MATLAB / Simulink Lab Course

Optimization & Statistics Toolbox

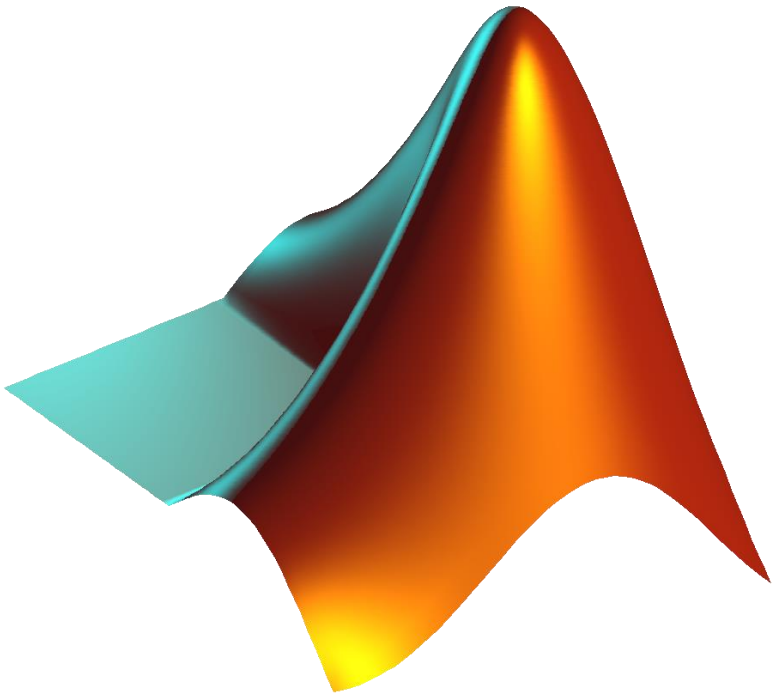

Table of Contents

1 Guidelines

- **Use the provided templates** and do not change function interfaces, file names or variable names. *This restricts your creative freedom, but streamlines the grading process. Since there are so many course participants, this is crucial.*
- **Use expressive names** for functions and variables. *This helps with debugging and grading.*
- **Ensure correct control flow and causality**. *Capture the expressions in variables instead of hardcoding intermediate results. Use conditional statements instead of hardcoding conclusions.*
- **Write generic code**, i.e., do not hardcode dataset/parameter sizes, indices or even values. *For example, when analyzing datasets, write code that still produces the right result if the data is shuffled, resized or otherwise updated. Define functions to avoid code duplication.*
- **Write code for humans, not just for the computer.** *Put spaces around operators. Write only one statement per line. Use blank lines to group closely related statements. Properly indent blocks of code. In short: Make hitting the space bar a habit.*
- **Make sure that your code works when run in a clean workspace.** *Before submission, clear all variables and rerun your solutions for every task to ensure that the code does not depend on results from earlier runs.*

