simulate and explain the practical and theoretical problem that the D-Optimal design is created to solve (FOR GRADES 4/5)

2 PART A: Coffee Tasting Using Fractional Factorial Designs

The task here is to carry out an experimental design and associated analysis for a Coffe Taste Modeling task.

2.1 Problem Description

A coffee taste modeling task was performed to improve the taste of the coffee at work based on 5 factors (variables):

x_1 = Amount of Coffee (2.5 to 4.0 oz.) Note: Real value
x_2 = Grind size (8-10mm) Note: Real value
x_3 = Brew time (3.5 to 4.5 minutes) Note: Real value
x_4 = Grind Type (burr vs blade) Note: only two discrete alternatives
x_5 = Coffee beans (light vs dark)) Note: only two discrete alternative

The response y is the average of the scores (1-9). of a panel of 5 coffee drinkers:

y = average overall score limited to the interval $[1, 9]$.

2.2 Design Choice: Fractional Factorial Design

A full factorial experiment, meaning running all combinations of lows and highs, would take $2^5 = 32$ taste tests. Assume we can only do 3 runs per day (in order to avoid over-caffienate the testers), and also assume we can only do the tests on days when all 5 testers are present. That means the test will probably take two weeks or so. Alternatively, a 2^{5-1} fractional factorial experiment can be done to save resources, as this will reduce the design from $2^5 = 32$ to $2^4 = 16$ experiments.

Programmatic creation of the factional factorial 2^{5-1} design
Using the Python package `dexpy` one can build a 2^{5-1} design using `build_factorial`. First we must install the package:

```
pip install dexpy
```

Then the following code produces and print the desired design matrix as a pandas data frame.

```
import dexpy.factorial
import pandas as pd
coffee_design = dexpy.factorial.build_factorial(5, 2**(5-1))
coffee_design.columns = ['amount', 'grind_size', 'brew_time', 'grind_type', 'beans']
coffee_design
```

2.3 Adding Center Points to Estimate Curvature and Noise

When using a 2-level design, like the standard 2^{5-1} design above, it is not possible to have purely quadratic terms like x_3^2 in the linear model $\hat{y} = \mathbf{w}^T\mathbf{z}$ being fitted. The basic reason for this can be explain already in the one-dimensional case by the fact that if you want to fit the 3 parameters (w_0, w_1, w_2) in the model $\hat{y} = w_0 + w_1 x + w_2 x^2$, you need to have at least 3 training examples (x_n, y_n) including at least three different values of x_n. Thus one cannot fit a polynomial of degree 2 for any of the variables. Phrased differently, to determine the curvature (second order derivative) of a function $y(x)$ you need to have observations from at least three different values on the x-axis: With only two different x-values, the slope but not the curvature can be estimated.

Notably, in this application the variables x_4 and x_5 can only take two different values, whereas the other three can take any value on some limited intervals (see above). Therefore, it is only meaningful to have the 3 quadratic terms x_1^2, x_2^2 , and x_3^2. This means we need at least one extra level for the corresponding variables x_1, x_2 , and x_3 in order to estimate the coefficients for these quadratic terms. In addition to the two original values -1 and $+1$, the most natural single extra level to choose is the center value, here encoded as 0. In terms of Python code for the package `dexpy`, this can be specified as follows:

```
center_points = [[0, 0, 0, -1, -1],
                 [0, 0, 0, -1, 1],
                 [0, 0, 0, 1, -1],
                 [0, 0, 0, 1, 1]]
center_points
```

This means that four new experiments have will be added where x_1, x_2 , and x_3 are all at their center value (zero), while all four possible combinations of values for x_4 and x_5 are covered.

If we want to repeat these central point experiments for example twice, they can be added to the already created design matrix as follows:

```
import numpy as np
coffee_design = coffee_design.append(pd.DataFrame(center_points * 2, columns=coffee_design.columns))
coffee_design.index = np.arange(0, len(coffee_design))
coffee_design
```

Here the 2nd line of code just adds another column that contains integers specifying the identity for every experiment to be performed.

Remark.
In order to be able to estimate experimental noise, one must repeat experiments having the same settings multiple times: Since the same experiment (meaning the same value $\mathbf{x}(n)$) should in theory give exactly the same response value, one can study the resulting deviations in the response values to understand how much variability (uncertainty) there are in the response values. This possibility to estimate for example the standard deviation of this variability is not considered in this assignment.

