MATLAB / Simulink Lab Course

Optimization & Statistics Toolbox

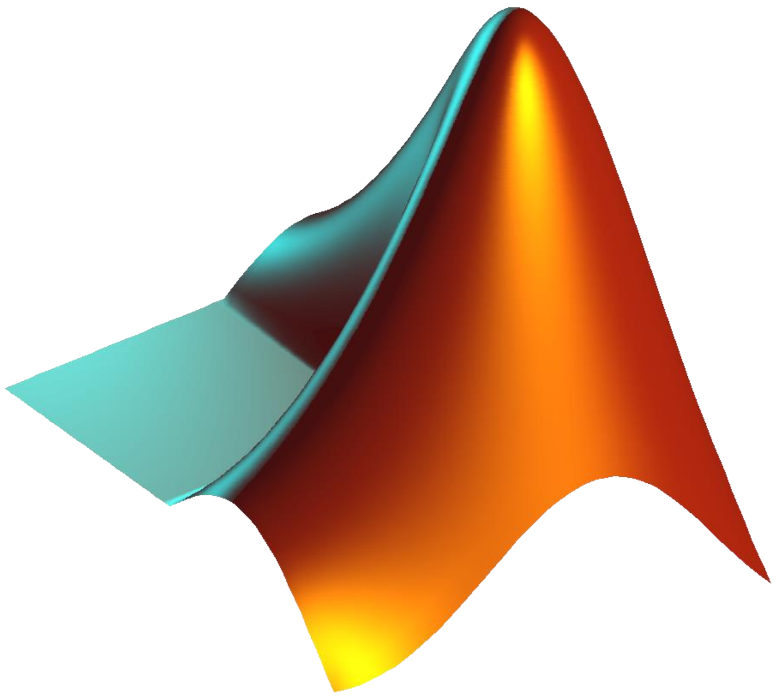

Table of Contents

1 Minimization of the Himmelblau Function (8 Points)

In this tutorial we consider the so-called Himmelblau function (named after David Mautner Himmelblau). The Himmelblau function is a good test function for optimization algorithms since it has four local minima and one local maximum. It is defined as:

$$f_H(\boldsymbol{x}) = ({x_1}^2 + x_2 - 11)^2 + (x_1 + {x_2}^2 - 7)^2$$

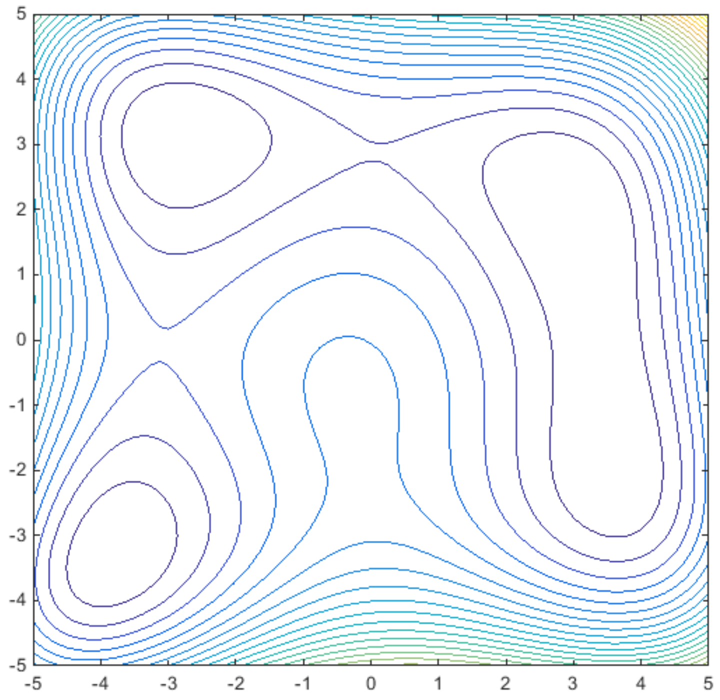

Figure 1-1: Contour plot of the Himmelblau Function

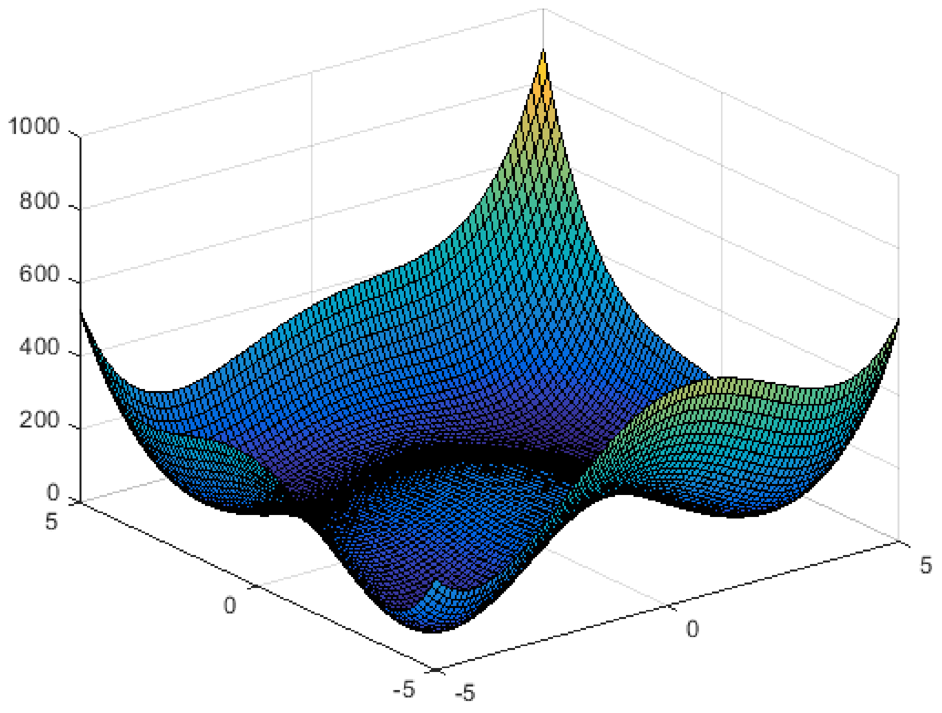

Figure 1-2: Surface plot of the Himmelblau Function

1.1 Implementation of the Himmelblau Function (1.5 Points)

Implement the Himmelblau function in a Matlab function `himmelblau`. For efficient evaluation, the function shall accept array arguments `x1` and `x2` (with compatible dimensions).

The function shall return the corresponding array `y` of Himmelblau function values. Take advantage of vectorized operations.

1.2 Plots (2 Points)

Get familiar with the MATLAB functions `contour` and `surf` and generate plots of the Himmelblau function as indicated in Figure 1-1 and Figure 1-2.

Hint: Use `ndgrid` to generate a grid of evaluation points. Avoid MATLAB loops.

1.3 Unconstrained Optimization (2 Points)

Use a suitable Optimization Toolbox routine to find local minima of the Himmelblau function for the following initial values:

a. X0 = [-3 , -3]
b. X0 = [-2 , 2]
c. X0 = [3 , 1]
d. X0 = [3 , -2]

Mark the optimal points in the contour and surface plots from the previous task.

1.4 Gradient (1.5 Points)

a. Calculate the gradient of the Himmelblau function manually (i.e. on paper; MATLAB capabilities regarding Symbolic Math will be discussed in the next session) and implement the result as a Matlab function `himmelblau_gradient`. The gradient function shall support the same input arguments as the previously implemented Himmelblau function and return the gradient vectors at all input points as column vectors, concatenated horizontally.
b. Apply the gradient function to all the results from exercise 1.3. What can be observed? *Hint: Write a short comment in your solution script.*

1.5 Finding the Local Maximum (1 Point)

Find a way to use the `fminunc` function to locate the local maximum of the Himmelblau function and plot the result.

Hint: Use initial value X0 = [-2 , 2]

2 Statistical Analysis of a Dataset (8 Points)

2.1 Maximal Horsepower (2 Points)

a. Load the MATLAB sample dataset `carbig.mat` and find the maximal horsepower (`Horsepower`) in the dataset.
   Hint: This dataset is available within a standard MATLAB installation and the `load()` function will find it automatically.
b. What is the manufacturer (`Mfg`) and model (`Model`) of the car with the maximal horsepower in the dataset?
c. How many models in the dataset have the maximal horsepower?

2.2 Chevrolets (1 Point)

How many Chevrolets are in the dataset?

Hint: Transferring the character array `Mfg` into a cell array might be beneficial.

2.3 Boxplots (3 Points)

a. Compare the boxplots of weights (`Weight`) for Chevrolet and Volkswagen using subplots.
b. Get familiar with the MATLAB function `quantile`. Calculate the median and the 75 % quantile for the weights for Chevrolet and Volkswagen cars and compare it with the boxplots.