2 Himmelblau Benchmarking Function (9 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:

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

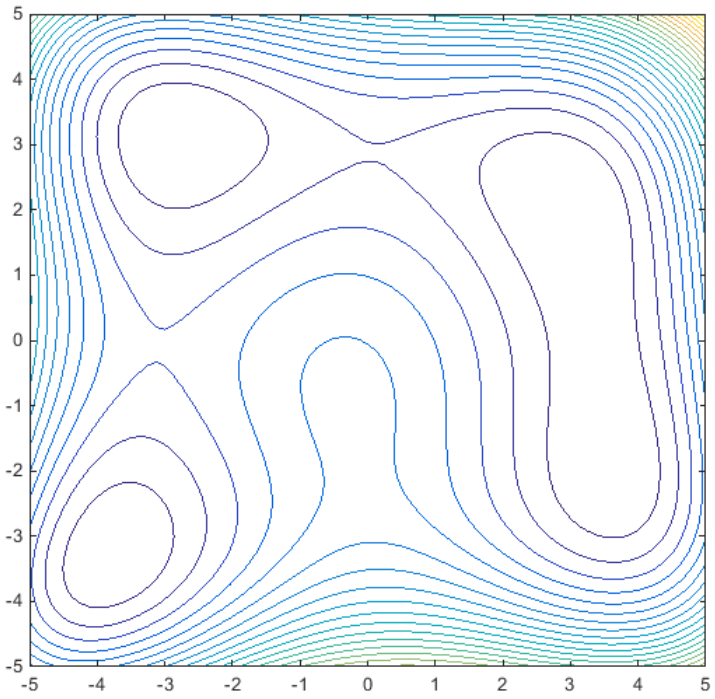

Figure 2-1: Contour plot of the Himmelblau Function

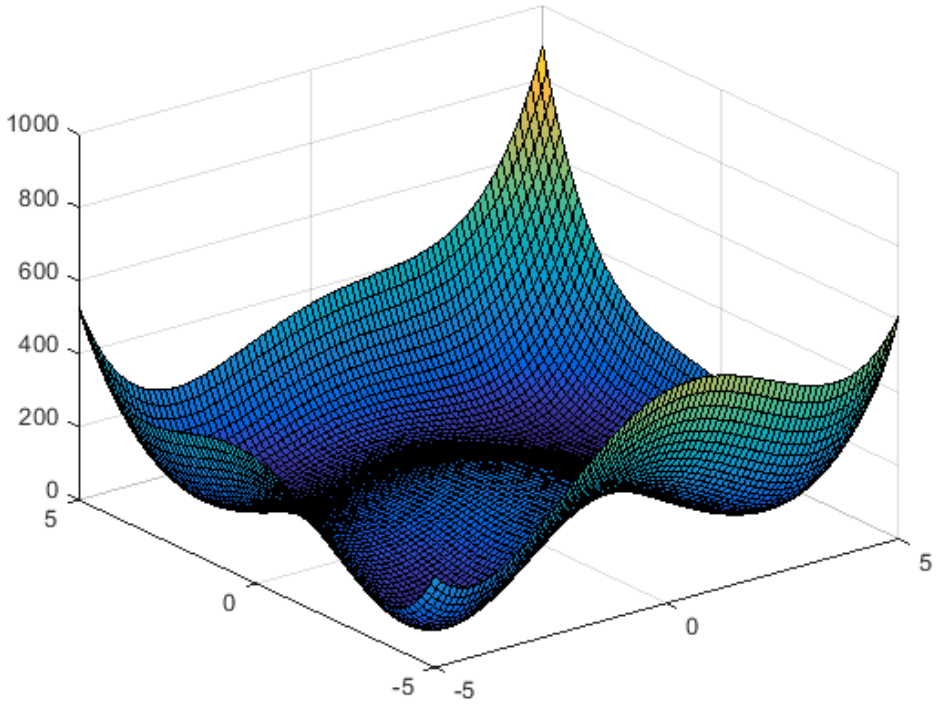

Figure 2-2: Surface plot of the Himmelblau Function

2.1 Implementation of the Himmelblau Function (2 Points)

In `himmelblau.m`:

- Implement the Himmelblau function.
- For efficient evaluation, the function shall accept array arguments `x1` and `x2` (with compatible dimensions).
- Use vectorized operations.

2.2 Visualization (2 Points)

In `main.m`:

Use `contour()` and `surf()` to generate plots of the Himmelblau function as indicated in Figure 2-1 and Figure 2-2.

Hint: Use `ndgrid()` to generate a grid of evaluation points. Avoid Matlab loops. You do not need to reproduce exactly the style of the figures.

2.3 Unconstrained Optimization (2 Points)

In `main.m`:

- Use a suitable Optimization Toolbox routine to find local minima of the Himmelblau function for the following initial values:
  a. $\boldsymbol{x}_1 = [-3, 3]^T$
  b. $\boldsymbol{x}_2 = [-2, 2]^T$
  c. $\boldsymbol{x}_3 = [3, 1]^T$
  d. $\boldsymbol{x}_4 = [3, -2]^T$
- Mark the optimal points in the contour and surface plots from the previous task.