2.4 Printing the Design: Coded as well as Using Actual Values

It is convenient to print out the design in terms of actual values, rather than the coded -1 and +1 values, to be use when actually making the coffee. This can be achieved as follows.

```
actual_lows = { 'amount' : 2.5, 'grind_size' : 8, 'brew_time': 3.5,
                'grind_type': 'burr', 'beans': 'light' }
actual_highs = { 'amount' : 4, 'grind_size' : 10, 'brew_time': 4.5,
                 'grind_type': 'blade', 'beans': 'dark' }
actual_design = dexpy.design.coded_to_actual(coffee_design, actual_lows, actual_highs)
actual_design
```

2.5 Task A1 - Determine and visualize design matrices

(i) Modify the code snippets above to create a design based on a pure 2^{5-2} fractional factorial design without any additional central points. How many experiments are created in this case?

(ii) If you have not already done it, just pull together the code snippets above to create and print out a design matrix based on combining a 2^{5-1} fractional factorial design with the following central points in duplicate:

```
center_points = [[0, 0, 0, -1, -1],
                 [0, 0, 0, -1, 1],
                 [0, 0, 0, 1, -1],
                 [0, 0, 0, 1, 1]]
center_points
```

As in the code snippets above, the matrix should be presented in two formats: The first matrix should be expressed in the coded values $\{-1, 0, +1\}$ and the second matrix should show the actual values to be given to the experimentalist assigned to perform the experiment.

2.6 Task A2 - Collect response values $y(n)$ and add them to your dataset

In a real coffee tasting project (actually performed and reported) the design matrix created in Task A1(ii) resulted in the 24 response (score) values $y(n)$ shown below. Run again your code for A1(ii) and make sure it produces 24 experiments. Then run the following code in order to insert the values as a new column in the Pandas dataframe `coffee_design`

```
coffee_design['taste_rating'] = [
    4.4, 2.6, 2.4, 8.6, 1.6, 2.8, 7.2, 3.4,
    6.8, 3.4, 3.8, 9.0, 5.2, 3.6, 8.2, 7.0,
    5.4, 6.8, 3.6, 5.4, 4.8, 6.2, 4.4, 5.8
]
coffee_design
```

2.7 Task A3 - Fit coefficients in a plain linear regression model using OLS

After completing the dataset collection, we are ready to fit the coefficient vector $\mathbf{w} = [w_0 \;\; w_1 \;\; ...]^T$ of model $\hat{y} = \mathbf{w}^T\mathbf{z}$ that we choose. For example we may use a plain linear model where $\mathbf{z}$ does not contain any interaction terms or quadratic terms. In other words $\hat{y} = \mathbf{w}^T\mathbf{z} = w_0 + w_1x_2 + ... + w_5x_5$ in this case so $\mathbf{z} = [1 \; x_1 \; ... \; x_5]^T$.

The fitting of the coefficient vector $\mathbf{w}$ is done by the ordinary least squares (OLS) method that minimizes the function

$$J_{OLS}(\mathbf{w}) = \sum_{n=1}^{N}(y(n) - \hat{y}(n))^2 = \sum_{n=1}^{N}(y(n) - \mathbf{w}^T\mathbf{z}(n))^2.$$

This means that OLS finds the coefficient vector $\mathbf{w}_{OLS}$ that minimizes the sum of squared prediction errors. Searching for the minimum by solving the system of equations $\frac{\partial J_{OLS}}{\partial \mathbf{w}} = \mathbf{0}$ gives the closed form solution

$$\mathbf{w}_{OLS} = (Z^TZ)^{-1}Z^T\mathbf{y}$$

where $\mathbf{y} = [y(1) \;\; y(2) \;\; ... \;\; y(N)]^T$ and where row $\mathbf{z}(n)^T$ of the matrix Z corresponds to experiment $\mathbf{x}(n)$. In this particular case of a purely linear model, this row can be written

$$\mathbf{z}(n)^T = [1 \;\; x_1(n) \;\; x_2(n) \;\; ... \;\; x_5(n)]^T$$

but it can easy be extended, for example with all possible interactions terms as in this simplified example that includes only 3 variables:

$$\mathbf{z}(n)^T = [1 \;\; x_1(n) \;\; x_2(n) \;\;\; x_3(n) \;\; x_1x_2(n) \;\; x_1x_3(n) \;\; x_2x_3(n)]^T$$

Note: In the examples here, the row $\mathbf{z}(n)^T$ contains the constant value one as a first element. This is to match the intercept coefficient w_0 in the model. If you want to have a model without any intercept term (meaning that you are assuming a linear model that passes through the origin), then this constant values should be removed from $\mathbf{z}(n)^T$.

