Table of Contents

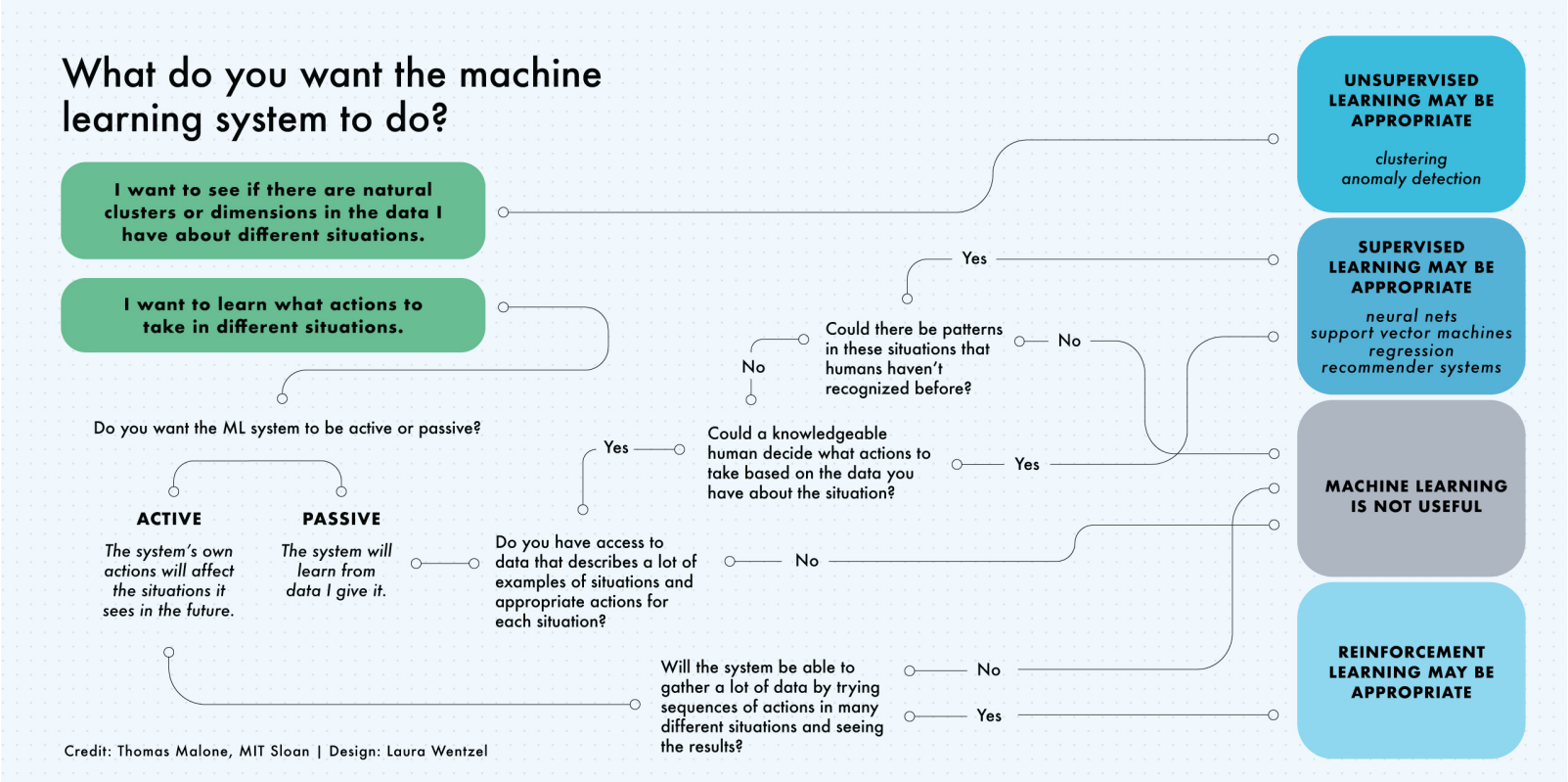
Introduction

Machine learning shows your social media feed. It suggests movies on Netflix and helps your doctor to give your diagnosis. Machine learning is part of artificial intelligence and ables computers to learn without being programmed. It was defined in the 1950s by AI pioneer Arthur Samuel as “the field of study that gives computers the ability to learn without explicitly being programmed.”

There will be no job, that ML is not part of it. An occupation can be divided into a bundle of tasks. Some of these tasks offer better applicability for technology than others. **"One criterion for a task suitability for machine learning is that the set of inputs and the corresponding set of outputs for the task can be measured sufficiently well that a machine can learn the mapping between the two sets."**

Machine learning starts with data, numbers, text, records, time series data from sensors, and so on. The data is gathered and prepared to be used as training data. The more data, the better the algorithm.

“The function of a machine learning system can be descriptive, meaning that the system uses the data to explain what happened; predictive, meaning the system uses the data to predict what will happen; or prescriptive, meaning the system will use the data to make suggestions about what action to take,” the researchers wrote.