2.4 Constrained Optimization (2 Points)

In `main.m`:

- Implement the constraint $\left(\frac{x_1-5}{2}\right)^2 + (x_2 - 4)^2 \leq 1$ as a function handle.
- Use a suitable Optimization Toolbox routine to find the minimum of the Himmelblau function under this constraint.
- Mark the point in the plots.

2.5 Finding the Local Maximum (1 Point)

In `main.m`:

- Use a suitable Optimization Toolbox routine to find the local maximum of the Himmelblau function.
- Mark the point in the plots.

Hint: Use the starting point $\boldsymbol{x}_0 = [-2, 2]^T$.

3 Model-Predictive Control (MPC) of a Linear System (15 Points)

In this exercise, we seek the optimum input time history to control a discrete-time linear dynamic system over a given lookahead horizon, considering a quadratic performance index as well as linear state/control/output constraints.

Note: Here, you need to reformulate an engineering problem in a standard form compatible with the generic mathematical optimization framework. No specific background knowledge in optimal control is required.

3.1 Dynamic System

We consider a controllable linear time-invariant discrete-time dynamic system with states $\boldsymbol{x} \in \mathbb{R}^{n_x}$, controls $\boldsymbol{u} \in \mathbb{R}^{n_u}$, outputs $\boldsymbol{y} \in \mathbb{R}^{n_y}$ and disturbances $\boldsymbol{v} \in \mathbb{R}^{n_v}$:

$$\boldsymbol{x}[k+1] = \boldsymbol{A}\,\boldsymbol{x}[k] + \boldsymbol{B}\,\boldsymbol{u}[k] + \boldsymbol{E}\,\boldsymbol{v}[k], \quad \boldsymbol{y}[k] = \boldsymbol{C}\,\boldsymbol{x}[k] + \boldsymbol{D}\,\boldsymbol{u}[k]$$

Here, k denotes the time step.

3.2 Optimal Control Problem (OCP)

Assuming the current state of the system is $\boldsymbol{x}_i$, we consider a lookahead horizon of $N-1$ steps (i.e., N samples including the current state). The commanded output $\boldsymbol{y}_c[k]$ is given for all samples $k \in \{1, \dots, N\}$ within the lookahead horizon.

We formulate the discrete-time MPC tracking problem as a Quadratic Program that minimizes the sum of the output tracking error (weighted by $\boldsymbol{Q} = \boldsymbol{Q}^T \geq 0,\ \boldsymbol{Q} \in \mathbb{R}^{n_y \times n_y}$) and a constraint violation penalty (weighted by $\mu \geq 0$):

$$\min_{[e, \boldsymbol{x}[k]\dots\boldsymbol{x}[N], \boldsymbol{u}[1]\dots\boldsymbol{u}[N]]} \left(\frac{1}{2}\mu\, e^2 + \frac{1}{2}\sum_{k=1}^{N} (\boldsymbol{y}[k] - \boldsymbol{y}_c[k])^T\, \boldsymbol{Q}\, (\boldsymbol{y}[k] - \boldsymbol{y}_c[k]) \right)$$

$$\begin{aligned}
\text{subject to} \quad & \boldsymbol{x}[1] = \boldsymbol{x}_i \quad \text{(initial state)} \\
& \boldsymbol{x}[k+1] = \boldsymbol{A}\,\boldsymbol{x}[k] + \boldsymbol{B}\,\boldsymbol{u}[k], \quad k \in \{1, \dots, N-1\} \quad \text{(dynamics)} \\
& \boldsymbol{x}_{\text{lb}} - e \leq \boldsymbol{x}[k] \leq \boldsymbol{x}_{\text{ub}} + e \quad \forall k > 1 \quad \text{(relaxed state bounds)} \\
& \boldsymbol{u}_{\text{lb}} \leq \boldsymbol{u}[k] \leq \boldsymbol{u}_{\text{ub}} \quad \forall k \quad \text{(control bounds)} \\
& \boldsymbol{y}_{lb} - e \leq \boldsymbol{y}[k] \leq \boldsymbol{y}_{ub} + e \quad \forall k \quad \text{(relaxed output bounds)} \\
& 0 \leq e
\end{aligned}$$