Use the following code to determine the coefficient vector $\mathbf{w}$ in the model $\hat{y} = \mathbf{w}^T\mathbf{z}$ for the special case of a plain linear model where $\mathbf{z}$ does not contain any interaction terms or quadratic terms. In other words $\hat{y} = \mathbf{w}^T\mathbf{z} = w_0 + w_1x_2 + ... + w_5x_5$ in this case so $\mathbf{z} = [1 \; x_1 \; ... \; x_5]^T$.

```
coffee_design_as_np_array=coffee_design.to_numpy() #Convert from pandas to numpy array
y_np_array=coffee_design_as_np_array[:,5]
y_np_array=y_np_array.reshape(-1,1)  #Reshape to column 2D numpy array
X_np_array=coffee_design_as_np_array[:,0:5] #Examples as rows

#LEAST SQUARES CLOSED FORM SOLUTION w=inv(Z'*Z)*Z'*y
#Extend X_np_array with one extra column for the intercept
shapeX=X_np_array.shape
nrows=shapeX[0]
Z = np.c_[np.ones((nrows, 1)), X_np_array] # add x_0 = 1 to each instance (row)
                                           # to include an intercept term w_0 in the model
#Calculate w=inv(Z'*Z')*Z'*y using np.transpose() and np.matmul() and np.linalg.inv()
Ztranspose=np.transpose(Z)
ZtransposeZ=np.matmul(Ztranspose,Z)
ZtransposeZ_inv=np.linalg.inv(ZtransposeZ)
w_np_array=np.matmul(ZtransposeZ_inv,np.matmul(Ztranspose,y_np_array)) #column vector
print(w_np_array)
```

Tip: You should expect to get the following result

```
[[ 5.1       ]
 [ 0.875     ]
 [-0.125     ]
 [ 1.2       ]
 [-0.13333333]
 [ 0.45      ]]
```

2.8 Task A4 - Interpretation of the fitted coefficients

Based on the magnitudes (ignore their signs) of the estimated coefficients obtained in A2 above:
(i) What is your conclusion regarding which of the five variables x_i seem most influential on the response (score) value y?
(ii) Why are the magnitudes of the estimated coefficients directly comparable when using the coded values $\{-1, 0, +1\}$? Why is this not the case if instead the actual values are used (which are of different types eg. oz, mm, hours, and therefore on different scales) to estimate the model coefficients?

2.9 Task A5 - Refined analysis using interaction terms

Repeat Tasks A2 and A3 after extending the model to a model that includes interaction terms but no pure quadratic terms. Confirm that such a model has 16 parameters (coefficients).

Tip: The following lines of Python code should give you an idea how to include all the interaction terms in the model. Notably, here only the inclusion of two of the desired terms are shown. Thus on lines 2 and 3 in this code snippet, a new column is introduced in the matrix `Z_np_array` that corresponds to a particular interaction term.

```
1 Z_np_array = np.c_[np.ones((nrows, 1)), X_np_array] # add x0 = 1 to each instance (row)
2 Z_np_array= np.c_[Z_np_array, X_np_array[:,0]*X_np_array[:,1]]
3 Z_np_array= np.c_[Z_np_array, X_np_array[:,0]*X_np_array[:,2]]
```

Questions:
(i) What interaction terms are most influential according to the model fitted here? Is there any variable x_i where the corresponding weight w_i has one sign while at the same time the corresponding interaction weight w_{ij} has the opposite sign?
(ii) Explain why the inclusion of the interaction terms may create difficulties understanding the actual net effect of for any variable x_i on the response y. Tip: Consider the the sum $w_i x_i + w_{ij} x_i x_j$ and its partial derivative with respect to input x_i:

$$\frac{\partial\, w_i x_i + w_{ij} x_i x_j}{\partial x_i} = w_i + w_{ij} x_j$$

Apparently the change in the response y when x_i is slightly increased is challenging to express in simple words. This is because the influence of variable x_i on the response y is context dependent. For example when $x_j \approx 0$, then the term $w_{ij} x_i x_j$ can be ignored even if w_{ij} is far from zero. In other words, the influence of x_i on y depends partly on the values of other variables like x_j.