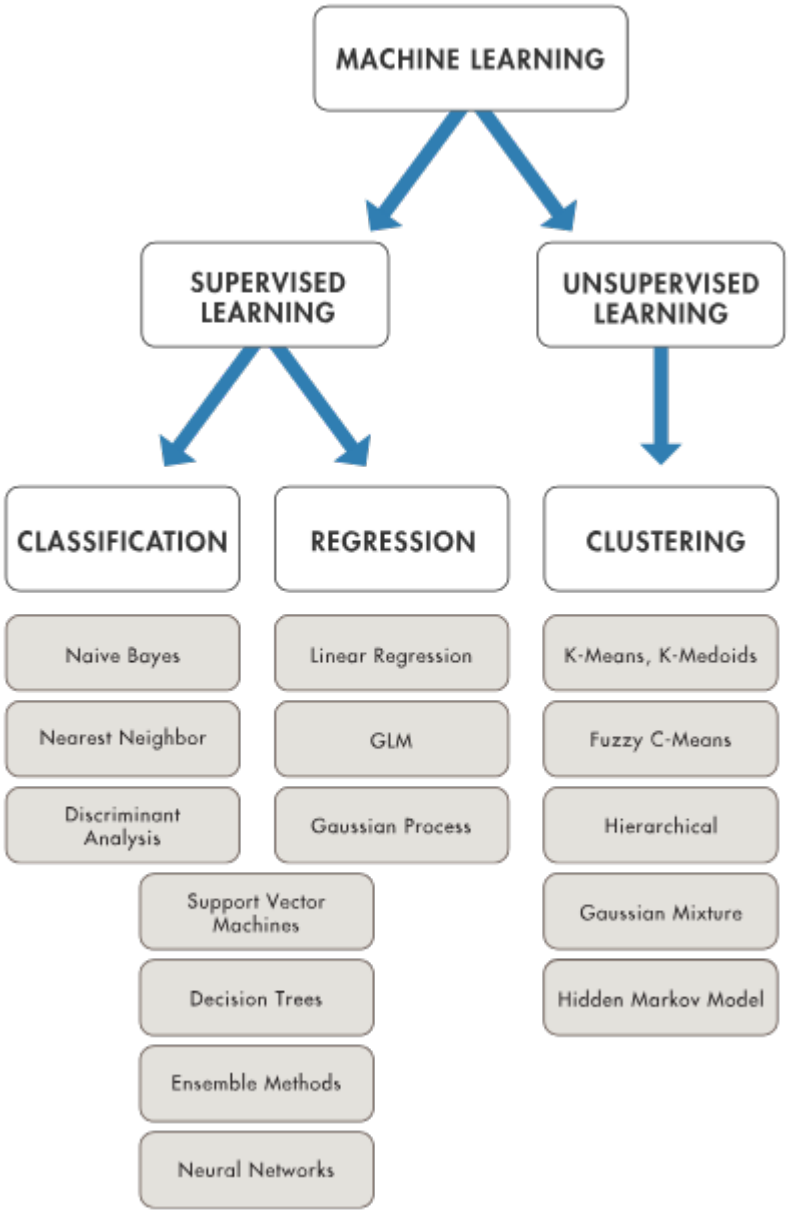
Explainability

ML Models are usualy very complex. The ability to understand what the ML model is doing and the decisions are made is the topic of "explainability". You should always ask yourself why a model does what it does. You should never treat ML as a black box. Maybe this example can help to understand the importance of "explainability". An ML algorithm is to classify radiographs as tuberculosis or non-tuberculosis. It seams that the ML could outperform the physicians. But it turned out the alogorithm was correlating results with the machines that took the image, and not necessarily the image itself. The disease is more common in developing countries. It is also more probable to find an old X-ray machine in a developing country. The ML algorithm learned that if the radiograph is taken on an older machine, the patient has probably tuberculosis. The algorithm completed the task but in a different manner as the programmer intended.

Machine learning models

Machine learning classification and regression are two main types of machine learning. The choose of right algorithem is time taking and needs experience also it can involve trial and error.

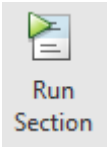
Below is an overview of machine learning classification and regression models to get you started.



Today we will concentrate on linear regression, which is one of the most simple models.

Note: you'll run into code that you have to run.

Just click into the current section you're reading and press the button "run Section":



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 using rental prices in Zürich

Similar to all the ML algorithems, the first step of training any algorithem is data. Our goal is a hypothesis which describs the relation between the input parameters and the output. The most simplest hypothesis could be a linear combination of the parameters.

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

Where θ_0 to θ_n are the coefficient of the parameters, and x_1 to x_n are the input parameters. We aim to find the appropriate coefficients for the above equation so that, given the input parameters (x_1 to x_n) accuratly we predict the $h(\theta)$. As instance we can define the rental prices in Zürich as a function of area and number of the bedrooms. Therefore the hypothesis will look like:

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

and in Matrix form:

$$h(\theta) = \boldsymbol{\theta}^{\mathbf{t}}\mathbf{X} = \begin{pmatrix}\theta_0 & \theta_1 & ... & \theta_n\end{pmatrix}\begin{pmatrix}1\\ x_1\\ ...\\ x_n\end{pmatrix}$$

Since most of the AI programs are vectorized macheans, it is necessary for the engineer to know how to perform the calculations in matrix form. This will reduce the calculation costs and improve the efficiency of the machine available to you.

But before we dive into solving the equations above, we will cover one of the most important and nessecary to understand features in ML: the cost function.

Cost function

To work precisely in the real world ML models need a high level of accuracy. But how do we know how wrong or respectively how right our model is? This is where the cost function (or: loss function) comes into play.

The purpose of the cost function is to measure how far off our model's predictions are from the actual results. The goal during training is to minimize this difference. It quantifies the error between predicted values and expected values and presents it in the form of a single real number. Depending on the problem type, different loss functions are used.

Method 1: Gradient Descent

The first method we will learn is called gradient descent. Gradient descent is an optimization algorithm for finding a local minimum of a **differentiable** function. In ML gradient descent is used to find the values of a function's coefficients that minimize a cost function as far as possible.

The main idea behind gradient descent is straightforward:

1. Start with an initial set of parameters.
2. Compute the gradient of the cost function with respect to each parameter.
3. Update the parameters in the direction of the negative gradient.

Let's have a look at each step individually:

1. Initial parameter set

To keep it simple we will focus only on the area as our input parameter in the first step. Therefore our hypothesis looks like this:

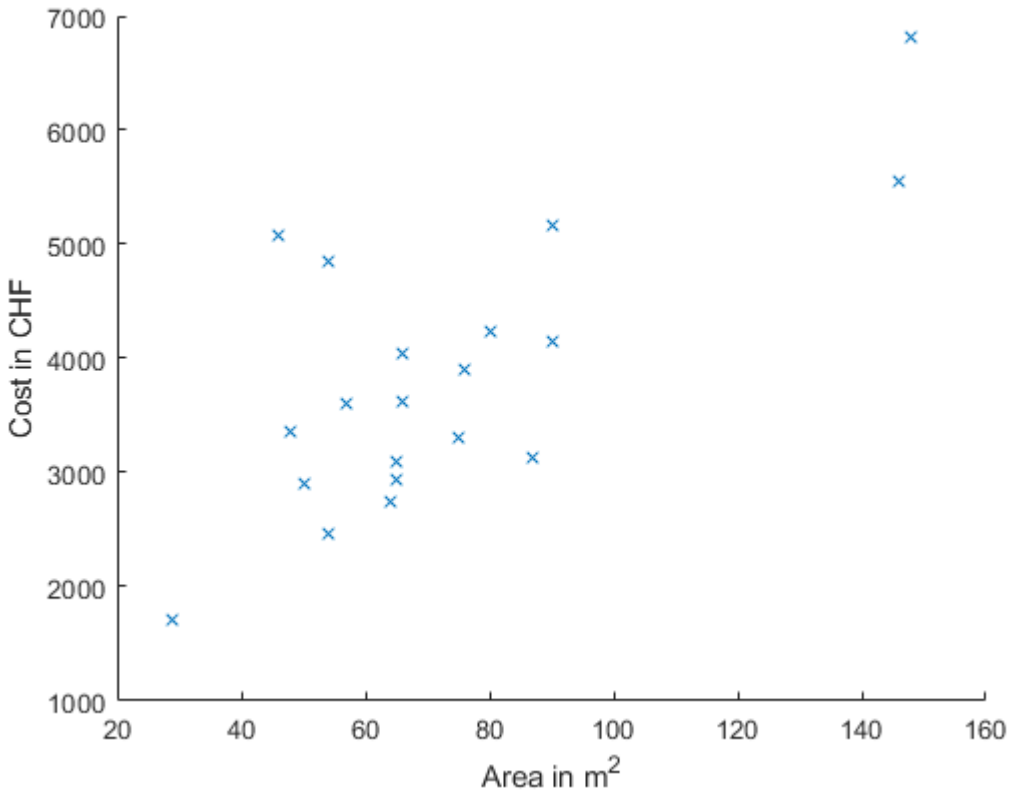
$$\text{Price} = \theta_0 + \theta_1(\text{Area})$$

We gathered some data from a popular real estate website:

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

Let's get an overview with a plot, can you already identify a trend?

```
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

First, we'll need to define a suitable cost function for our problem. Since we're looking for a linear regression solution, the Mean Square Error (MSE) cost function will suit our problem just fine. Therefore our cost function looks like this:

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

- m is the number of training examples
- $x^{(i)}$ and $y^{(i)}$ are the input and output of the i^{th} training example

If you're looking for additional information regarding the MSE, have a look at this website.

Now that we have our cost function, we'll have to arbitrarily initalize our parameters. In this example we'll choose:

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

Let's now compute the gradient for θ_1 (process is similar 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 our initial values of $\theta_0 = 0$ and $\theta_1 = 0$:

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

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

...

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

The gradient for θ_1 becomes:

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

which evaluates to:

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

This will be our fist value for the θ_1 gradient. The next step will be to update this parameter until convergence.

3. Update the parameters and repeat

We need to update the parameter to get a converging value. For this we will define an update rule:

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

where α is our learning rate. By plugging this gradient into the update rule and iterating over it, we move towards the minimum of the cost function, thereby finding the best parameters for our model.

To get a better understanding of what we've learned until now, let's visualize the gradient descent method.

Visualization of Gradient Descent

Disclaimer: you do not have to understand every detail in the code below.

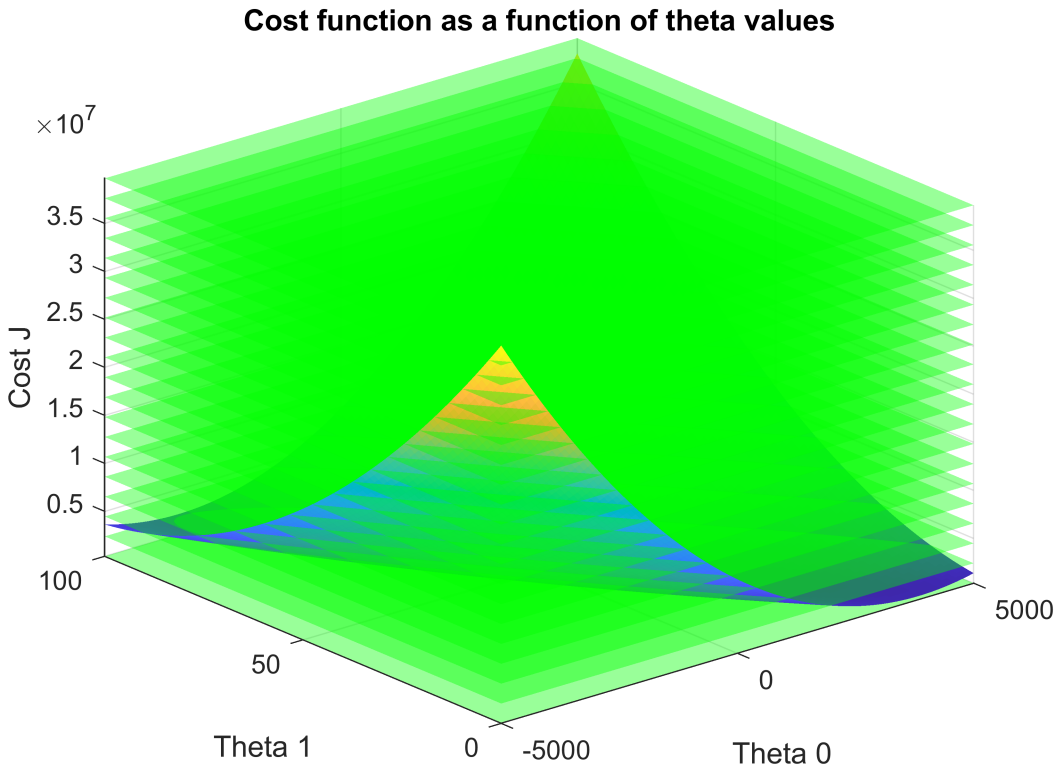
Let's plot the cost function in dependence of θ_0 and θ_1:

```
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;
```

We can observe that the plotted surface resembles a paraboloid-like structure. The next step will be dividing the surface with orthogonal x-y-planes along the z-axis:

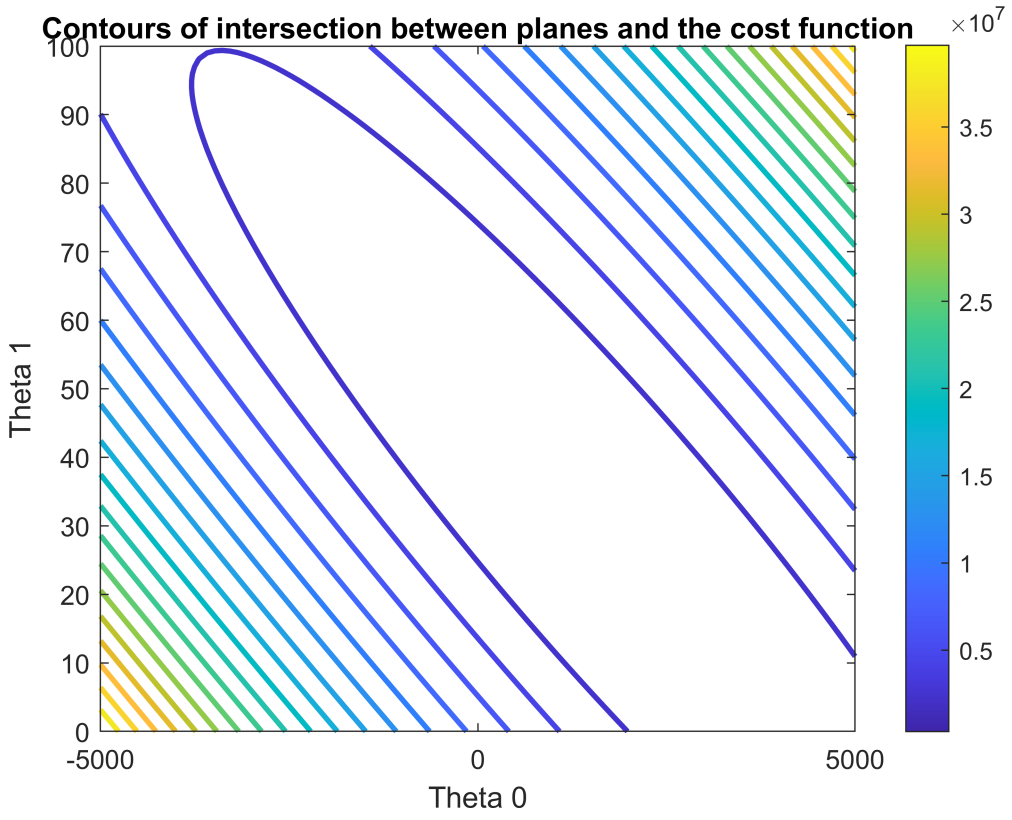
```
for z = z_vals
    % Plot each plane
    [X_plane, Y_plane] = meshgrid(theta0_vals, theta1_vals);
    Z_plane = z * ones(size(X_plane));
    surf(X_plane, Y_plane, Z_plane, 'FaceColor', 'g', 'FaceAlpha', 0.4, 'EdgeColor', 'none');
    pause(0.5);
end

hold off;
```



Now we'll generate a 2D-plot of the contour intersection lines between the cost function and the planes that were generated previously.

A contour plot represents a three-dimensional surface by plotting constant z slices, called contours, on a two-dimensional format. Each contour line represents locations that have the same constant z value. That is, if you were to walk along a contour line, you would neither ascend nor descend; you'd be walking on flat ground at a constant altitude.

