Authors: Behdad Sadeghian, Josip Dubravac, Prof. Dr. Pierre Jousset	**Last update:** 20.12.2023

Fundamentals

Introduction

Artificial Intelligence (AI) is defined as the capability of computer systems to perform tasks that normally require human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages.
Machine learning (ML) is a part of AI and describes the use and development of computer systems that are capable to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data.
As an illustration of everyday life, ML can suggest favorite movies on Netflix and helps doctors to provide a diagnosis to a disease. It can observe and analyses input of individuals on social media to guide and influence their future choices and decisions.

A task which is suitable for machine learning must have a set of inputs and a set of outputs that can be measured accurately for the machine to learn the mapping between the two sets.

Consequently, ML requires as start point data such as numbers, text, time series or measures from sensors. The provided database is gathered to be used as training data, which is the data to be used to train an algorithm or ML model to predict the outcome. Theoretically, the more data, the better the ML prediction.

A ML system can have three main different functions:

- Descriptive: The system uses the input data to understand data.
- Predictive: The system uses the data to predict what is going to happen.
- Prescriptive: The system uses the data to make suggestions about what action to take.

There are three main types of machine learning algorithm:

1. Supervised ML algorithms: This type of ML require accurately labeled data. They are used to either categorize or predict a continuous outcome. The former is called classification and the earlier regression.
2. Unsupervised ML algorithms: The algorithm is used to understand unlabeled or unknown datasets. It can find patterns or trends that people are not explicitly looking for. This is called clustering.
3. Reinforcement ML algorithms: The ML algorithm can make decisions on its approach to a problem or process itself by establishing a reward system. The Algorithm can recognize the right decisions.

The usage of these algorithms is summarized in the following decision flowchart.

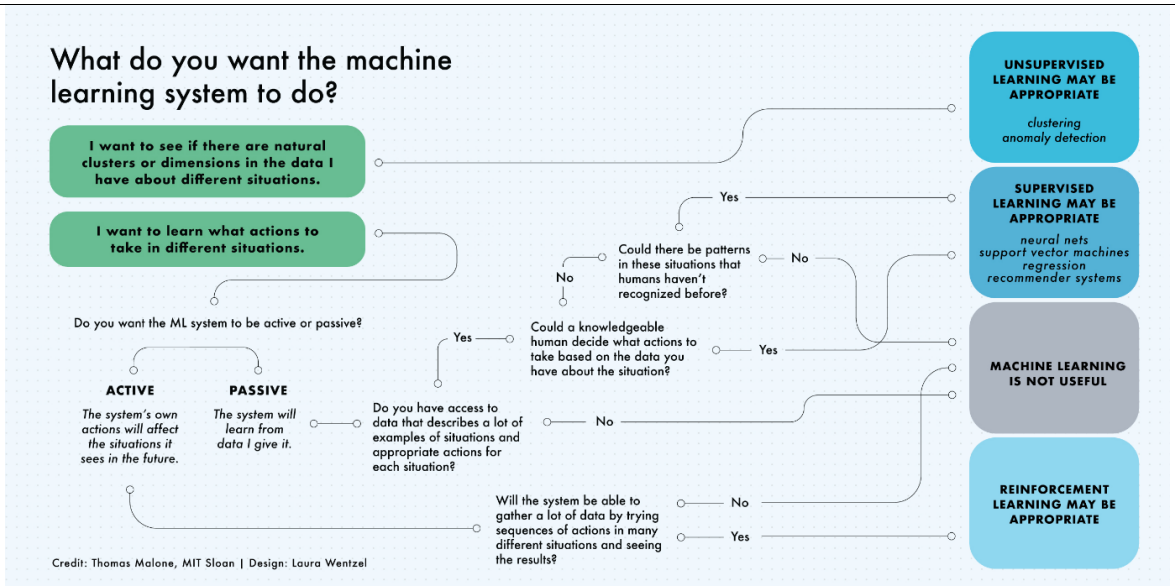


Figure 1: practical guideline on choosing the best ML algorithm. Source: [Machine learning, explained | MIT Sloan]

The choice of the right algorithm requires experience and must sometimes be made pragmatically with trials and errors.

For the following only supervised learning algorithms will be considered. This group of ML can be divided into classification and regression models.

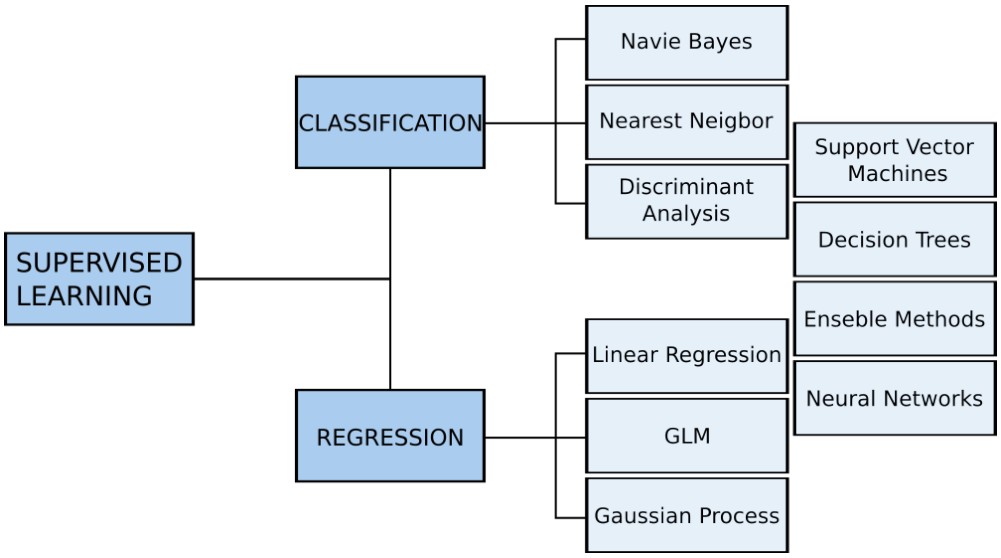


Figure 2: Categories of supervised learning. Source: [Types of Machine Learning Models Explained - MATLAB & Simulink (mathworks.com)]

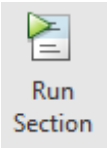
This tutorial will focus on one of the simplest ML model available: the linear regression model, belonging to the supervised learning category.

Part 1

Start of the tutorial.

The following tutorial used a Matlab code illustrating the use of ML supervised learning algorithms step by step.

As soon as a section of the Matlab code has been read, click on the button "run Section" below:



Try it!

```
% Create an anonymous function for laughter
laugh = @(n) repmat('ha', 1, n);


% Generate a funny greeting
greeting = ['Hello, World! ', laugh(randi([2, 7])), '!'];


% Display the funny greeting
disp(greeting)
```

```
Hello, World! hahahahahaha!
```

Example of linear regression based on rental prices in Zürich

The goal of this section is to model the rental prices in Zürich using a linear regression model to make the student more familiar with it.

For this purpose, an algorithm will be used. This algorithm contains a mathematical function providing a forecast of the rental prices in Zürich as a function of input variables. These variables can for example be the area of an apartment or its number of rooms. One of the simplest functions is a linear combination of the parameters x_i:

$$h(\theta) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$$

Where *h(θ)* is the hypothesis or mathematical function describing the price of the apartment, θ_0 to θ_n are coefficients to be determined and x_1 to x_n are known input variables. The aim is to identify the appropriate coefficients θ_0 to θ_n to predict $h(\theta)$ as a function of the input variables (x_1 to x_n) accurately. Considering a simple function including only the two variables "apartment area" and "number of bedrooms", the function take the form:

$$Price = \theta_0 + \theta_1(\#bedrooms) + \theta_2(Area)$$

where the coefficients θ_0, θ_1 and θ_2 must be identified.

Cost function

In order to measure the accuracy provided by the prediction of an ML model, a criterion measuring this accuracy is required: the cost function.

The cost function measures the gap between real measurements and the computed response provided by the ML Model. The goal during training is to minimize this difference. It quantifies the error between computed/predicted values of the ML model and the real measurements. Considering the example of a force-displacement curve measured in a laboratory and characterizing the mechanical behavior of a dog-bone specimen and assuming it exists a model to predict this behavior, the cost function can compute for each displacement the difference between the measured and the computed response. At the end, the cost function provides the sum (a scalar) of these differences for all points of the curve, indicating the global difference between the two curves. To increase the predictability of the ML model, the cost function must be minimized. Different forms of cost functions are available in the literature, which depend on the problem to minimize. Additionally, different optimization methods are available to minimize the cost function. Two of these methods are detailed in the following chapters:

1. The gradient descent method
2. The normal equation method

Method 1: Gradient descent Method

The gradient descent (also called steepest descent method) is an optimization method for finding a local minimum of a differentiable function. In the previous rental-prices example the gradient descent can be used to find the values of coefficients θ_0, θ_1 and θ_2 that minimize the cost function, that is the gap between the function $h(\theta)$ and the real measurements.

The main steps behind gradient descent are:

1. Start with an initial set of variables: x_1 (number of bedrooms) and x_2 (Area)
2. Compute the gradient of the cost function with respect to each variable x_1 and x_2
3. Update the parameters in the direction of the negative gradient.