3 PART B: D-Optimal Experimental Design

The basic result from the theory of D-optimal design is as follows: In order to reduce the variability in the OLS estimate $\mathbf{w}_{\text{OLS}}$ of the coefficient vector for a pre-defined number of experiments N, the experiments should be chosen so that the model matrix Z employed should maximize the determinant $|Z^T Z|$. For example, row n in the matrix Z equals

$$\mathbf{z}(n)^T = [1 \;\; x_1(n) \;\; x_2(n) \;\; x_1(n)x_2(n) \;\; (x_1(n))^2 \;\; (x_2(n))^2]^T$$

in the special case we are considering a full quadratic response model (for more details see below).

3.1 A First Example - Chemical Process Analysis

Here follows a first example from Chemical Process Analysis, showing how D-optimal experimental design can be used in practice.

3.1.1 Problem Description

Assume you are trying to improve the yield of a chemical process by adjusting the following two input variables.

x_1 = Reaction time (40-50 minutes) Note: Real value
x_2 = Temperature (80-90 °C) Note: Real value

3.1.2 Design Choice: D-Optimal Design

Assume that due to the great expense of each experiment, in total only 6 experiments can be performed. Therefore you choose to employ a D-optimal design in order to fit a fully quadratic model.

3.1.3 Creating a D-Optimal Design in Python

The required D-optimal design can easily be created in Python using `build_optimal` in the package `dexpy`. In this case a full quadratic model looks as follows: $y = w_0 + w_1x_1 + w_2x_2 + w_{12}x_1x_2 + w_{11}x_1^2 + w_{12}x_1^2$. Thus the number of parameters to tune is actually equal to six. The minimum number of experiments to perform is therefore six and to make a design consisting of six experiments, we specify that by the parameter specification `run_count=6` as shown in the code snippet below. Here follows a complete code snippet that creates and then presents the desired design.

```
import dexpy.optimal
from dexpy.model import ModelOrder


#D-optimal design of quadratic model
no_of_factors=2
reaction_design = dexpy.optimal.build_optimal(no_of_factors, run_count=6, order=ModelOrder.quadratic)

#Print out of the design and conversion to real experimental values to use in the actual experiments
    to be performed
column_names = ['time', 'temp']
actual_lows = { 'time': 40, 'temp': 80 }
actual_highs = { 'time': 50, 'temp': 90 }
reaction_design.columns = column_names
print(dexpy.design.coded_to_actual(reaction_design, actual_lows, actual_highs))
```

3.1.4 Graphical visualization of the D-Optimal Design in Python

As you can see, the runs are now arranged in a less structured manner compared to the standard designs shown before. One can use the Python code provided below to understand graphically how the points are spread out throughout the space.

```
import matplotlib as plt
reaction_design_np_array=reaction_design.to_numpy()
fig = plt.pyplot.figure()
ax = fig.add_subplot()

# plotting the points
plt.pyplot.scatter(reaction_design_np_array[:,0], reaction_design_np_array[:,1],color='green', marker
    ='o')
plt.pyplot.xlabel('time')
plt.pyplot.ylabel('temp')
ax.set_xticks([-1, 0, 1])
ax.set_xticklabels(['40 min', '45 min', '50 min'])
ax.set_yticks([-1, 0, 1])
ax.set_yticklabels(['80C', '85C', '90C'])
ax.grid()
plt.pyplot.title('Chemical Process Data')
plt.pyplot.show()
```

3.1.5 Task B1 - D-optimality is about maximizing $|Z^T Z|$

We can calculate the D-optimality of this design, which is the determinant of the matrix $|Z^T Z|$, where the row n in Z equals $[1 \;\; x_1(n) \;\; x_2(n) \;\; (x_1(n))^2 \;\; (x_2(n))^2 \;\; x_1(n)x_2(n)]$. This can be done based on the following code snippet.

```
import numpy as np
X_np_array=reaction_design_np_array

#Add columns corresponding to a full quadratic model to the design matrix Z_np_array
shapeX=X_np_array.shape
nrows=shapeX[0]
Z_np_array = np.c_[np.ones((nrows, 1)), X_np_array] # add x0 = 1 to each instance (row)
Z_np_array = np.c_[Z_np_array, X_np_array[:,0]*X_np_array[:,0]]
Z_np_array = np.c_[Z_np_array, X_np_array[:,1]*X_np_array[:,1]]
Z_np_array= np.c_[Z_np_array, X_np_array[:,0]*X_np_array[:,1]]
Z_np_array
det_Z_np_array = np.linalg.det(np.matmul(np.transpose(Z_np_array),Z_np_array))
det_Z_np_array
print("|(Z'Z)| for optimal design:",det_Z_np_array)
```

Use the code snippets above to solve the following subtasks.