```
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;
```

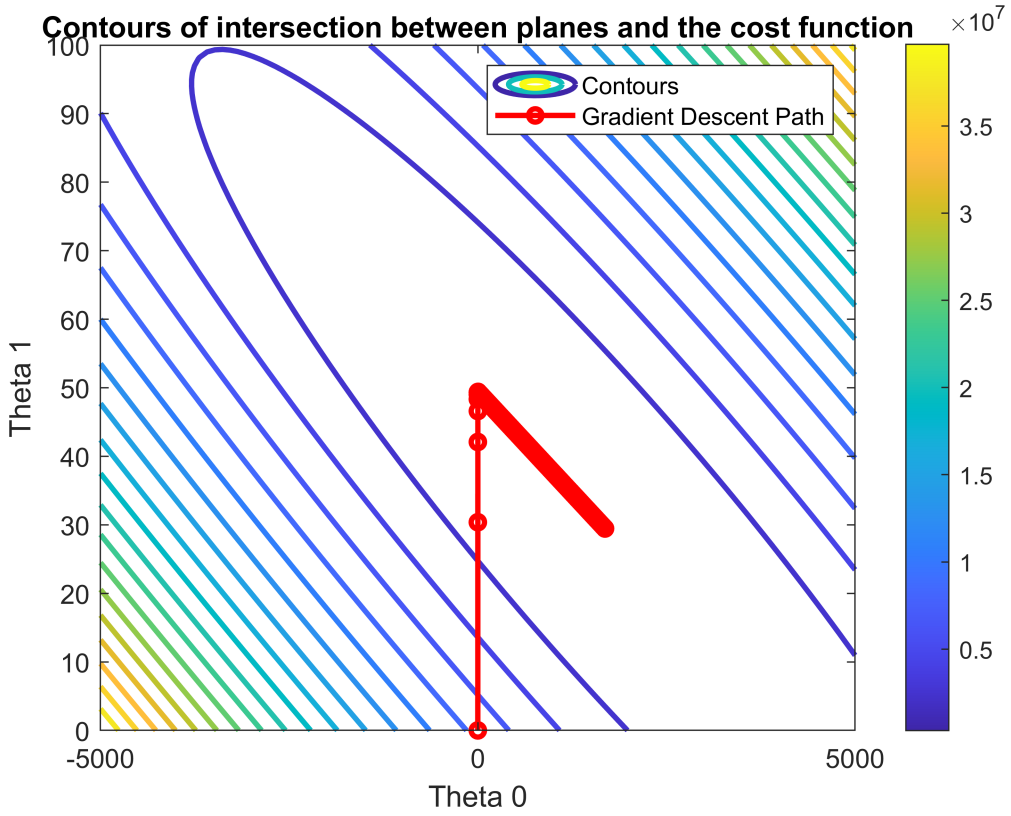


When we perform gradient descent on this landscape, it's like watching a hiker trying to find the lowest point in the valley by always moving in the direction of steepest descent. The path they take would be represented on the contour plot and can provide insights into the efficiency and behavior of the optimization process.

Let's have a look how our theta-values perform with the update function we defined earlier. The path of our values should move to the lowest point of the cost function, which is represented in the very center of the innermost 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;
```



Since alpha has a value below zero, the increments (or steps) that are made with each iteration become smaller.

Have a look at the learning rate used in this example. Do you think gradient descent is suitable for this case?

Before we move on to the next method, discuss what you've learned about gradient descent with your seat neighbor.

Method 2: Normal Equation

The second method we'll learn is called Normal Equation. The Normal Equation is rooted in the idea of finding the "best fit" by minimizing the sum of squared errors between the predicted and observed values, which forms the basis for the least squares method.

The formula looks like this:

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

Unlike gradient descent, the Normal Equation doesn't require feature scaling, as it solves for θ analytically.

Derivation of the formula

Let's derive the normal equation formula together to get a better understanding. First we need a cost function, which is the same as with the gradient descent method since we're rooting for linear regression using the least squares method:

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

In matrix notation, the hypothesis $h_\theta(x)$ can be expressed as $X\theta$ where X is an $m \times (n+1)$ matrix containing all the feature vectors $x^{(i)}$ (each one augmented with a 1 for the bias term θ_0) and θ is an $(n+1) \times 1$ column vector of parameters. Similarly, y is an $m \times 1$ column vector containing all the labels $y^{(i)}$.

Important to understand: since we're rooting for a linear combination notation:

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

and θ_0 has no dependence to the input parameters $x^{(i)}$ we need to implement the additional column of ones into the matrix in order for θ_0 to be mathematicaly solveable. The matrix in the case of one input parameter will look like this:

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

Please note that the θ-vector notation contains all coefficient parameters. In our case:

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

The Term $h_0(x) - y$ can thus be written in matrix form as $X\theta - y$. This is an $m \times 1$ vector where each element is $h_\theta(x^{(i)}) - y^{(i)}$.

The cost function can now be written as:

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

The last equality comes from the definition of vector-matrix multiplication.

Next, we want to minimize $J(\theta)$, so we set its gradient equal to zero:

$$\nabla J(\theta) = 0$$

The gradient of $J(\theta)$ with respect to θ can be calculated as:

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

Setting $\nabla J(\theta) = 0$ and solving for θ:

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

$$X^TX\theta = X^Ty$$

$$\theta = (X^TX)^{-1}X^Ty$$

This is the Normal Equation formula for finding θ that minimizes $J(\theta)$.

Let's visualize the normal equation method now.

Visualization of Normal Equation

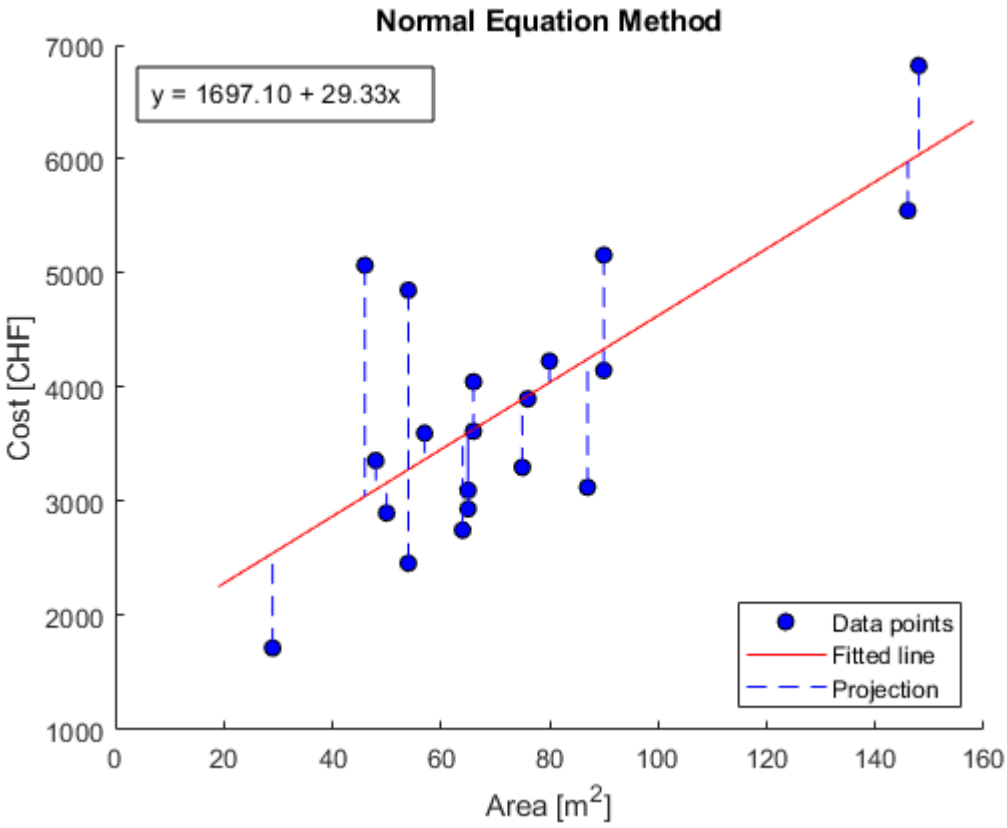
Run the code below. It will generate the known scatter plot of the data you've already worked with.

With the normal equation method (see row 104) we'll generate the best fitting approximation function for our data. The minimized error is visible as the projection of point to the prediction.