Each step will be considered individual in the following example:

1. Initial parameter set

To keep the problem simple the function "price" will be considered to depend on one variable "Area" only:

$$\text{Price} = \theta_0 + \theta_1(\text{Area})$$

The data "Price" and "Area" are gathered from a real estate website:

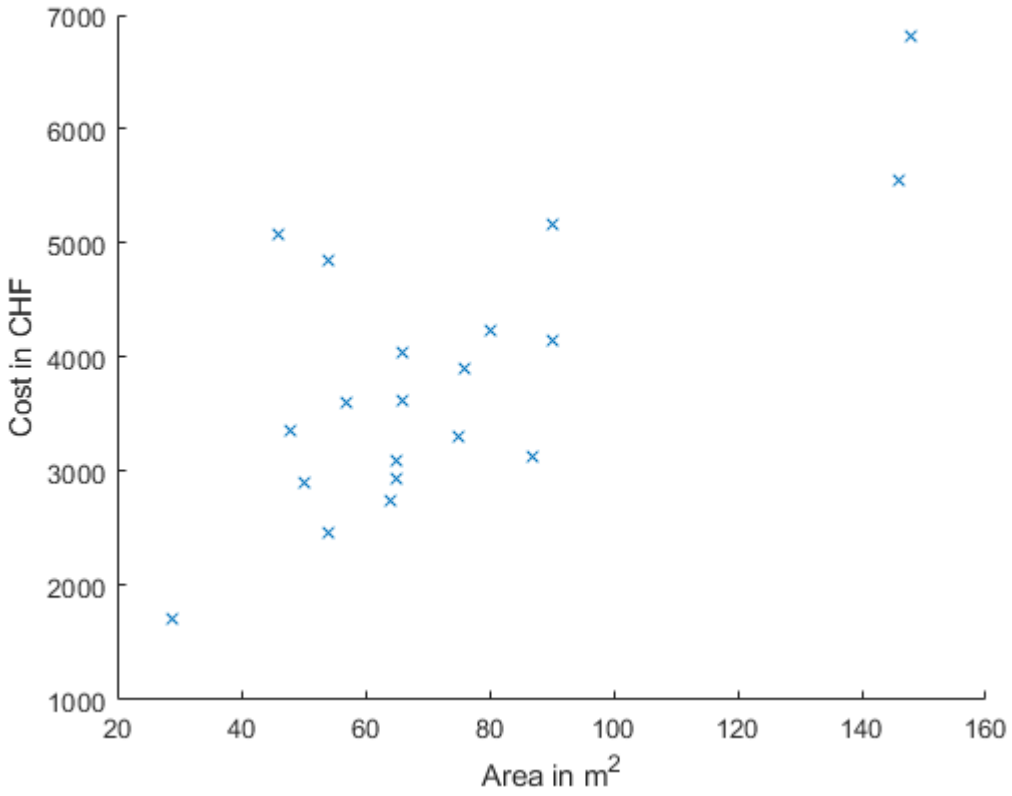
```
clear ITBO_interactive_script; close all; clc; clear all;
```

```
T1 = readtable('Zurich_area_prices.xlsx','VariableNamingRule','preserve');
```

An overview of the different couples (Area, Price) is represented in the following plot.

Question: Is it possible to already identify a mathematical function providing a trend of the distribution of these couples?

```
scatter(T1.("Area [m^2]"),T1.("Cost [CHF]"),'x');
xlabel('Area in m^2');
ylabel('Cost in CHF');
```



2. Compute the gradient of the cost function

A suitable cost function must be defined for this problem. The Mean Square Error (MSE) cost function *J(θ)* will be suitable here:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

- m is the number of measured couples, also called training examples.
- $x^{(i)}$ is the value of the variable “Area” and $y^{(i)}$ is the value of the real price of the i^{th} training example.

Additional information regarding the MSE can be found here: website.

The next step consists in computing the cost function for an initial set of coefficients. Theses coefficients can be arbitrarily chosen, for example:

- $\theta_0 = 0$
- $\theta_1 = 0$

Computing the gradient of the cost function with respect to θ_1 (similar process for θ_0):

$$\frac{\delta J(\theta)}{\delta \theta_1} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

Given the initial values of $\theta_0 = 0$ and $\theta_1 = 0$:

$h_\theta(x^{(1)}) = 0$

$h_\theta(x^{(2)}) = 0$

...

$h_\theta(x^{(20)}) = 0$

It comes:

$$\frac{\delta J(\theta)}{\delta \theta_1} = \frac{1}{20} \left[(0 - 3300) \cdot 75 + (0 - 2750) \cdot 64 + \ \ldots \ + (0 - 3600) \cdot 57 \right]$$

$$\frac{\delta J(\theta)}{\delta \theta_1} = -3.0378e + 05$$

This provides the first value of the derivative of the cost function with respect to θ_1. In the next step the value of θ_1 will be updated. This new value decreases the value of the cost function and brings one step closer to the goal of minimizing it.

3. Update the coefficient θ_1 and repeat

The coefficient θ_1 will be updated as follows:

$$\theta_1 := \theta_1 - \alpha \frac{\delta J(\theta)}{\delta \theta_1}$$

where α is the so-called learning rate. The learning rate controls the step size and has a strong influence on performance of the model. If the learning rate is too large, the algorithm will not converge and if it is too small, minimizing the objective could be very time-consuming, requiring a high number of iterations. The choice of the learning rate is based on experience and needs some trial and error. By plugging this gradient into the update rule and iterating it over, the cost function moves towards its minimum, providing step by step the optimal parameter θ_1 minimizing the cost function.

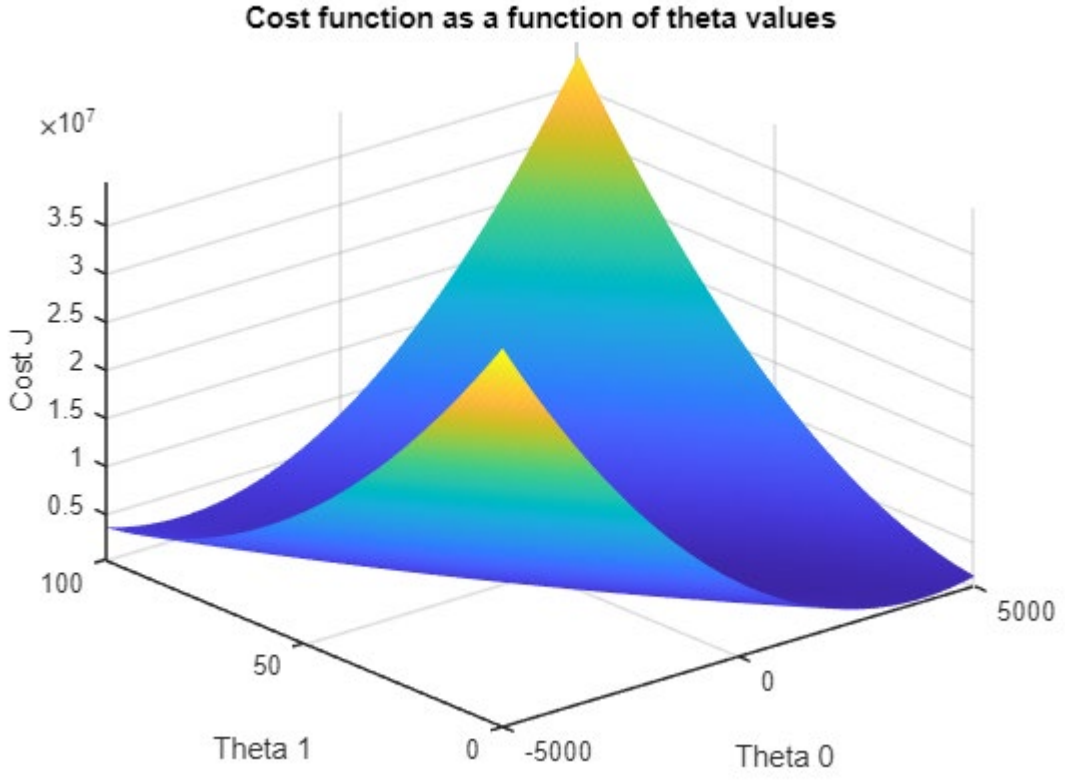
The following section provides an illustration of the gradient descent method.

Visualization of the gradient descent

Remark: All details in the code below have not to be understood into details.

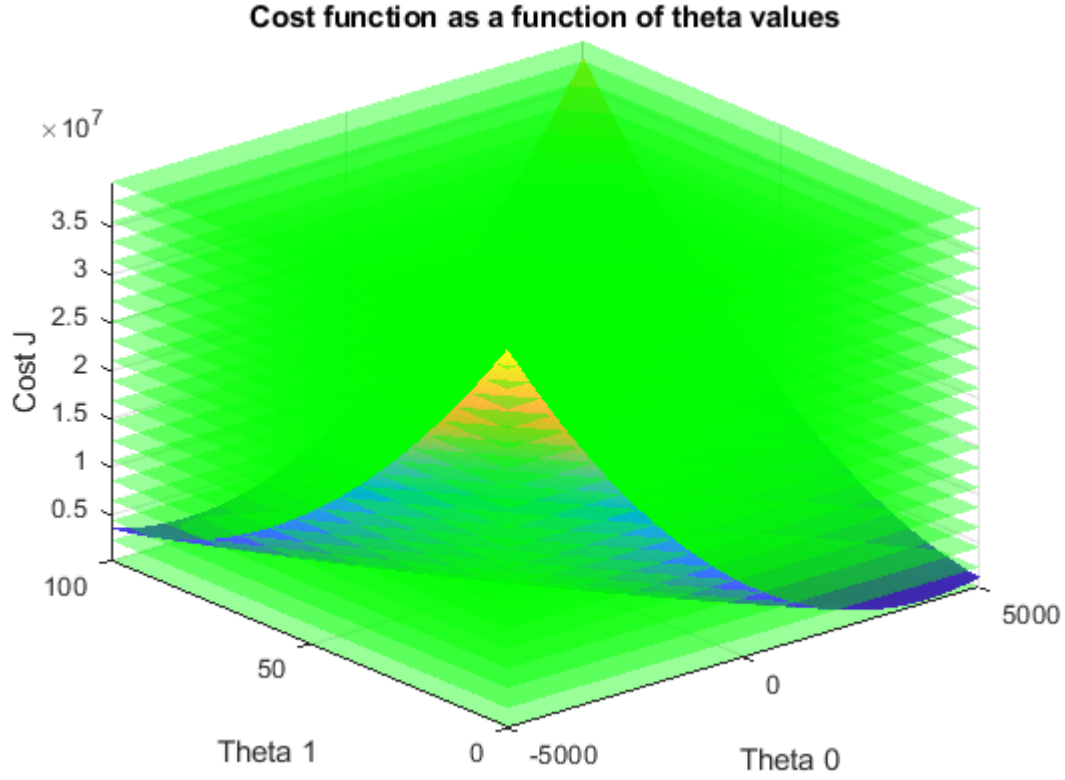
In the next section, the cost function is plotted as a function of θ_0 and θ_1:

```
clear all; clc;
T1 = readtable('Zurich_area_prices.xlsx','VariableNamingRule','preserve');


X = T1.("Area [m^2]");
y = T1.("Cost [CHF]");
m = length(y);


% Add the bias term to X matrix
X_bias = [ones(m, 1), X];


% Range for theta values
theta0_vals = linspace(-5000, 5000, 100);
theta1_vals = linspace(0, 100, 100);


J_vals = zeros(length(theta0_vals), length(theta1_vals));


for i = 1:length(theta0_vals)
    for j = 1:length(theta1_vals)
        t = [theta0_vals(i); theta1_vals(j)];
        h = X_bias * t;
        J_vals(i,j) = (1 / (2*m)) * sum((h - y).^2);
    end
end


% 3D Plot
figure;
surf(theta0_vals, theta1_vals, J_vals', 'EdgeColor', 'none');
xlabel('Theta 0');
ylabel('Theta 1');
zlabel('Cost J');
title('Cost function as a function of theta values');
view(-40, 30);


% Adjust axis limits
axis([-5000 5000 0 100 min(min(J_vals)) max(max(J_vals))]);
z_vals = linspace(min(min(J_vals)), max(max(J_vals)), 20); % 20 planes
hold on;
```