(i) Create D-Optimal Design for a full quadratic model for six experiments when the number of variables (factors) are $d = 2$.

(ii) Visualize the design made in (i) in terms of both a Table/Matrix and a graph that illustrate the experiments selected to be performed.

(iii) Calculate the maximally large determinant $|Z^T Z|$ where each row in the matrix Z corresponds to one of the 6 experiments selected. Tip: You are expected to get a value for this determinant close to 264.6 (sometimes lower).

(iv) Create and visualize a D-Optimal Design for a full quadratic model when the number of experiments are increased from $N = 6$ to $N = 10$. This gives you a good visual overview how the designed (selected) experiments are spread out.

(v) Check if the design made in (iv) contains any pair of experiments that are identical, meaning that the same experiment should be performed twice.

3.2 A Second Example - Paper Helicopter Design

An illustrative application example of experimental design for OLS is the search for the optimal paper helicopter design as described by the article by Erik Barry Erhardt (see your course literture).

3.2.1 Task B2 - OLS fitting of Paper Helicopter Data

Inspired by the Paper Helicopter article by Erik Barry Erhardt, perform OLS fitting to determine the coefficient of the quadratic Surface Response Model

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_{12} x_1 x_2 + w_{11} x_1^2 + w_{12} x_1^2$$

using the following paper helicopter flying data (where y denotes the flying time)

```
import numpy as np
X=[[-1,-1],[1,-1],[-1,1],[1,1],[0,0],[0,0],[0,0],[1,0],[-1,0],[0,1],[0,-1]]
y=[13.65,13.74,15.48,13.53,17.38,16.35,16.41,12.51,15.17,14.86,11.85]
X=np.asarray(X)
y=np.asarray(y).reshape(-1,1)
print(X)
print(y)
```

Tip: You should get the coefficient vector $\mathbf{w} = [w_0\ w_1\ w_2\ w_{12}\ w_{11}\ w_{22}]^T$, with its elements listed below.

```
[[15.95157895]
 [-0.75333333]
 [ 0.77166667]
 [-0.51      ]
 [-0.96894737]
 [-1.45394737]])

```

Remark.
After this fitting, you now have access to a quadratic mathematical model that can predict the flying time for any paper helicopter design (within reasonable limits)! As shown in the article by Erik Barry Erhardt, this model can be analyzed mathematically/numerically to find out the most promising helicopter design that gives the longest predicted flying time. In the next task below, we will instead assume the quadratic mathematical model we have created can be used to accurately simulate the flying time for different paper helicopters (instead of actually making physical helicopters and run the physical flying experiment).

3.2.2 Task B3 - Fit Simulated Paper Helicopter Data Using a D-Optimal Design

As explained in steps 1-3 below, use the coefficient vector $\mathbf{w}_{\text{simulator}}$ found in B2 above to simulate the flying time of new helicopters. Use this approach to study the potency of D-Optimal design. Assume that we only have the time budget to design N=6 paper helicopters. Then it is not possible to use a Central Composite Design, as such a design would need at least $2^2+4=8$ designs. Then instead perform the following steps.

1. Make a D-optimal design that consists of only $N = 6$ experiments $\mathbf{x}(n)$ for a two-factor quadratic model.

2. Use the weight vector found in B2, here denoted

$$\mathbf{w}_{\text{simulator}} = [w_0 \;\; w_1 \;\; w_2 \;\; w_{12} \;\; w_{11} \;\; w_{22}]^T,$$

to simulate the flying time of the 6 new paper helicopters designed in Step 1 by computing

$$y(n) = w_0 + w_1 x_1(n) + w_2 x_2(n) + w_{12} x_1(n) x_2(n) + w_{11}(x_1(n))^2 + w_{22}(x_2(n))^2 + \epsilon(n).$$

Here $\epsilon(n)$ is a random number drawn from a zero mean normal distribution with standard deviation $\sigma = 0.1$. Thus, this should result in 6 training examples $(\mathbf{x}(n), y(n))$, n=1,2,..,6.