Note for own future implementations: in row 104 we have to inverse the matrix-multiplication term X^TX. We use the 'pinv' command which is the Moore-Penrose pseudoinverse, which is more numerically stable than using 'inv' for inversion.

```
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

Let's have a look at advantages and disadvantages regarding the normal equation method:

Advantages:

- **No Need for Hyperparameter Tuning:** Unlike gradient descent, which requires you to choose an appropriate learning rate, the normal equation doesn't require this kind of tuning. You get the optimal parameters in one step.
- **No Iterations:** The normal equation gives a direct, closed-form solution to finding the optimal parameters, making it computationally simpler in that sense.

- **Ease of Implementation:** The algorithm is straightforward to implement as it only involves matrix operations (inversion, multiplication).

Disadvantages:

- **Computational Complexity:** The normal equation involves inverting a $(n+1) \times (n+1)$ matrix, where n is the number of features. The matrix inversion is an $O(n^3)$ operation, making the normal equation inefficient for datasets with a large number of features.
- **Lack of Flexibility:** Unlike gradient descent, which can be used for a variety of loss functions and can be adapted easily for various kinds of regression (like ridge or lasso), the normal equation is specifically derived for least squares linear regression and is less flexible.
- **Memory Requirements:** You need to store the matrices used in the calculations, which can consume a significant amount of memory for large datasets.

so, when do I use Normal Equation?

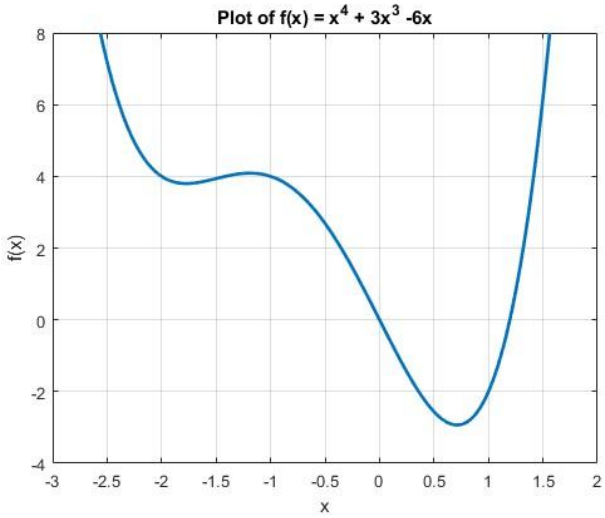
When to use

1. **Small n, Large m:** If you have a small number of features and a large number of samples, the Normal Equation can be computationally efficient and easier to implement.
2. **Exact Solution:** If you want the exact solution, and the inverse of $X^T X$ exists, the Normal Equation will provide that without requiring you to set a learning rate or iterate.
3. **Linear Regression:** It's important to note that the Normal Equation is specific to linear regression models and is not applicable for logistic regression or other types of generalized linear models.

Tasks regarding Theory of Gradient Descent and Normal Equation

Solve the following tasks with you seat neighbour:

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 we've worked on, which method do you think is more suited and why? Consider factors like the size of the dataset and the presence (or absence) of outliers.
5. Imagine you'd have to find the global minima of the function below. Which method of the above would you choose and why?



Multi-dimentional linear regression

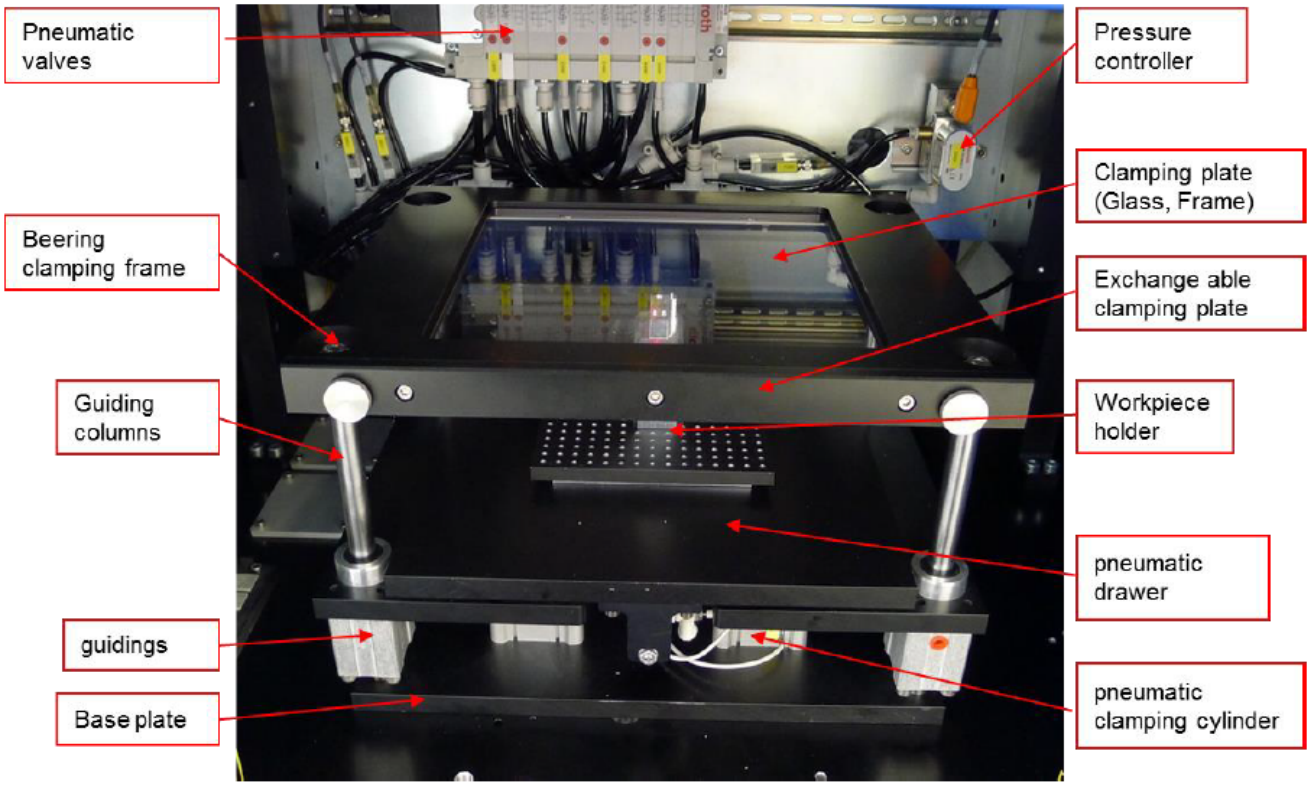
In the example above you learnd the basics of linear regression. However, oftentimes the phenomena under consideration is multi-dimensional and adding more variables could lead to a better aproximation. Lets have a look at the main topic of today: Laser transmision welding

Laser Transmission welding

From your lecture you have learned the process of laser transmission welding. For welding two plastic one part should be transparent and the other should absorb the light energy. The strength of the weld seam is dipendent on the Laser intensity: $I_p = \frac{Power}{Spot\ \ size}$ and laser-polymer interaction time which is defined by $T = \frac{Length\ \ of\ \ the\ \ weld\ \ seam}{Speed}$. We can introduce the laser energy density by combining the laser power density (I_p) and the laser-polymer interaction time (T):

$$I_E = I_p \times T$$

If the energy is sufficient the absorbent will melt and due to conduction and convection the weld seam is generated. Contact is a very important parameter in laser transmission welding. Since the conduction is done by contact. The contact between the surfaces is provided by the clamping device in the machine. You could adjust the pressure of the pneumatic cylinders to increase or decrease the clamping pressure.



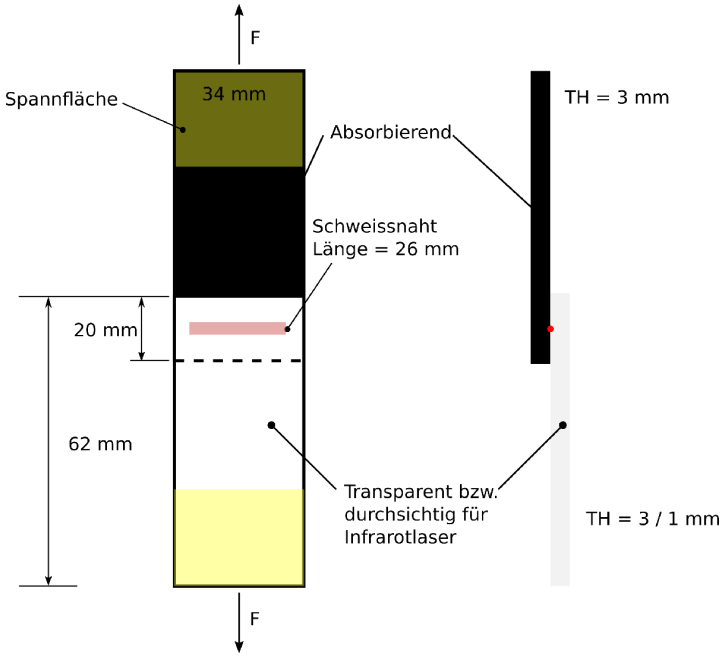
Front view of the clamping device.

Another important parameter is the thickness of the substrates. The transparent substrates available to this workshop are either 1 mm or 3 mm thick (look at the lap shear specimen in this section). The thickness of the substrates changes the pathway to the absorbent and thereby the spot size varies. How do you think this could affect the weld seam ?

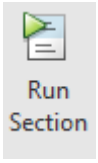
Your answer ...

Input date

We already performed some tensile shear tests. The substrates are 34 mm x 62 mm. The overlap length is 20 mm. The weld seam is set in the middle of the overlap area and is 26 mm long. The tests are performed with the 50 kN tensile test machine at the IWK test lab.



lets take a look at these data. Click on the



"Run Section" button to run the code bellow. You need to do this per each section in the future. Sections are seperated by a blue line.