It can be observed that the plotted surface has the shape of a paraboloid. In the next step the surface will be divided by orthogonal (x-y)-planes along the z-axis:

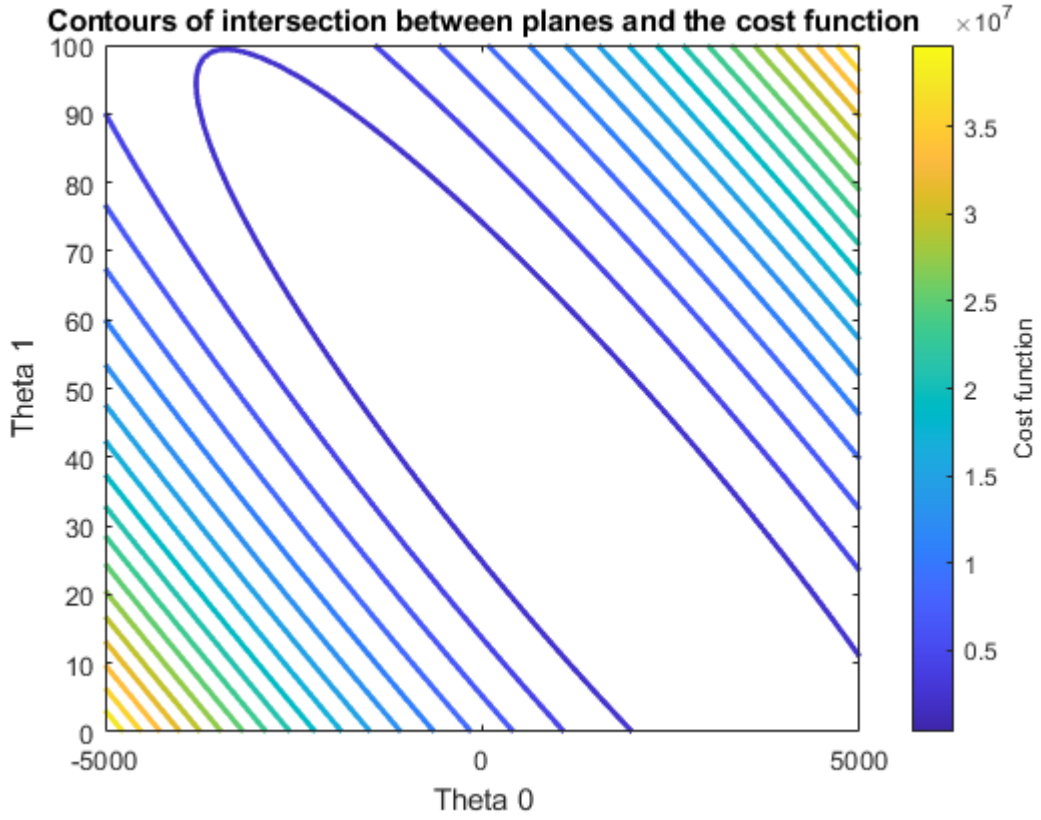
```
for z = z_vals
    % Plot each plane
    [X_plane, Y_plane] = meshgrid(theta0_vals, theta1_vals);
    Z_plane = z * ones(size(X_plane));
    surf(X_plane, Y_plane, Z_plane, 'FaceColor', 'g', 'FaceAlpha', 0.4, 'EdgeColor', 'none');
    pause(0.5);
end


hold off;
```



In the next step 2D-plots of the contour of the intersection lines between the cost function and the (x,y) planes are generated.

A contour plot represents a three-dimensional surface by plotting slices with constant z-values, called contours, in two-dimensions: Each contour line represents locations that have the same constant z-value.

```
figure;


contour(theta0_vals, theta1_vals, J_vals', z_vals, 'LineWidth', 2);
xlabel('Theta 0');
ylabel('Theta 1');
title('Contours of intersection between planes and the cost function');
colorbar;
```

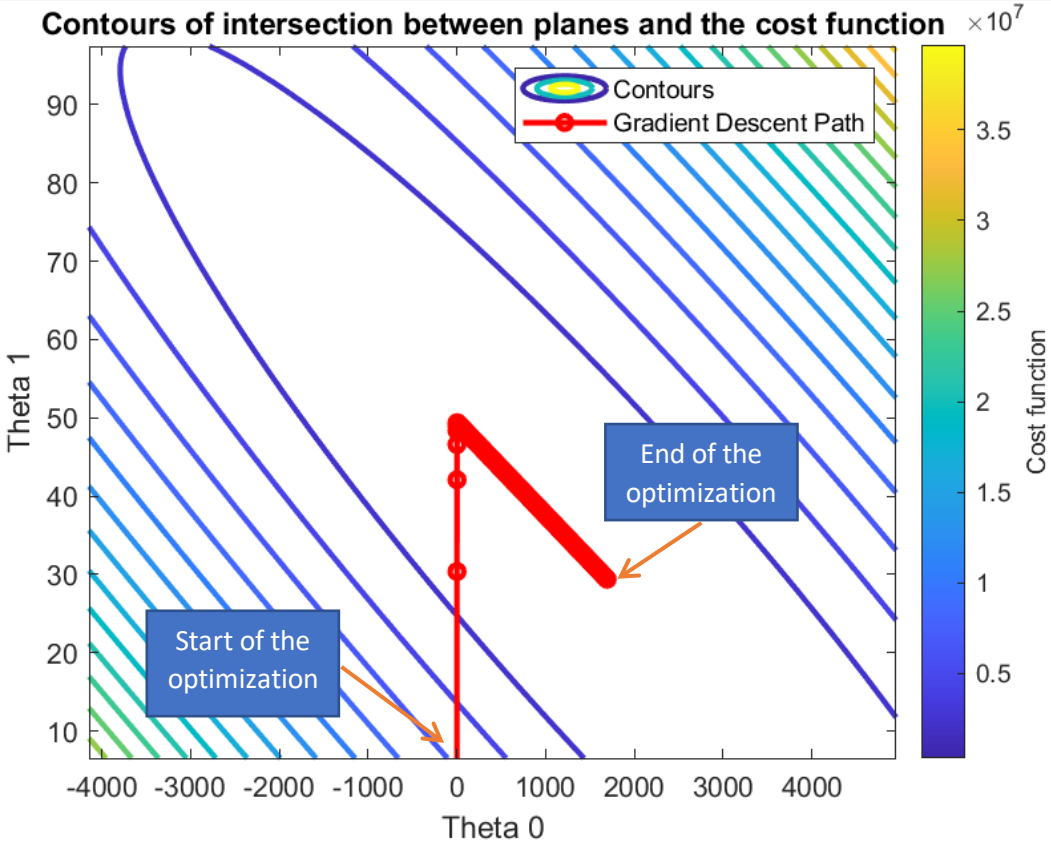


When gradient descent is performed starting on a define point of these contours, the process can be compared to a hiker progressing in a landscape and trying to reach the lowest point in the valley by always moving in the direction of steepest descent (the gradient method is also called the steepest descent method). The path of the hiker can be represented on the contours plot and provide visual insights into the efficiency of the optimization process.