Remarks:

- *For illustration, we include the dynamics as a collocation constraint, which increases the problem size. Instead, we could also explicitly calculate the sensitivity of the state history.*
- *For convenience, we introduce the initial state and the final control as decision variables, although the former is constant and the latter only has an effect if $\boldsymbol{D} \neq \boldsymbol{0}$.*
- *If we omit e, the state/control/output bounds are hard constraints. This can make the problem infeasible in case of disturbances. With the penalty approach, the problem remains feasible at the expense of (small) constraint violations.*
- *We assume that the initial state is known and the disturbance is unknown. The problem can be stated differently, for example by including an estimate of the current disturbance.*

3.3 Parameter Optimization Problem in Standard Form (10 Points)

We need to express the Quadratic Program in standard form:

$$\begin{aligned} \min_{\boldsymbol{z}} \left(\frac{1}{2}\, \boldsymbol{z}^T\, \boldsymbol{H}\, \boldsymbol{z} + \boldsymbol{f}^T\, \boldsymbol{z} \right) \\ \text{subject to} \quad \boldsymbol{C}_{eq}\, \boldsymbol{z} = \boldsymbol{v}_{eq} \\ \boldsymbol{C}_{ne}\, \boldsymbol{z} \leq \boldsymbol{v}_{ne} \\ \boldsymbol{z}_{\text{lb}} \leq \boldsymbol{z} \leq \boldsymbol{z}_{\text{ub}} \end{aligned}$$

In `createProblem.m`:

- Consider the vector of decision variables $\boldsymbol{z} = [e, \hat{\boldsymbol{x}}^T, \hat{\boldsymbol{u}}^T]^T$, where $\hat{\boldsymbol{x}} = [\boldsymbol{x}[1]^T, \dots, \boldsymbol{x}[N]^T]^T$ and $\hat{\boldsymbol{u}} = [\boldsymbol{u}[1]^T, \dots, \boldsymbol{u}[N]^T]^T$. Calculate the corresponding (sparse) Jacobians $\frac{\partial e}{\partial \boldsymbol{z}}, \frac{\partial \hat{\boldsymbol{x}}}{\partial \boldsymbol{z}}, \frac{\partial \hat{\boldsymbol{u}}}{\partial \boldsymbol{z}}$.
- Define $\hat{\boldsymbol{y}}$ just like $\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}$, so that $\hat{\boldsymbol{y}} = \hat{\boldsymbol{C}}\, \hat{\boldsymbol{x}} + \hat{\boldsymbol{D}}\, \hat{\boldsymbol{u}}$, and calculate the Jacobian $\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}}$.
- Specify the bounds for the decision variables, $\boldsymbol{z}_{\text{lb}}, \boldsymbol{z}_{\text{ub}}$.
- Reformulate the dynamic constraint (for all time steps!) in the form $\boldsymbol{C}_{eq}\, \boldsymbol{z} = \boldsymbol{v}_{eq}$.
- Reformulate the (relaxed) state and output constraints (for all time steps!) in the form $\boldsymbol{C}_{ne}\, \boldsymbol{z} \leq \boldsymbol{v}_{ne}$.
- Rewrite the objective as $\frac{1}{2}\, \boldsymbol{z}^T \left(\left(\frac{\partial e}{\partial \boldsymbol{z}} \right)^T \mu \left(\frac{\partial e}{\partial \boldsymbol{z}} \right) + \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} \right)^T \hat{\boldsymbol{Q}} \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} \right) \right) \boldsymbol{z} - \hat{\boldsymbol{y}}_c^T\, \hat{\boldsymbol{Q}} \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} \right) \boldsymbol{z} + \text{const}$, where $\hat{\boldsymbol{y}}_c$ is defined analogously. The constant term is not of interest, since it does not affect the location of the optimum. From this expression, calculate the Hessian matrix $\boldsymbol{H} = \left(\frac{\partial e}{\partial \boldsymbol{z}} \right)^T \mu \left(\frac{\partial e}{\partial \boldsymbol{z}} \right) + \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} \right)^T \hat{\boldsymbol{Q}} \left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} \right)$ and the Jacobian of the linear term with respect to the commanded outputs, $\frac{\partial \boldsymbol{f}}{\partial \hat{\boldsymbol{y}}_c} = -\left(\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} \right)^T \hat{\boldsymbol{Q}}$.