3. Use the training data set $(\mathbf{x}(n), y(n))$ generated in Step 2 to fit the parameter vector $\mathbf{v} = [v_0 \;\; v_1 \;\; v_2 \;\; v_{12} \;\; v_{11} \;\; v_{22}]^T$ of the quadratic model $y = v_0 + v_1 x_1 + v_2 x_2 + v_{12} x_1 x_2 + v_{11} x_1^2 + v_{22} x_2^2$ using the OLS method. In the following, name the resulting fitted parameter vector as $\mathbf{v}_{\text{D-optimal}}$.

4. Compare how well the resulting fitted coefficient vector $\mathbf{w}_{\text{D-optimal}}$ agrees with the "true" parameter vector $\mathbf{w}_{\text{simulator}}$ that was used to generate the training data.

4 PART C*: D-OPTIMAL DESIGN THEORY (GRADES 4/5)

NOTE: FOR GRADES 4/5 ONLY

Experimental design for OLS based linear regression is about creating diversity among the training examples so that each of them provide new information, rather than being redundant. Making the experiments as different as possible is not only important from an efficiency perspective as we want to make as few experiments as possible. It is in fact necessary to avoid the OLS method to collapse in the following sense: If the experiments $\mathbf{x}(n)$ are sufficiently similar in the sense that the the elements of these vectors are strongly correlated, then the OLS method collapses in the following sense.

4.1 Theory

Assume that the training examples $(\mathbf{z}(n), y(n))$ being analyzed comes from the model $y(n) = \mathbf{w}_{\text{true}}^T \mathbf{z}(n) + \epsilon(n)$ where $\epsilon(n)$ is a zero mean random measurement noise with standard deviation σ_y. Then one can show theoretically that the OLS solution

$$\mathbf{w}_{OLS} = (Z^T Z)^{-1} Z^T \mathbf{y}$$

is unbiased in the sense that

$$E\{\mathbf{w}_{OLS}\} = \mathbf{w}_{\text{true}}.$$

This means that if one repeats the same data collection B times, using the same experiments $\mathbf{z}(n)$ but getting different observed response values $y_b(n)$ for each data collection b, then the average $< \mathbf{w}_{OLS}(b) >_b$ of the resulting estimates $\mathbf{w}_{OLS}(b)$ will equal $\mathbf{w}_{\text{true}}$ provided that B is large enough:

$$< \mathbf{w}_{OLS}(b) >_b = \frac{1}{B} \sum_{b=1}^{B} \mathbf{w}_{OLS}(b) \approx \mathbf{w}_{\text{true}}.$$

Although this is desired and expected, one can also show that the covariance matrix for the OLS solutions satisfy

$$\frac{1}{B} \sum_{b=1}^{B} (\mathbf{w}_{OLS}(b) - < \mathbf{w}_{OLS}(b) >_b)(\mathbf{w}_{OLS}(b) - < \mathbf{w}_{OLS}(b) >_b)^T \approx \sigma_y^2 (Z^T Z)^{-1}$$

that for infinitely large values of B can be written

$$Cov\{\mathbf{w}_{OLS}\} = \sigma_y^2 (Z^T Z)^{-1}$$

From this we can conclude that if the matrix $Z^T Z$ is close to singular, then the covariance matrix of $\mathbf{w}_{OLS}$ will contain very large elements, and this means that the element of $\mathbf{w}_{OLS}$ fluctuate violently. As will be demonstrated via computer simulations below, in practice this means:

Conclusion 1:

Although the OLS estimate $\mathbf{w}_{OLS}$ agrees with the truth on average (it is unbiased), it may vary violently of the matrix $Z^T Z$ is close to singular!

This conclusion can be used to motivate the D-optimal design: In order to make $Z^T Z$ far from singular, we should make the smallest eigenvalue λ_{min} of this matrix as large as possible. Notably, per definition the determinant $|Z^T Z|$ of the matrix $Z^T Z$ is the product of its eigenvalues λ_i:

$$|Z^T Z| = \prod_{i=1}^{d} \lambda_i.$$

Therefore, one indirect way of ensuring that the smallest eigenvalue is large is to choose the experiments in Z so that $|Z^T Z|$ is maximized. Using this method to avoid that $Z^T Z$ is close to singular is known as the D-optimal design method.