How the values of θ_0 and, θ_1 evolve with the update function defined earlier has been plotted in the following graph. The path should move to the lowest point of the cost function (its minimum), which is here located in the very center of the most central contour line:

```
hold on;


% Gradient Descent Parameters
alpha = 0.0001;
num_iterations = 1000000;
theta_0 = 0;
theta_1 = 0;


% Store the path of theta values
theta_path = zeros(2, num_iterations + 1); % +1 to store the initial theta
theta_path(:,1) = [theta_0; theta_1];
```

```
for iter = 1:num_iterations
    % Gradient Descent Update
    h = X_bias * [theta_0;theta_1];
    errors = h - y;
    delta_0 = (1/m) * (sum(errors));
    delta_1 = (1/m) * sum(X_bias(:,2)' * errors);
    theta_0 = theta_0 - alpha * delta_0;
    theta_1 = theta_1 - alpha * delta_1;

    % Store the updated theta value
    theta_path(:,iter+1) = [theta_0; theta_1];
end


% Plot the gradient descent path on top of the existing contour plot
plot(theta_path(1,:), theta_path(2,:), 'r-o', 'LineWidth', 2, 'MarkerSize', 5);
legend('Contours', 'Gradient Descent Path');


hold off;
```

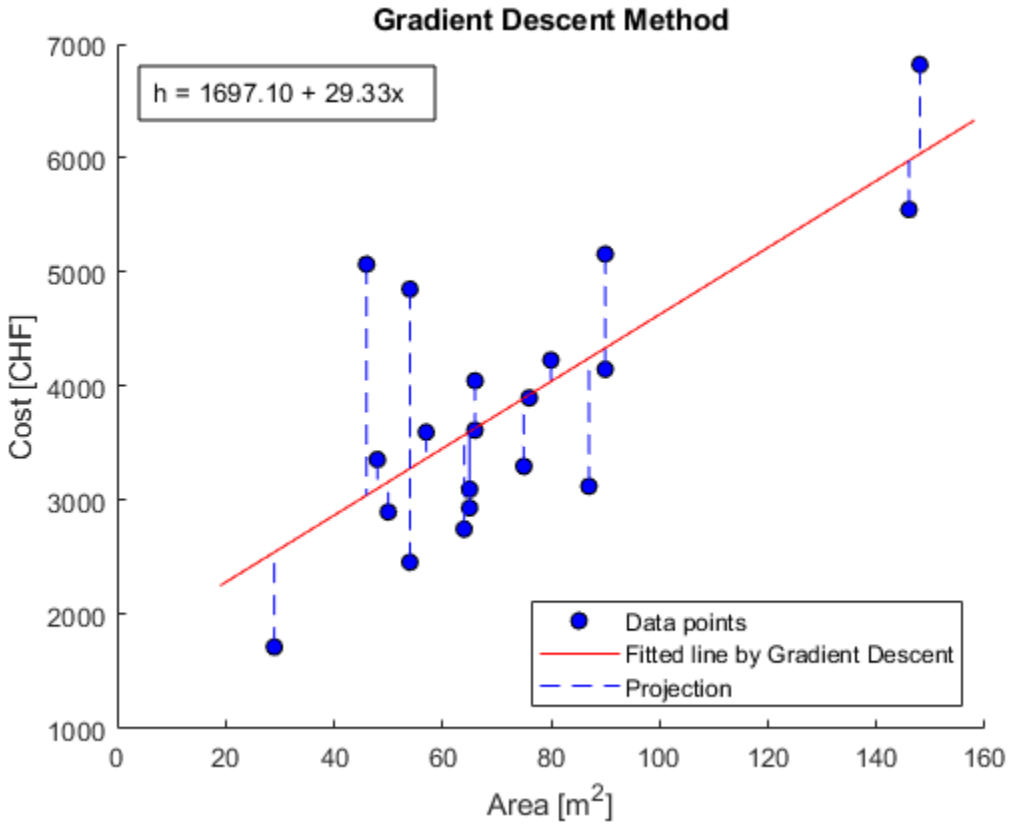


For each successive step of the algorithm, the slope (gradient of the cost function) decreases. Hence, each subsequent increment of θ_0 and θ_1 (or steps) corresponding to a new iteration becomes smaller.

The code below crates a plot of the line providing the best fit through the data points. The number of iterations can be changed in the code to see its influences in the fitting process.

```
clear all; clc;
T1 = readtable('Zurich_area_prices.xlsx','VariableNamingRule','preserve');


X = T1.("Area [m^2]");
y = T1.("Cost [CHF]");
m = length(y);


% Add the bias term to X matrix
X_bias = [ones(m, 1), X];


% Gradient descent parameters
alpha = 0.0001;
iterations = 1000000;
theta_gd = [0; 0];


% Perform gradient descent
for iter = 1:iterations
    h = X_bias * theta_gd;
    errors = h - y;
    delta = (1/m) * X_bias' * errors;
    theta_gd = theta_gd - alpha * delta;
end


% Create a new figure
figure;


% Plot the original data
scatter(X, y, 'MarkerEdgeColor','k','MarkerFaceColor','b');
hold on;
xlabel('Area [m^2]');
ylabel('Cost [CHF]');
title('Gradient Descent Method');


% Generate points for the regression line
x_range = [min(X) - 10, max(X) + 10];
y_range_gd = theta_gd(1) + theta_gd(2) * x_range;


% Plot the regression line
plot(x_range, y_range_gd, '-r');


% Show projections onto the line (the residuals)
for i = 1:length(X)
    x_pt = X(i);
    y_pt = y(i);
```

```
    y_proj_gd = theta_gd(1) + theta_gd(2) * x_pt;
    line([x_pt, x_pt], [y_pt, y_proj_gd], 'Color', 'b', 'LineStyle', '--');
end


% Add legend
legend('Data points', 'Fitted line by Gradient Descent', 'Projection','Location','southeast');


% Display the function of the fitted line
str_gd = sprintf('h = %.2f + %.2fx', theta_gd(1), theta_gd(2));
annotation('textbox', [0.15, 0.8, 0.1, 0.1], 'String', str_gd);


hold off;
```



Question: Considering the learning rate used in this example, do you think that the gradient descent method is suitable here?

Method 2: Normal Equation

The second optimization method presented in this tutorial is called “Normal Equation”.

Again, a least square cost function is used to minimize the sum of terms between the measured and the computed response. In the case of the Normal Equation the measured response must be compulsory fitted using a linear-regression model.

The Normal Equation is formulated as follows:

$$\theta = (X^T X)^{-1} X^T y$$

where *X* is a vector containing each input instance (for example the flat areas of the previous example) and y is a vector containing output for each instance (real price, in the previous example).

θ is a vector containing the coefficients of the linear regression function that minimize the cost function.

Contrary to the gradient descent method, which is solved iteratively, the Normal Equation method is solved analytically in one unique step.

Cost function and its derivative

As in the previous example, the cost function to minimize is the least square function:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

To find the minimum of *J(θ),* its derivative with respect to *θ* must be computed.

The mathematical function, or hypothesis $h_\theta(x)$ providing a computed prediction of the real measurements, is chosen as a linear function of the input variables (x_1 to x_n):

$$h(\theta) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$$

In matrix notation, $h_\theta(x)$ can be expressed as $X\theta$ where *X* is a $m \times (n+1)$ matrix containing all the input parameters (for example, n_1=area, n_2=number of bedrooms), where m denotes the number of samples where the difference between the measured response (price) and the hypothesis is computed; n is the number of input parameters that have been considered (n=2 is the previous example). The matrix *X* contains n+1 columns in order to include the additional coefficient θ_0 in the hypothesis. Finally, *θ* is a $(n+1) \times 1$ column vector containing all coefficients θ_i and *y* is a $m \times 1$ column vector containing all the measured responses $y^{(i)}$.To solve the equation $\theta = (X^T X)^{-1} X^T y$, an additional column has to be added to the vector containing the $x^{(i)}$ to provide a (n+1)x2 matrix:

$$X = \begin{bmatrix} 1 & x_1^1 \\ 1 & x_1^2 \\ 1 & \vdots \\ 1 & x_1^n \end{bmatrix}$$

Considering the simplified problem, where only one input parameters (for example the area of flats) is considered, the θ-vector containing all coefficients of the hypothesis has only two components:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

In matrix form, the Term $h_0(x) - y$ can be written as $X\theta - y$. This is a $m \times 1$ vector where the i[th] element is equal to $h_\theta(x^{(i)}) - y^{(i)}$.

Replacing in the cost function, it comes:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (X\theta - y)^{(i)2} = \frac{1}{2m} (X\theta - y)^T (X\theta - y)$$

In order to minimize $J(\theta)$, its gradient with respect to θ must be equal to zero:

$$\nabla J(\theta) = \frac{1}{m} X^T (X\theta - y) \text{ and } \nabla J(\theta) = 0$$

Where:

Solving for θ:

$$X^T (X\theta - y) = 0$$

$$X^T X\theta = X^T y$$

$$\theta = (X^T X)^{-1} X^T y$$

This is the Normal Equation formula. Its solution provides the vector of coefficients θ that minimizes $J(\theta)$.

To make it more comprehensive, the normal equation method will be illustrated in the next section.