```
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

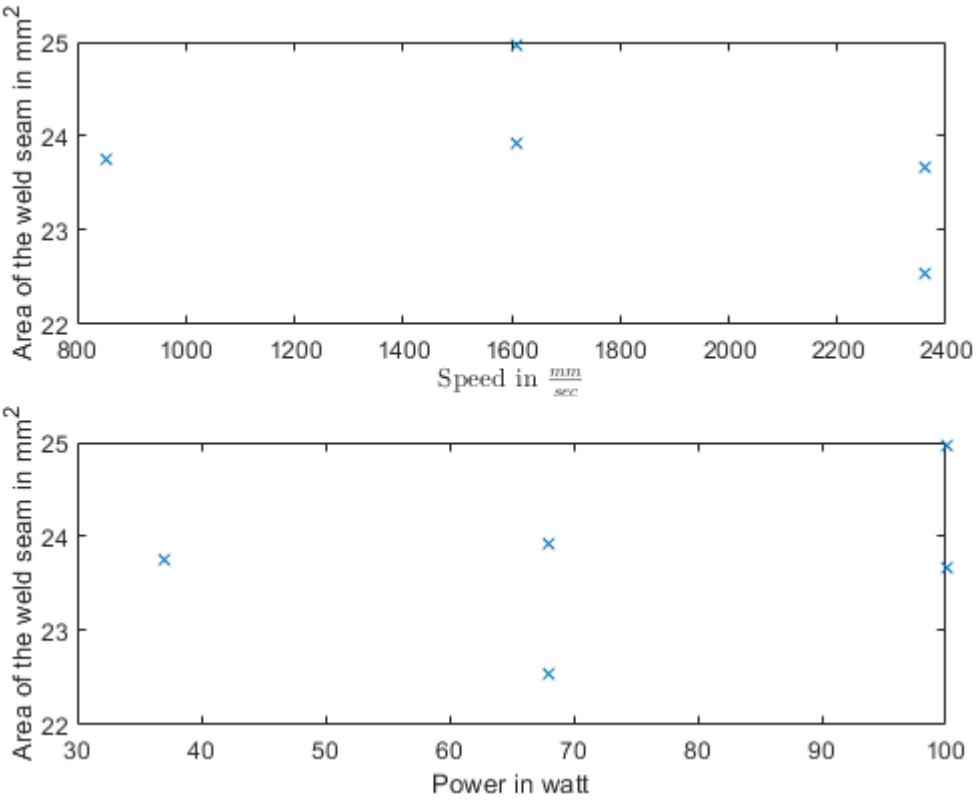
	Power	Speed	Pressure	TH	Weld_Seam_Area	Failure_Force	Strength
1	68	2362	1	3	22.5333	395.1100	17.5345
2	37	854	1	3	23.7467	501.5500	21.1209
3	68	1608	1	3	23.9200	521.2300	21.7906
4	100	2362	1	3	23.6600	587.9100	24.8483
5	100	1608	1	3	24.9600	660.8700	26.4772

Scroll through the table and try to get a feeling about the data.

We aim hier to predict the weld seam area by deploying the power and speed.

Plotting the data helps you to understand the correlations between the different parameters. Put your cursor in the gray box below and run the secion.

```
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")
```



What is your interpretation of this data? Use the table above and write your answer bellow.

Your answer ...

Why at some points we see same speed or same power, however the area of the weld seam varies ? Use the table above.

Your answer ...

Which trends could you recognize ?

Your answer ...

Let us define a hypothesis that predicts the weld seam area as a linear function of power and speed.

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

in other form:

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

x1 and x2 present the power and speed. Since we want to use the vectorization ability of Matlab the x1 and x2 are summarized into the matrix X. Later we will add a column of ones at the beginning (line 25 of the code).