2.4 Fitting Distributions (2 Points)

a. Fit a Normal distribution to the horsepower data of all Chevrolets using the MATLAB function `fitdist`.
b. In addition, fit a Weibull distribution to the same data as in a. Get familiar with the MATLAB function `negloglik`. Which distribution fits the data better?

3 Calibration of the Arduino Robot (4 Points)

Within this MATLAB/Simulink Lab Course, a maneuvering Arduino Robot is being programmed which will follow a given track at the end of the course.

In this exercise, we use statistical analyses to find some important calibration parameters that will be used in the subsequent tutorials to program the robot.

The following variables are relevant in this exercise and can be found in `arduino_data.mat`:

- `propulsion_first_gear`
  This is the gear ratio of the engines to the first following gearwheel.

- `delta_t_engine1` and `delta_t_engine2`
  In specific gearwheels of the Arduino, there are holes through which light can pass. Sensors measure the time difference in seconds between two light signals for each engine (which depends on the speed) and they are stored in the considered variables.

- `light_measurement_engine1` and `light_measurement_engine2`
  These are values between 0 (no light) and 1024 (full light) describing the intensity of the light shining through the holes of the gearwheel.

- `sensor_left_white`, `sensor_right_white`, `sensor_left_black`, and `sensor_right_black`
  The Arduino is equipped with two underground sensors (left and right) observing the ground and their values are recorded in the considered variables.

3.1 Mean Time Difference between Two Light Signals (1 Point)

Load the required data `arduino_data.mat`. Calculate the two mean values of the vector `delta_t_engine1` and `delta_t_engine2`.

3.2 Increase of Engine Rotation Speed per Unit Engine Control Number (1 Point)

The engine can be controlled by a specific number between 0 (no power) and 255 (full power). In this exercise, we are interested in the increase of engine rotation speed for an increase of the engine control number of unit 1 (assuming linear behavior).

a. Calculate the rotation speeds of both engines according to the formula:

$$rotation\_speed = \frac{2 \cdot \pi}{4 \cdot mean\_delta\_t \cdot propulsion\_first\_gear}$$

The multiplication by 4 in the denominator comes from the 4 holes in the gearwheels, which lead to four light pulses per revolution.

b. Since we assume a linear relationship, we can calculate the variable of interest for both engines individually via the following formula:

$$k = \frac{rotation\_speed}{255}$$

3.3 Calibrating Engine Light Sensors (1 Point)

Similar to the last exercise 3.2, we are interested in calibration parameters for the light sensor at the gearwheels. Get familiar with the MATLAB function `round`. The parameter of interest for each engine light sensor can be calculated as the rounded mean of all engine light measurements.

3.4 Calibrating Underground Sensors (1 Point)

For the subsequent tutorials it is important to know which values are recorded by the sensors for white and black underground to be able to let the Arduino follow a given track. Calculate those parameters based on the measurements `sensor_left_white`, `sensor_right_white`, `sensor_left_black`, and `sensor_right_black`. Thereby, the same procedure considering the rounded mean values like in exercise 3.3 should be applied.

4 Nonlinear Least Squares (7 Points)

This task allows you to make use of your newly-developed skills in exploratory data analysis and nonlinear optimization.

4.1 Load the Dataset (0.5 Points)

Load the example dataset from `nls_example_dataset.m`.

4.2 Explore the Dataset and Define a Model Structure (2 Points)

Use the visualization features discussed in the lecture to inspect the dataset. The goal is to express `z` as a function of `x` and `y`. Identify a suitable function structure and implement it as a vectorized Matlab function handle of the form `model = @(x, y, a, b, …)` … where `a`, `b`, etc. are the parameters of your model (you can also pass a single parameter vector instead). Guess the model parameters from the plot (in many cases the correct order of magnitude should be sufficient).

Hint: For this exercise, a fairly simple model with two to four parameters is totally sufficient.

4.3 Implement the Residual Function (1 Point)

Implement a function handle `residual = @(p)` … that generates a residual vector (!) from all samples in the dataset. This residual function accepts a vector (!) of parameters.

4.4 Identify the Optimal Parameters (2 Points)

Apply the `lsqnonlin` solver to fit your model to the dataset. You may need to use `optimoptions` to increase the maximum number of function evaluations or maximum number of iterations. Also, it is advisable to set the `Display` option to `'iter'` for more detailed solver output. Adjust the initial guess for the parameter vector. You should be able to observe multiple local minima, depending on the starting point.

4.5 Visualize the Fitted Surface (1.5 Points)

For visual inspection of the fit quality, add a surface or mesh plot of the model output. Use `ndgrid` to generate a suitable grid of evaluation points. The surface should be very close to the original data points.