Visualization of the Normal Equation

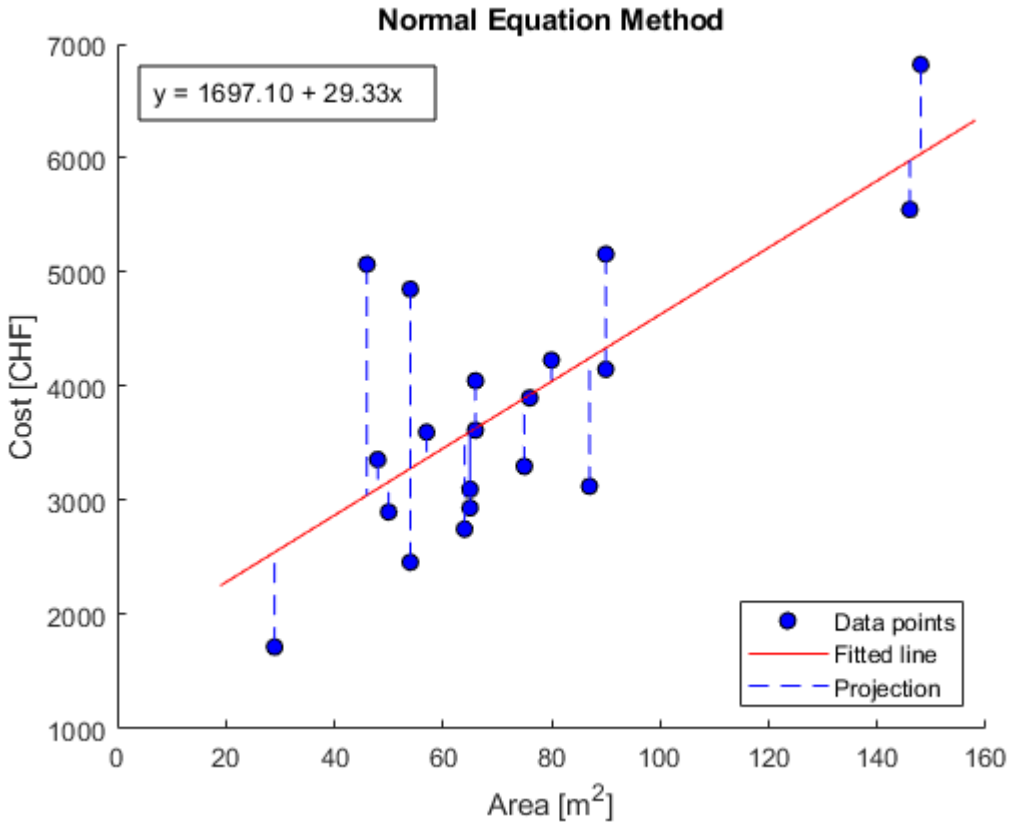
Running the code below will generate the plot of couples (area, price) that have already been used with the steepest descent method.

With the normal equation method, the best fitting approximation is the function $h_\theta(x)$ (hypothesis) for our data. The minimized error corresponds to the projection of one measurement point to the prediction function $h_\theta(x)$, parallel to the "price" axis.

Remark: The matrix-$X^T X$ has to be inverted. The 'pinv' command corresponding to the Moore-Penrose pseudoinverse, which is more stable numerically than the 'inv' command will be used for the purpose.

```
clear all; clc;
T1 = readtable('Zurich_area_prices.xlsx','VariableNamingRule','preserve');


X = T1.("Area [m^2]");
y = T1.("Cost [CHF]");


% Add the bias term to X matrix
X_bias = [ones(length(X), 1), X];


% Calculate theta using the Normal Equation
theta = pinv(X_bias' * X_bias) * X_bias' * y;


% Create a new figure
figure;


% Plot the original data
scatter(X, y, 'MarkerEdgeColor','k','MarkerFaceColor','b');
hold on;
xlabel('Area [m^2]');
ylabel('Cost [CHF]');
title('Normal Equation Method');


% Generate points for the regression line
x_range = [min(X) - 10, max(X) + 10];
y_range = theta(1) + theta(2) * x_range;


% Plot the regression line
plot(x_range, y_range, '-r');


% Show projections onto the line (the residuals)
for i = 1:length(X)
    x_pt = X(i);
    y_pt = y(i);
```

```
    y_proj = theta(1) + theta(2) * x_pt;
    line([x_pt, x_pt], [y_pt, y_proj], 'Color', 'b', 'LineStyle', '--');
end


% Add legend
legend('Data points','Fitted line','Projection','Location','southeast');
hold off;


% Display the function of the fitted line
str = sprintf('y = %.2f + %.2fx', theta(1), theta(2));
annotation('textbox', [0.15, 0.8, 0.1, 0.1], 'String', str);
```



Now print the theta values:

```
% Print the calculated theta values
fprintf('Calculated theta values:\n');
```

```
Calculated theta values:
```

```
fprintf('theta_0 = %.4f\n', theta(1));
```

```
theta_0 = 1697.1043
```

```
fprintf('theta_1 = %.4f\n', theta(2));
```

```
theta_1 = 29.3317
```

Advantages and disadvantages of the normal equation method

Advantages and disadvantages regarding the normal equation method are discussed below:

Advantages:

- **No Need for additional parameters:** The gradient descent requires to choose the appropriate learning rate α, defining the convergence rate of the method. The normal equation has the advantage to provide the optimal coefficients in one step.
- **No Iterations:** The normal equation provides a direct, closed-form solution to identify the optimal parameters set θ, making it computationally simpler as an iterative process.
- **Easy to Implement:** The algorithm is straightforward to implement as it only involves matrix operations such as inversion or multiplication.

Disadvantages:

- **Computational Complexity:** The normal equation implies to invert a $(n+1) \times (n+1)$ matrix, where *n* is the number of input parameters. The matrix inversion is in case of high n values time-intensive, making the normal equation inefficient.
- **Lack of Flexibility:** Unlike gradient descent, which can be used to find the minimum of a variety of cost functions and can be adapted to various kinds of regression easily (like ridge or lasso), the normal equation is specifically dedicated to linear regression costs functions and is therefore less flexible.
- **History storage:** Matrices used in the calculations of the normal equation must be stored, which can consume a significant amount of memory for large datasets.

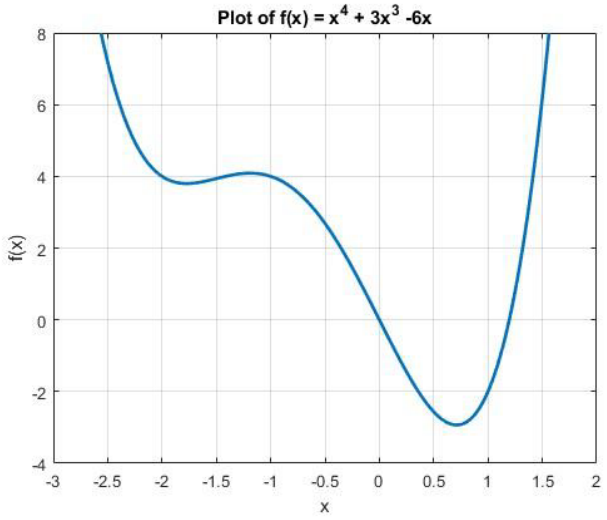
The following section sums up when the Normal Equation should be used.

Domain of usability:

1. **Small *n*, large *m*:** A small number of input parameters and a large number of measurements, are the conditions for a computational efficient use of the Normal Equation.
2. **Exact Solution:** If the inverse of $X^T X$ exists, the Normal Equation will provide the exact solution in one step and without any iteration.
3. **Linear Regression:** The Normal Equation works only with linear regression models and is not applicable to logistic regression or to other types of models.

Questions regarding the Gradient Descent and the Normal Equation methods

1. Explain in your own words how gradient descent and the normal equation method work. What are they trying to minimize?
2. Discuss the computational complexity of both gradient descent and the normal equation method. Which one do you think would be more efficient for very large datasets?
3. Do both methods lead to the same values of θ? Run the code for both methods and compare the θ values. What do you observe?
4. Given the dataset that has been worked on, which method is more suited from your point of view and why? Consider factors like the size of the dataset and the presence (or absence) of outliers.
5. Which of the two-optimization method would you choose to find the global minimum of the function below? Justify your answer!



Part 2

Multi-dimensional linear regression

In the example above, the basics of linear regression have been learned, where only one single input parameter has been considered (price = f(area)). However, the experience shows that considering the influence of multiple input variables, making the problem multi-dimensional, generally lead to a better approximation of the function of interest.

The new example of optimizing laser transmission welding to weld plastics together will illustrate this purpose in the next section.

Laser Transmission welding

The laser transmission welding process has been learned in the KT4 lecture. The process is a fast, contactless, and environment-friendly joining process of thermoplastics. The laser first penetrates a laser-transparent substrate and is stopped by a second substrate, which absorbs the energy of the laser. The contact area between the two substrates is heated and reached the melt-temperature of the substrates, which fuse together. In the cooling phase, the melt solidifies, resulting in a solid welded-joint between the two substrates. The strength of the weld seam is dependent on the laser intensity:

$$I_p = \frac{Power}{Spot\ \ size}$$

and laser-polymer interaction time:

$$T = \frac{Length\ \ of\ \ the\ \ weld\ \ seam}{Speed}.$$

The laser energy density I_E is defined as:

$$I_E = I_p \times T$$

Since the heat conduction results from the contact between the two substrates, contact pressure is a very important process-parameter. The contact between the surfaces is generated by a clamping device in which four pneumatic cylinders compress the two substrates to be welded against a safety glass. The pressure of the pneumatic cylinders generating the clamping pressure is an input parameter that can be increased or decreased.

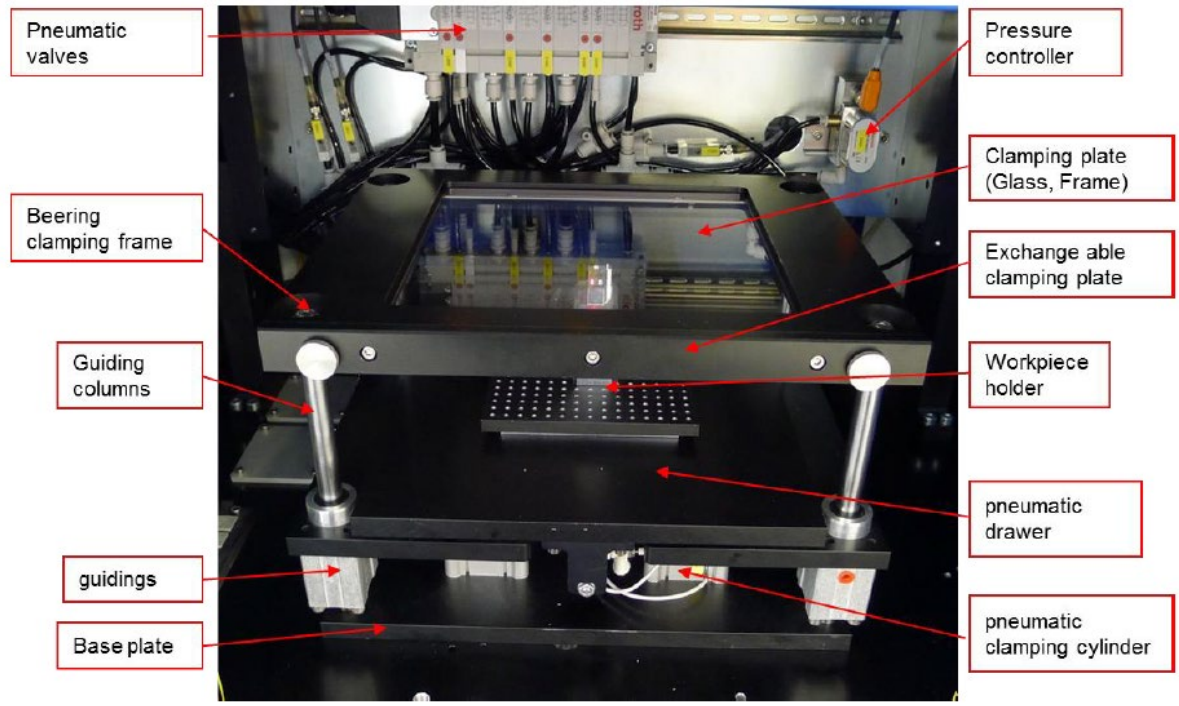


Figure 3: Front view of the clamping device.