Conclusion 2: D-Optimal Design

In D-Optimal Design, one chooses experiments so that the determinant $|Z^T Z|$ is maximized, thereby trying to avoid getting the matrix $Z^T Z$ close to singular.

In the following the practical consequences of having correlated vectors $\mathbf{x}(n)$ will be considered via computer simulations.

4.2 Task C1* - Simulation study

Step 1: Generate N random but correlated input examples $\mathbf{x}(n)$ with dimension $d = 10$.
(i) Generate two random vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ having $d = 10$ elements each. Choose each element uniformly from the interval $[-10, 10]$.

(ii) Generate $N = 200$ examples according to $\mathbf{x}(n) = \mathbf{b}_1 \cdot t_1(n) + \mathbf{b}_2 \cdot t_2(n) + \sigma_x \mathbf{e}(n)$ where $\sigma_x = 0.001$ and $t_1(n)$ and $t_2(n)$ as well as all the elements of the vector $\mathbf{e}(n)$ are different independent random numbers drawn from a standard zero mean unit variance normal distribution $N(\mu = 0, \sigma = 1)$. IMPORTANT: For each $\mathbf{x}(n)$ generated, a new random vector $\mathbf{e}(n)$ must be generated, as well as two new random values $t_1(n)$ and $t_2(n)$.

(iii) Confirm that many of the variables in $\mathbf{x}(n)$ are strongly correlated by determining and printing the 10×10 dimensional correlation matrix. Note: Some of the variables will not be strongly correlated, whereas others will be.

Step 2: Generate training examples $(\mathbf{x}(n), y(n))$ using the linear model $y(n) = \mathbf{w}_{true}^T \mathbf{x}(n) + \epsilon_y(n)$
(i) Generate one d-dimensional weight vector $\mathbf{w}_{\text{true}}$ by choosing each element uniformly from the interval $[-3, 3]$.

(ii) Use $\mathbf{w}_{\text{true}}$ together with the generated vectors $\mathbf{x}(n)$ from above to create training pairs according to $y(n) = \mathbf{w}_{\text{true}}^T \mathbf{x}(n) + \sigma_y e_y(n)$ where $e_y(n)$ is a random number drawn from the standard normal $N(\mu = 0, \sigma = 1)$, $n = 1, 2, ..., N$ and $\sigma_y = 0.1$. Note: The model here does not have any intercept term as the first element of the vector $\mathbf{x}(n)$ is not equal to one (a constant).

Step 3: Estimate the unknown $\mathbf{w}_{true}$ using the training data
Use the N training examples $(\mathbf{x}(n), y(n))$ to estimate $\mathbf{w}_{\text{true}}$ as the OLS solution $\mathbf{w}_{\text{OLS}} = (X^T X)^{-1} X^T \mathbf{y}$. Here row n in the matrix X equals $[\mathbf{x}_1(n)\ \mathbf{x}_2(n)\ ...\ \mathbf{x}_d(n)]^T$ and $\mathbf{y} = [y(1)\ y(2)\ ...\ y(N)]^T$.

Step 4: Repeat OLS fitting many times then present the results graphically
(i) Perform the above fitting at least $B = 10^4$ times, each time using a new set of response values generated as $y(n) = \mathbf{w}_{\text{true}}^T \mathbf{x}(n) + \sigma_y e_y(n)$. You should use exactly the same vectors $\mathbf{x}(n)$ all the time, but $N = 200$ new response values $y(n)$ should be created by simulating $N = 200$ new noise values $e_y(n)$. This means you will simulate a case where you repeat the same set of $N = 200$ experiments every day for $B = 10^4$ days, and for every day b you obtain a new noisy response vector $\mathbf{y}_b$ and therefore a corresponding new OLS estimate $\mathbf{w}_{\text{OLS}}(b) = (X^T X)^{-1} X^T \mathbf{y}_b$. NOTE: Here the same vectors $\mathbf{x}(n)$ that were generated in Step 2 are used in all the $B = 10^4$ runs. Therefore, the only change between these different runs will be the simulated random noise terms $e_y(n)$.

(ii) Plot the average vector $< \mathbf{w}_{\text{OLS}}(b) >_b = \frac{1}{B} \sum_{b=1}^{B} \mathbf{w}_{\text{OLS}}(b)$ in blue color and the true vector $\mathbf{w}_{\text{true}}$ in red color, both in the same graph for comparison. You should expect to get results similar the ones in the left part of Fig. 1.