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

After the code section you will see the first five elements of the table in the order of "[power speed], area". Chek if this applies.

```
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 you will fitt the hypothesis to the dataset using gradient descent.

Before we start witht the gradient descent we need to define what are we going to optimize. Let us define the error as the absolut value between the hypothesis and the real size of the weld seam. The cost function is defined as the sumation of the quadratic errors between the hypothesis and the dataset:

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

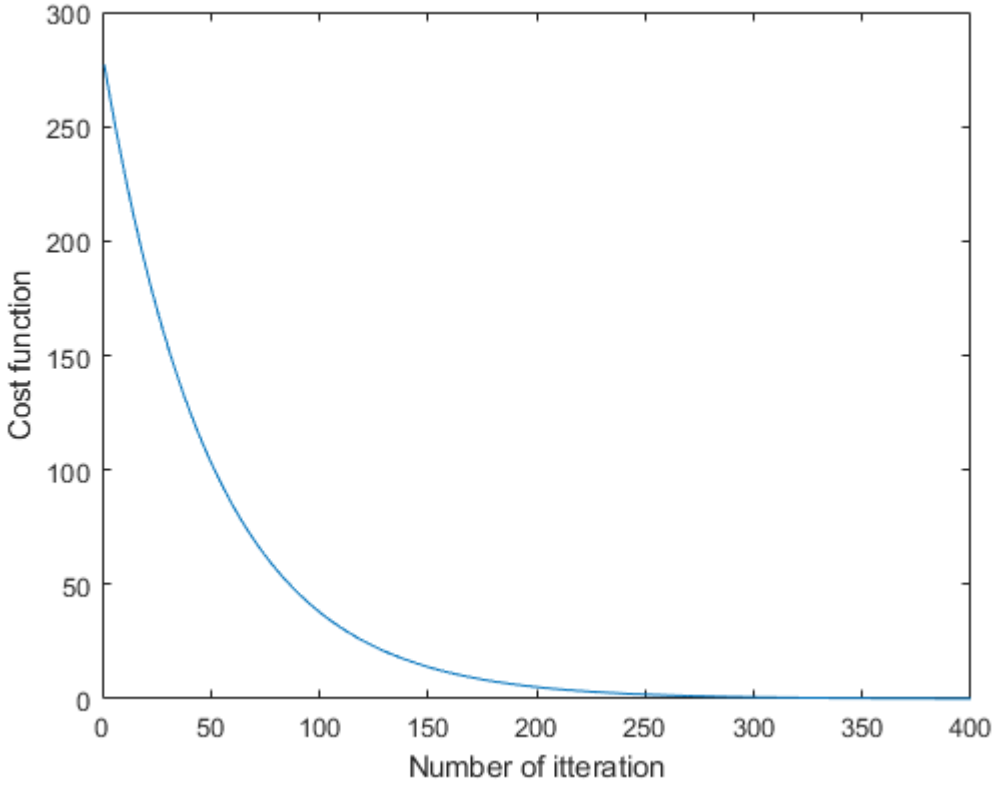
Remember that the parameters of the model are θ. We aim to adjust these parameters to minimize the cost $J(\theta)$. From your math course you have learned how to calculate the gradient of an equation. By solving these equations you could directly calculate the parameters of the system. We will introduce this method later.

Another way is to use the gradient descent algorithm. In gradient descent, first, we guess, what would the answer be. From that point, we simultaneously update our guess to the point that the cost function reaches a minimum. The minimum is not necessarily the global minimum.

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

With each update, the parameters come closer to the optimal value, which will lead to the lowest cost. is the learning rate. Too small a learning rate will slow down the calculation and a large learning rate leads to divergence. One way to see if the learning rate is appropriate is by plotting the cost function. We expect that the cost function decreases steadily. The cost function should never increase. If you find an optimum you should not keep updating the parameters further, since this could also lead to instabilities. Choose different alpha from the drop-down menu to see the influence of alpha on cost function.

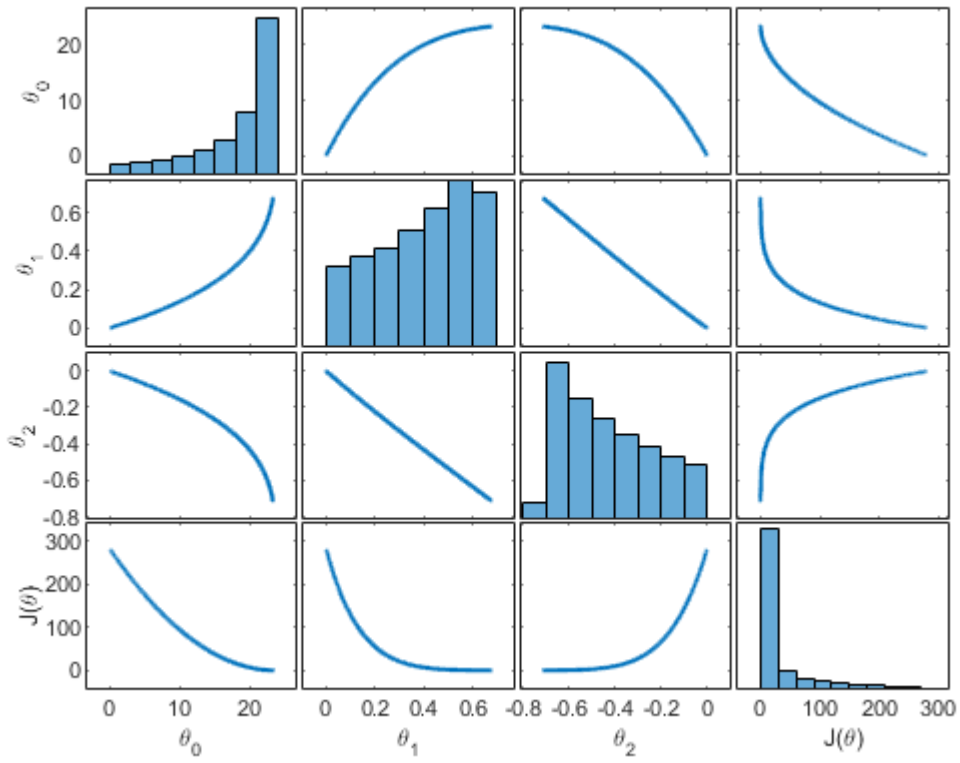
```
theta_g = zeros(width(X),1); % initialize fitting parameters
iterations = 400;
alpha = 0.01;

% The fuction "gradientDescent" takes the input data, first guss of theta [0 0 0], learning rat
% This function returns theta from the last iteration and the history of theta and cost functio
% The response of gradient descent is affected by the first guess, as you learnd in the previou
[theta_g,theta_hist,J_hist] = gradientDescent(X, y, theta_g, alpha, iterations);

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



What should be the learning rate in your opinion ? Why?

In this particular case, $J(\theta)$ is a four-dimensional function. As humans, we are incapable of imagining a four-dimensional space. However, we could present the dependencies of the parameters in a matrix plot. Try to understand the matrix plot below. If any difficulties arose contact your supervisor. At the end of the 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
```



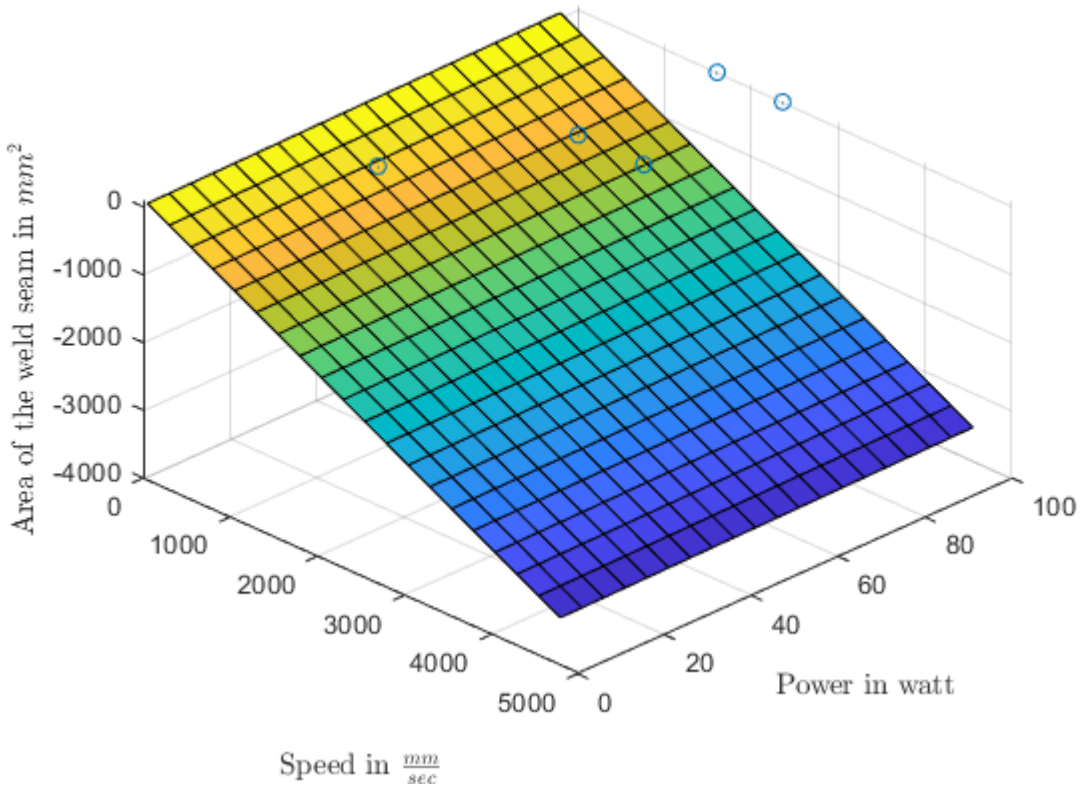

```
fprintf("Parameter vector computed from gradient descent: "); theta_g
```

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

Now that we have the coefficients of the hypothesis, we can plot it to gain a better intuition of our hypothesis. The coefficients of the hypothesis should be $\theta_1 = (23.3374 \quad 0.6740 \quad -0.7092)$ and the hypothesis will be similar 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 power and the speed of the weld process. you can plot the predicted hypothesis per each iteration of the gradient descent by runing the code bellow.