Another important input parameter is the thickness of the substrates. The thickness of transparent substrates available in this workshop are equal either to 1 mm or 3 mm. The thickness of the transparent substrate influences the pathway of the laser to the absorbing substrate and modifies the spot size of the laser at the interface.

Question: How can the thickness of the transparent substrate influence the area and the strength of the weld seam? Justify your answer!

Input data

Tensile shear tests of welded specimens have been performed. The substrates dimensions are 34 mm x 62 mm where the overlap length of the two substrates is equal to 20 mm. The weld seam is set in the middle of the overlapping area and is 26 mm long. The tests are performed with the 50 kN tensile test machine at the IWK. The strength and the area of the weld seam are measured. Both outputs can be used as criteria to evaluate the quality of the welded seam. A high resp. low strength and a large rep. not visible welded surface can be a priori linked to a good resp. bad quality of the weld seam.

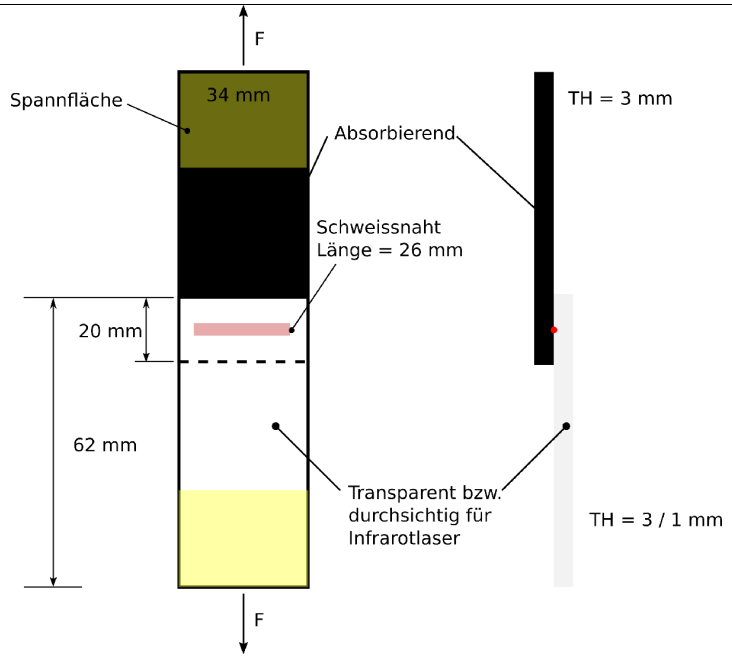
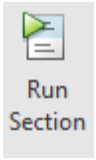


Figure 4: Dimensions of the lap shear specimen

Clicking on the "Run Section" button run the code below and generate a table containing intput and output data:




```
clear ITBO_interactive_script; close all; clc; clear all;
% read data;
opts = detectImportOptions('Training_Data.xlsx',...
    'sheet','ITBO-2parameter');
T = readtable('Training_Data.xlsx',...
    opts)
```

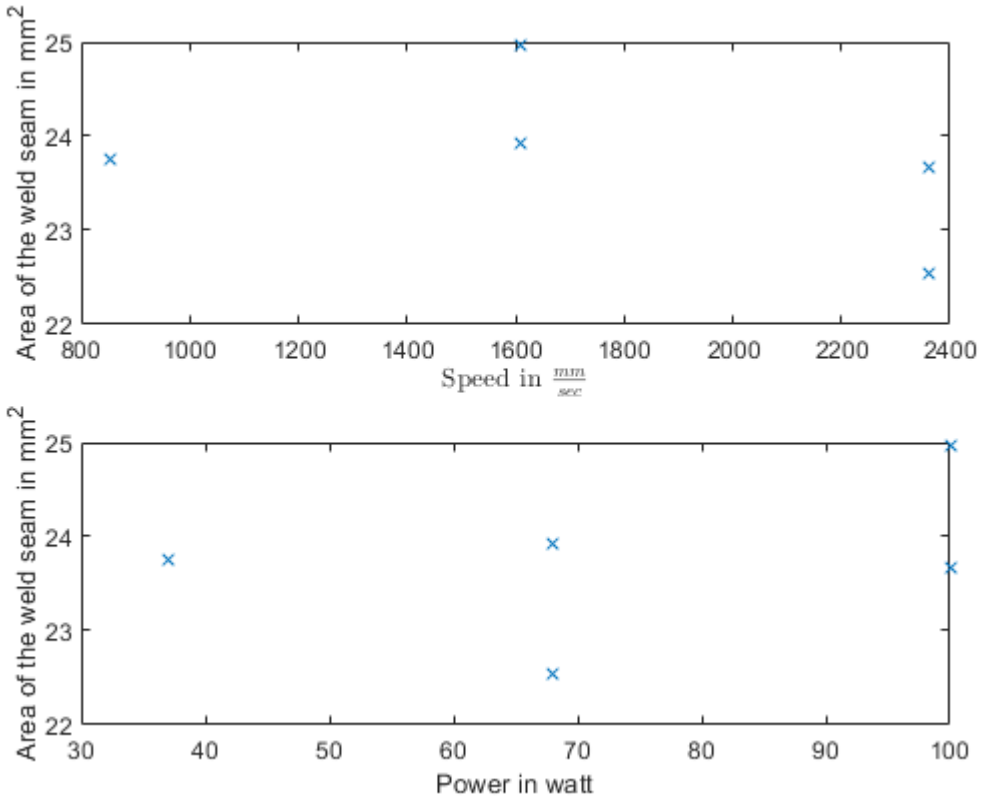
T = 5×7 table

	Power	Speed	Pressure	TH	Weld_Seam_Area	Failure_Force	Strength
1	68	2362	1	3	22.5333	395.1100	17.5345
2	37	854	1	3	23.7467	501.5500	21.1209
3	68	1608	1	3	23.9200	521.2300	21.7906
4	100	2362	1	3	23.6600	587.9100	24.8483
5	100	1608	1	3	24.9600	660.8700	26.4772

The first proposed task consists in predicting the weld seam area as a function of the two input parameters power and speed.

Plotting the weld area as a function of each input parameter separately helps to visualize a possible correlation between the input and output data. Run the following section.

```
tiledlayout(2,1);
nexttile;
plot(T.Speed,T.Weld_Seam_Area, 'x');
xlabel('Speed in $\frac{mm}{sec}$', "Interpreter","latex");
ylabel("Area of the weld seam in mm^2");


nexttile;
plot(T.Power, T.Weld_Seam_Area, 'x');
xlabel("Power in watt");
ylabel("Area of the weld seam in mm^2")
```



Question:

1. Do you observe a correlation between the input and output parameters?
2. Considering the equation defining laser energy density, explain why do some points have the same speed or the same power, whereas the corresponding area of the weld seam varies?

A hypothesis to predict the weld seam area as a linear function of power and speed is written as follow:

$$weld\_seam\_area(\theta) = \theta_0 + \theta_1 power + \theta_2 speed$$

or:

$$h(\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Where x_1 and x_2 represent the input parameters power and speed. In MATLAB, the values of x_1 and x_2 are summarized into the matrix X. A column of "1" will be added at the beginning of the Matrix, as explained in the section relative to the normal equation.

$$h(\theta) = \begin{pmatrix} \theta_0 & \theta_1 & \theta_2 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$$

The code section below generates the five first elements of the table in the following order: "[power speed], area":

```
x1 = T.Power;
x2 = T.Speed;
X = [x1, x2];
y = T.Weld_Seam_Area;


fprintf(' x = [%.0f %.0f], y = %.0f \n', [X(1:5,:) y(1:5,:)]');
```

```
 x = [68 2362], y = 23
 x = [37 854], y = 24
 x = [68 1608], y = 24
 x = [100 2362], y = 24
 x = [100 1608], y = 25
```

```
[X, mu, sigma] = featureNormalize(X);
m = length(X); % number of training examples
X = [ones(m,1),X]; % Add a column of ones to x
```

Gradient descent

In this section the coefficient θ of the hypothesis function will be identified using the gradient descent optimization function.

The cost function to minimize is defined as the absolute value between the hypothesis and the real size of the weld seam. The cost function $J(\theta)$ is defined as the summation of the quadratic errors between the hypothesis and the measured dataset:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

The parameters of the model to be identified are the coefficients θ of the function h. The optimal parameter set is going to minimize the cost function $J(\theta)$. To minimize $J(\theta)$, its

gradient with respect to θ can be computed and the value of θ for which the gradient is equal to zero is a solution of the problem.