(iii) Plot the $B = 10^4$ vectors $\mathbf{w}_{\text{OLS}}(b)$ obtained from (i) in the same graph using some form of dotted lines in blue to make them show weakly. Add to the same graph also the true vector $\mathbf{w}_{\text{true}}$ as a red solid line. You should expect to get results similar to those shown in the right part of Fig. 1.

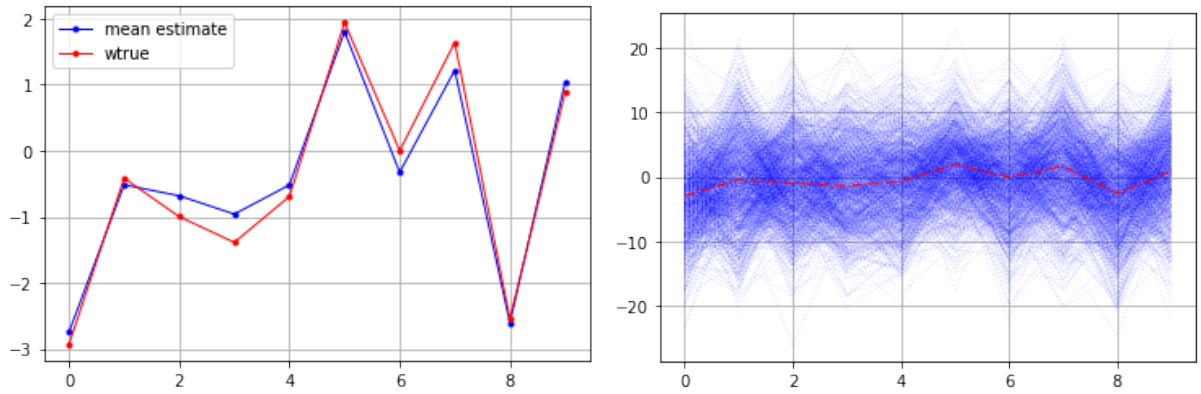

Figure 1: The foundation of D-Optimal design - results after repeating the OLS fitting $B = 10^4$ times for a case where the elements of the experimental vectors $\mathbf{x}(n)$ are strongly correlated. *LEFT:* The mean vector $< \mathbf{w}_{\text{OLS}}(b) >_b$ (blue) and the true vector $\mathbf{w}_{\text{true}}$ (red). This empirically shows the fact that the OLS estimates $\mathbf{w}_{\text{OLS}}(b)$ are unbiased. *RIGHT:* The individual OLS estimates $\mathbf{w}_{\text{OLS}}(b) >_b$ (blue) and the true vector $\mathbf{w}_{\text{true}}$ (red). Notably, the red curve in the right figure is exactly the same here as in the left figure, but it looks different due to another scale on the y-axis! This empirically shows the fact that the OLS estimates $\mathbf{w}_{\text{OLS}}(b)$ may vary violently in cases where the elements of the experimental vectors $\mathbf{x}(n)$ are strongly correlated so that the matrix $Z^T Z$ is close to singular.

4.3 Task C2* - Questions to answer

Based on your results in Task C1, please provide answers to the following questions.
(1) Does the average vector $< \mathbf{w}_{\text{OLS}}(b) >_b = \frac{1}{B} \sum_{b=1}^{B} \mathbf{w}_{\text{OLS}}(b)$ agree well with $\mathbf{w}_{\text{true}}$, as expected?
(2) Can you confirm that most of the individual fits $\mathbf{w}_{\text{OLS}}(b)$ do not agree at all with $\mathbf{w}_{\text{true}}$?
(3) Make sure you agree that OLS based linear regression is not practically useful when elements of the input vector $\mathbf{x}(n)$ are correlated as in the case studied here?

5 EXAMINATION

The examination of this exercise will be in the form of:

1. Uploading and getting approved your notebooks+results to the course home page before the scheduled mandatory seminar. You should **submit two different notebooks**, one for grade 3 tasks and one for grade 4/5 tasks, respectively.
2. Presenting and discussing your results orally during one already scheduled mandatory seminar. **During this mandatory seminar, you need to have access to a computer so that you are able to share your results with others in the seminar via a course specific online meeting**. The permanent link to this online meeting is available on the course home page. If you are not able to attend it, contact the main teacher in order to book a separate seminar with him/her.
3. Questions about these exercises and the associated theory and Python code during the final written exam.

Enjoy and Good Luck!