```
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
```



What do you think is happening here? What do you understand from this diagram ? discuss with your teammates.

Does the hypothesis do what you expected ?

Hint: The last iteration of the gradient descent gives us the optimum parameters.

Your answer ...

What is the prediction of this hypothesis from the area of a weld seam, which is made by a laser with a power of 50 watts and a speed of 850 mm/sec ? Fill in the code. Discuss the answer with your teammates.

```
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 you could see in the above section the gradient descent could not find the global optima. This lies in the fact that the answer to the gradient descent depends on the first guess, the learning rate, and the data available. The location of the first guess along with the learning rate defines the pathway from the first guess to the next optima. The gradient descent is suitable for large data, where calculating the inverse of huge dense matrixes is very time-consuming. In those cases, since the available data is immense the predictions would be better. However, in our case, we can introduce a direct method, which could find the absolute minimum of the cost function and thereby the parameters of the hypothesis.

Normal Equation

If you calculate the gradient of the cost function per each direction θ and solve the equations, you will be able to calculate the θ directly, which is called the closed-form solution to the linear regression:

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

In this method there is no loop to find the optimum, rather we calculate directly the parameter vector θ in one go.

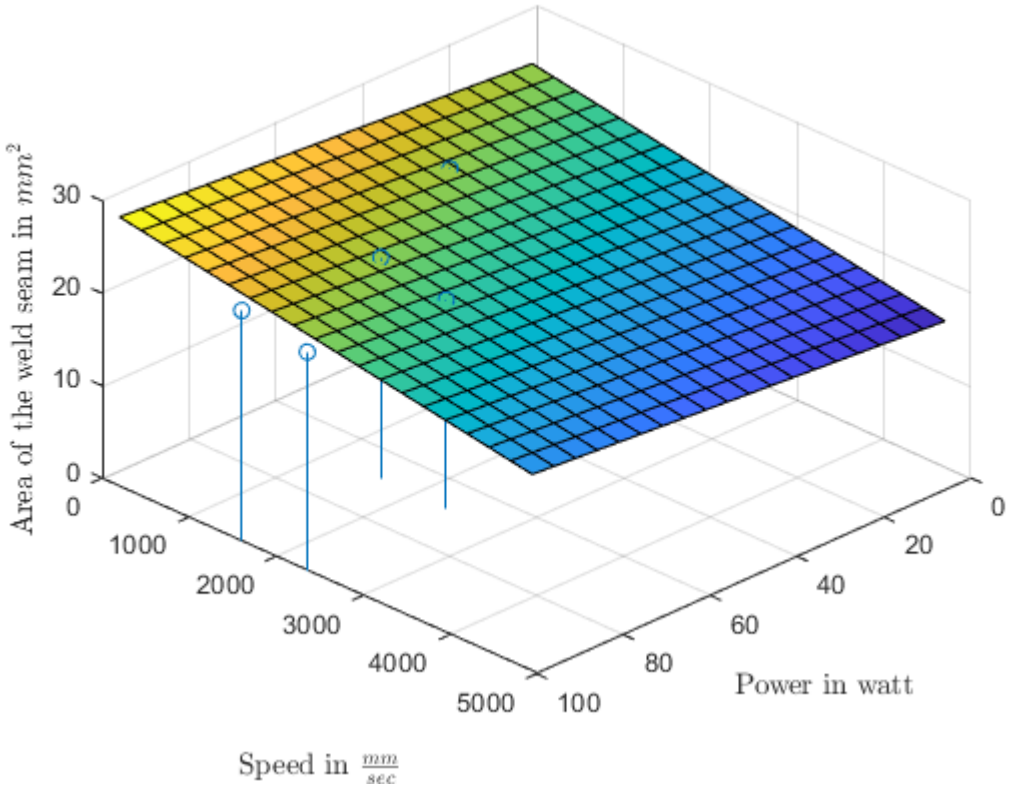
```
% normal equation
X = [x1, x2];
y = T.Weld_Seam_Area;
m = length(y);

% 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 gradient descent: "); theta_n
```

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

Let us plot this hypothesis and see if the results differ form the gradient disscent method.

compare the hypothesis from gradinet disscent with the normal equation.

```
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
```

If you have done everything correctly you should see the area of $24.3mm^2$.

Task:

1. Predict the weld strength by using the same data set.
2. Consider you want to add the thickness of the substrate in the prediction. Where should we change the script? Remember that the graphs are just for visualization purposes. therefore you can neglect them.

Discuss with your teammates and write down your answer below. Your Supervisor will evaluate your answer.

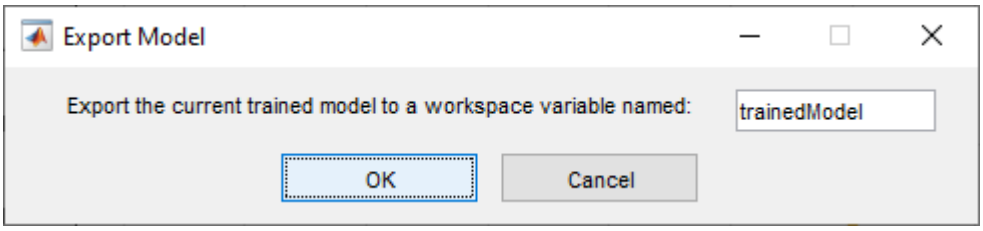
3. Make a lap shear specimen and test the tensile shear strength of the specimen. Does the result of your test match the AI prediction? Add your results at the end of the "Training_Data.xlsx". This will help us to improve the gradient descent.
4. If you would apply the physical background of laser transmission welding, how would you change the hypothesis?
5. How can we improve this model?

Optional task:

Linear Regression with Matlab AI APP

You have learned how linear regression operates. Now it is time to use the Regression Learner from MATLAB.

In the ribbon "Apps" you can find a drop-down menu. There is a section named "Machine Learning and deep learning". Open "Regression learner", and create a new session. You will be asked for the data set. Choose "T" from the drop-down menu. Choose the weld seam area as the response, and untick the unnecessary predictors. At the end start the session. From the MODEL TYPE choose the linear model. Train your model and Export the model by clicking on the green tick in the upper right corner of the regression learner window.



When the above window appears, do not change the neme of the model and just click on OK. after that run the sessoin.

```
linMdl = trainedModel.LinearModel;
```

```
Unable to resolve the name trainedModel.LinearModel.
```

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

Did you get the same result as your calculation ?

In the end, we hope you could get started with artificial intelligence. Remember that it is not necessary to know every detail about AI. However, it is important that you do not treat it as a black box.