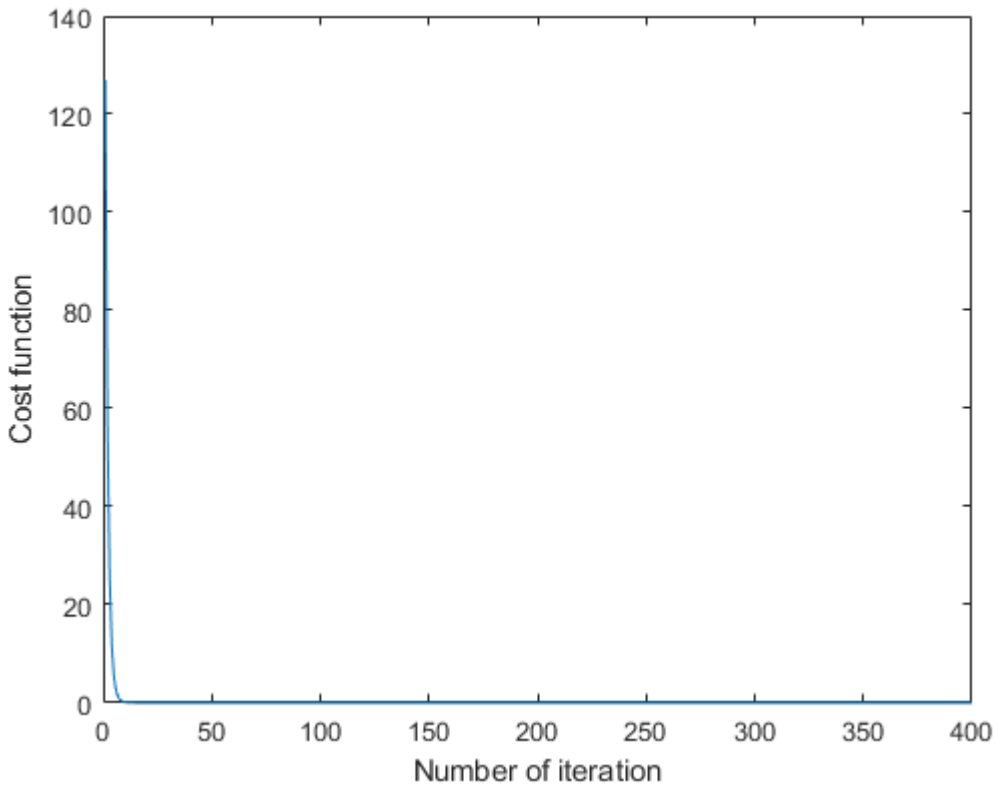
This will be done here iteratively using the gradient descent method. When using gradient descent, the user must have an idea about the location of the global minimum of $J(\theta)$. Otherwise, in the case where $J(\theta)$ is a continuous function with different local minima, the solution of the method will be the closest local minimum to the start point of the optimization process. Unfortunately, this local minimum can be far from the real global minimum.

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)}$$

After each new increment, the coefficients θ come closer to the optimal value leading to the lowest value of the cost function. A too small learning rate α, will slow down the calculation whereas a large learning rate can result in a diverging solution. An appropriate learning rate can be evaluated by plotting the cost function: The cost function should decrease steadily and never increase. After an optimum has been found, the parameters should not be updated further, since this could lead to instabilities.

In the following code, different values of α will be chosen from the drop-down menu to see its influence on the minimization of the cost function.

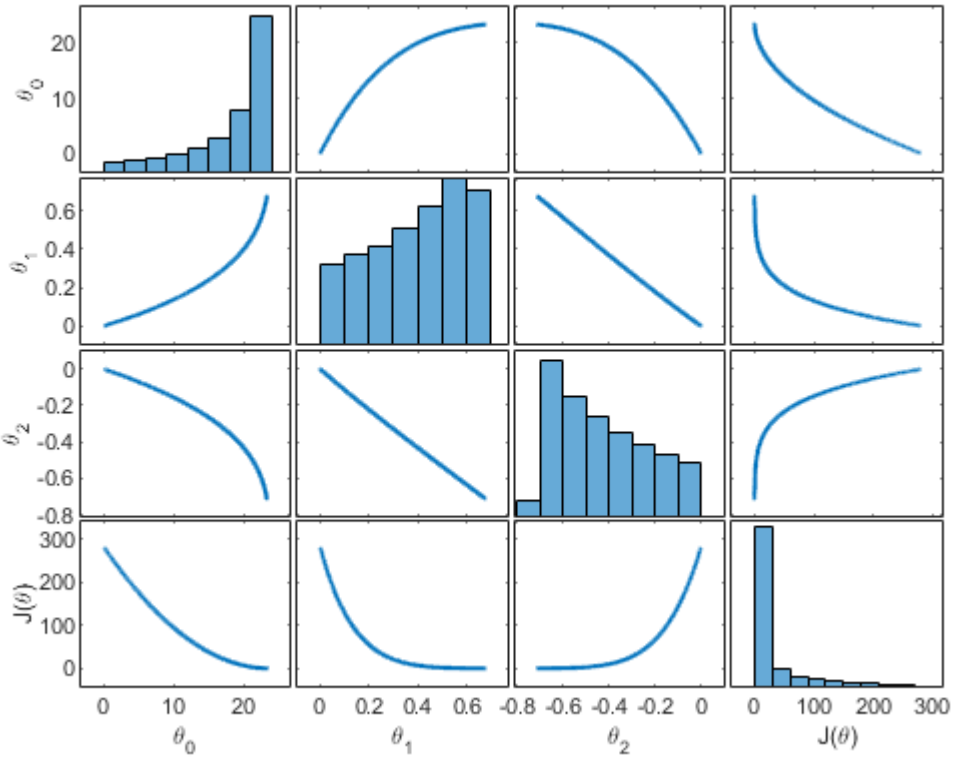
```
theta_g = zeros(width(X),1); % initialize fitting parameters
iterations = 400;
alpha = 0.01;


% The function "gradientDescent" takes the input data, first guess of theta [0 0 0], learning rate and number of iterations.
% This function returns theta from the last iteration and the history of theta and cost function.
% The response of gradient descent is affected by the first guess, as you learned in the previous section.
[theta_g,theta_hist,J_hist] = gradientDescent(X, y, theta_g, alpha, iterations);


figure;
plot(J_hist);
xlabel("Number of itteration");
ylabel("Cost function");
```



Question: Which value of α seems accurate to solve the problem? Justify your answer!

In the present case, $J(\theta)$ is a function depending on 3 parameters and should be represented in a 4-dimensional space. This cannot be represented. However, the dependencies of $J(\theta)$ to the parameters can be plotted in a matrix as represented below. At the end of the matrix plot, you should be able to see the calculated theta vector.

```
figure;
labels = {'\theta_0', '\theta_1', '\theta_2', 'J(\theta)'};


[h,ax] = plotmatrix([theta_hist J_hist]);
for i = 1:4                              % label the plots
  xlabel(ax(4,i), labels{i})
  ylabel(ax(i,1), labels{i})
end
```



The subplots along the diagonal are histogram plots of $\theta_1, \theta_2, \theta_3$, and J(θ) from the upper left corner to the lower right corner.

```
fprintf("Parameter vector computed from gradient descent: "); theta_g
```

```
Parameter vector computed from gradient descent:
theta_g = 3×1
   23.3374
    0.6741
   -0.7092
```

The coefficients of the hypothesis have been identified and the corresponding function can be plotted to gain a better understanding. The coefficients of the hypothesis are:

$$\theta_1 = (23.3374 \quad 0.6740 \quad -0.7092)$$

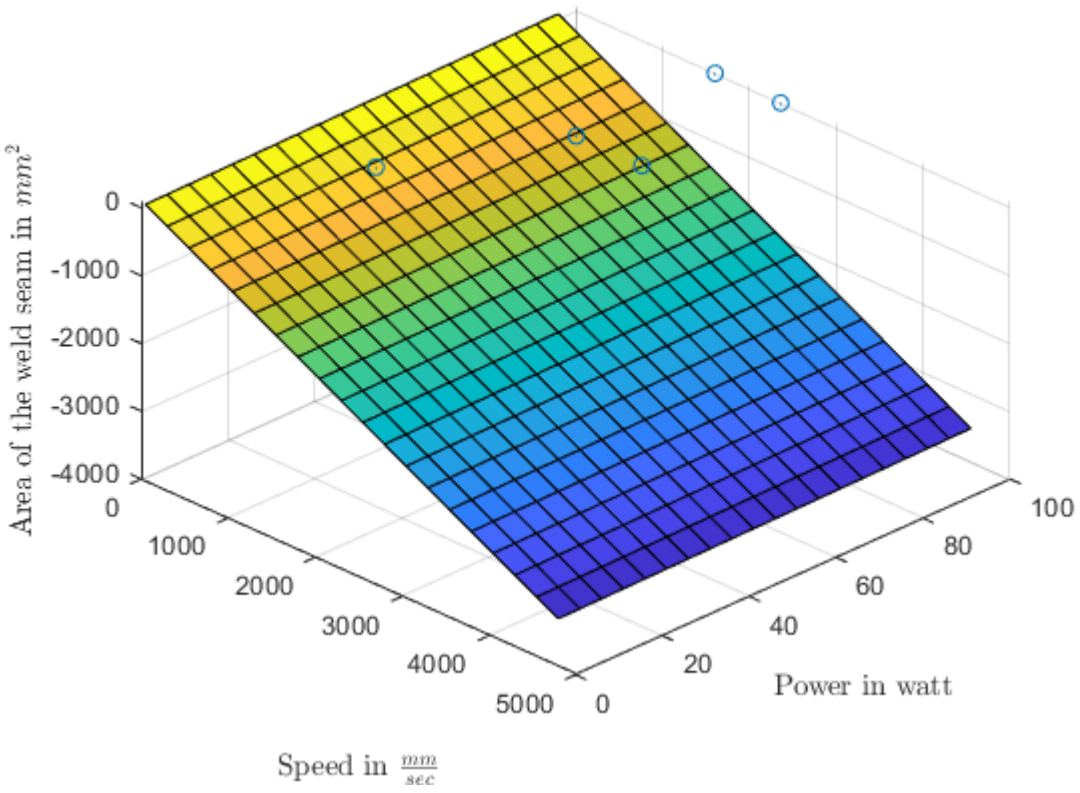
and the hypothesis is equal to:

$$h(\theta) = (23.3374 \quad 0.6741 \quad -0.7092)\begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$$

where x_1 and x_2 are the input parameters power and speed of the laser during the welding process.

The predicted hypothesis can be plotted for each iteration of the gradient descent by running the code below. The diagram in this script shows the hypothesis of the last iteration.

```
figure;
[x2__, x3__] = meshgrid(1:5:100,1:250:5000);
for i = 1:iterations
    theta_temp = theta_hist(i,:);
    h__ = theta_temp(1) + theta_temp(2)*x2__ + theta_temp(3)*x3__;
    surf(x2__,x3__,h__); hold on;
    stem3(T.Power,T.Speed,T.Weld_Seam_Area);
    ax = gca;
    ax.YDir = 'reverse';
    view([-45 45])
    xlabel("Power in watt","Interpreter","latex"); xlim([0 100]);
    ylabel("Speed in $\frac{mm}{sec}$","Interpreter","latex"); ylim([0 5000]);
    zlabel("Area of the weld seam in $mm^2$","Interpreter","latex");
    pause(0.05)
    hold off
end
```



Question:

1. What can be understood from this diagram?
2. Is the hypothesis function as expected? Justify your answer!

Hint: The last iteration provides the optimum set of parameters.

Question:

What is the area of a weld seam predicted by this hypothesis using a laser with a power of 50 watts and with a speed of 850 mm/sec?

To answer this question, the following code must be completed:

```
theta_ = [0 0 0]
theta_ = 1×3
     0     0     0
x_ = [1, 50, 850]'
x_ = 3×1
     1
    50
   850
test_answer =  theta_*x_;
fprintf("%.1f unit", test_answer);
0.0 unit
```

As observed in the section above, the Gradient Descent method was not capable to find the global optimum. This comes from the fact that its prediction is strongly linked to the chosen start point (or starting set of parameters), the learning rate α, and the number of measured data available. The location of the first θ parameters-set and the choice of the learning rate α defines the pathway to the next optimum. The gradient descent is suitable for continuous h functions and large data, where calculating the inverse of huge dense matrices is very time-consuming.

In the present case, a direct method, namely the Normal Equation method can also be used to find the absolute minimum of the cost function and thereby the parameters of the hypothesis.

Normal Equation

Computing the gradient of the cost function for each input parameter θ and solving the equations, θ can be calculated in one shot directly, which is called the closed-form solution of the linear regression:

$$\theta = (X^T X)^{-1} X^T \vec{y}$$

No further iteration is required.

```
% normal equation
X = [x1, x2];
y = T.Weld_Seam_Area;
m = length(y);


% Add intercept term to X
X = [ones(m, 1) X];
```

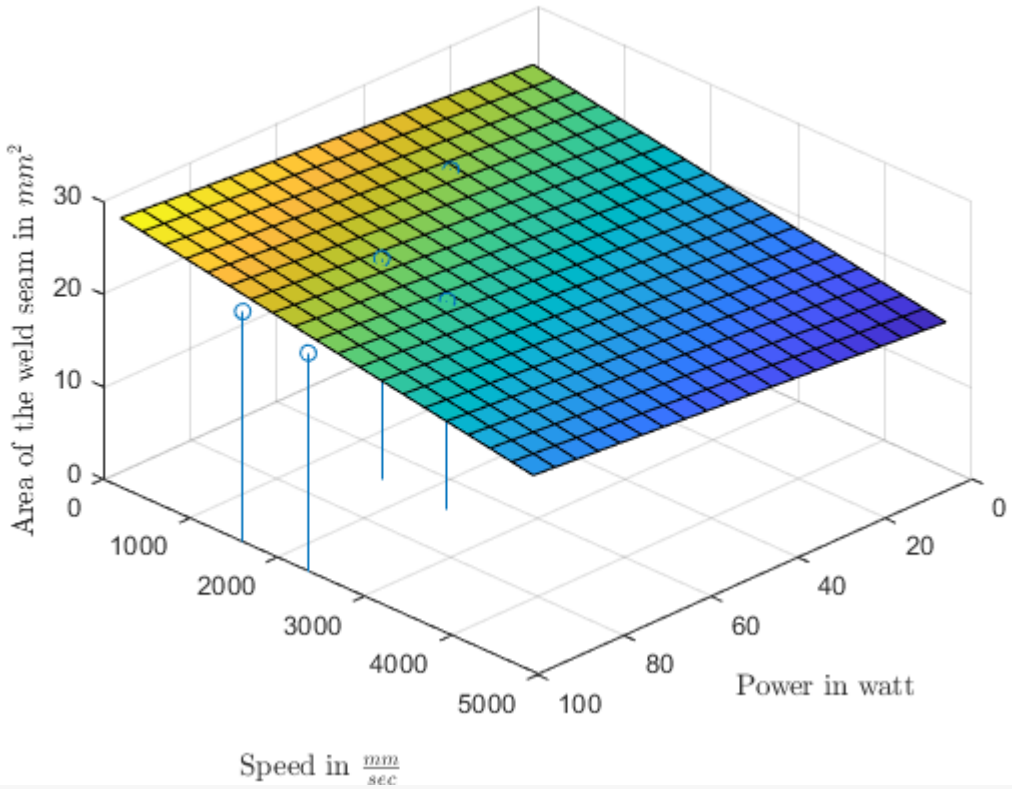
```
% Calculate the parameters from the normal equation
theta_n = inv(X'*X)*X'*y;
fprintf("parameter vector computed from normal equation: "); theta_n
```

```
parameter vector computed from gradient descent:
theta_n = 3×1
   23.8268
    0.0375
   -0.0016
```

The following code generates a plot the hypothesis obtained from normal equation.

Question: Compare and comment the optimum hypothesis models resulting from the gradient descent and from the normal equation methods.

```
figure;
[x2__, x3__] = meshgrid(1:5:100,1:250:5000);
h__ = theta_n(1) + theta_n(2)*x2__ + theta_n(3)*x3__;
surf(x2__,x3__,h__); hold on;
stem3(T.Power,T.Speed,T.Weld_Seam_Area);
ax = gca;
ax.YDir = 'reverse';
ax.XDir = 'reverse';
xlabel("Power in watt","Interpreter","latex"); xlim([0 100]);
ylabel("Speed in $\frac{mm}{sec}$","Interpreter","latex"); ylim([0 5000]);
zlabel("Area of the weld seam in $mm^2$","Interpreter","latex");
view([-45 45])
```




```
test_answer = theta_n'*x_;
fprintf("%.1f unit", test_answer);
```

24.3 unit

Results check: If the database is not updated, the optimum area should be equal to $24.3mm^2$. Otherwise ask the supervisor for more help.

Tasks

1. Find the optimum hypothesis predicting weld strengths using the same data set.
2. In the case where the thickness of the transparent substrate should be added to the prediction. Where has the script to be changed? Plots can be used for visualization purposes to understand the solution.
3. A lap shear specimen with a new sets of input parameters has been welded and its tensile strength has been evaluated. Does the result of the test match the prediction of the previously identified hypothesis function h? Add this new input at the end of the "Training_Data.xlsx". This will help to improve the quality of the prediction.

Optional task:

Linear Regression with MATLAB AI APP

How linear regression operates has been learned. The Regression Learner from MATLAB can be tested now.

In the ribbon "Apps" a drop-down menu will be found. A section named "Machine Learning and deep learning" can be found.

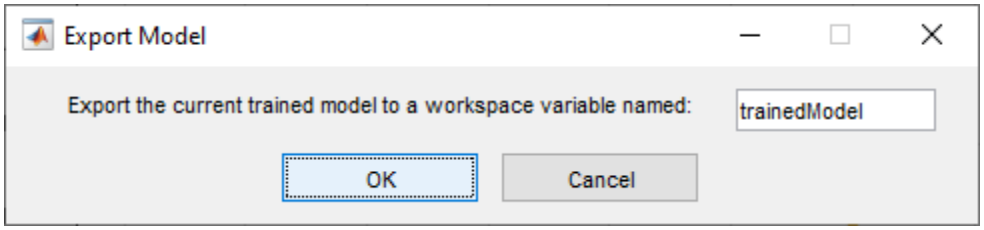
Open "Regression learner" and create a new session. A data set will be asked for.

Choose "T" from the drop-down menu.

Choose the weld seam area as the response and untick the unnecessary predictors. Finally start the session.

From the MODEL TYPE choose the linear model.

Train the model and export it by clicking on the green tick in the upper right corner of the regression learner window.



When the window above appears, the name of the model must not be changed, and click on OK. The following section has to be run finally:

```
linMdl = trainedModel.LinearModel;
theta_m = linMdl.Coefficients.Estimate;
predict(linMdl,x_(2:end)')
```

ans = 24.3190

Can the previous results be retrieved?

Conclusion:

This tutorial has provided simple example of Machine Learning processes and optimization methods used to identify optimum prediction of a set of real measured data, which depend on different input parameters.

These are basics which are used in the very actual topic of Artificial Intelligence. It is possible to use different and more advances methods to identify optimal solution, without needing to understand their mathematical background into details.