Tips:

- Rewrite the dynamic constraint in the form $\hat{\boldsymbol{A}}\, \hat{\boldsymbol{x}} + \hat{\boldsymbol{B}}\, \hat{\boldsymbol{u}} = \hat{\boldsymbol{R}}\, \hat{\boldsymbol{x}}$, then use the Jacobians of $\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}$ to obtain $\boldsymbol{C}_{eq}\, \boldsymbol{z} = \boldsymbol{v}_{eq}$. Pay attention to the dimensions, since the constraint links the states and controls at $k = 1, \dots, N-1$ to the states at $k = 2, \dots, N$. The matrix sizes are $\hat{\boldsymbol{A}} \in \mathbb{R}^{(N-1)\, n_x \times N\, n_x}, \hat{\boldsymbol{B}} \in \mathbb{R}^{(N-1)\, n_x \times N\, n_u}, \hat{\boldsymbol{R}} \in \mathbb{R}^{(N-1)\, n_x \times N\, n_x}$.
- Rewrite the relaxed state and output constraints in the form $\hat{\boldsymbol{x}}_{\text{lb}} - e \leq \hat{\boldsymbol{x}} \leq \hat{\boldsymbol{x}}_{\text{ub}} + e$ and $\hat{\boldsymbol{y}}_{lb} - e \leq \hat{\boldsymbol{y}} \leq \hat{\boldsymbol{y}}_{ub} + e$, then rearrange the inequalities to arrive at $\boldsymbol{C}_{ne}\, \boldsymbol{z} \leq \boldsymbol{v}_{ne}$.
- The Kronecker tensor product, `kron()`, is a very useful means to conveniently assemble the (sparse) matrices $\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}, \hat{\boldsymbol{C}}, \hat{\boldsymbol{D}}, \hat{\boldsymbol{R}}, \hat{\boldsymbol{Q}}$.

3.4 Solution (5 Points)

In `solveMpc.m`:

- Update the bound vectors $\boldsymbol{z}_{\text{lb}}, \boldsymbol{z}_{\text{ub}}$ according to the given initial condition $\boldsymbol{x}_i$.
- Calculate the linear objective coefficient vector $\boldsymbol{f}$ for the given command sequence $\hat{\boldsymbol{y}}_c$ using the Jacobian from the previous subtask.
- Choose a suitable solver from the *Optimization Toolbox*, pass the objective and constraint information and solve the MPC problem.
- Throw an error if the solver fails to produce a valid solution.

3.5 Simulation

Run `main.m` to simulate the MPC system for an example problem. In every time step, an optimization problem is solved to obtain the next control. Observe how the control law accounts for known command changes and constraints within the specified lookahead horizon, thereby tracking the command without any lag. Observe how model limitations are respected, up to the tolerance caused by the constraint relaxation.

4 Statistical Analysis (8 Points)

4.1 Data Import and Preprocessing (1 Point)

In `analyze_cars.m`:

- Load the dataset `carbig.mat`.
  Hint: This dataset is distributed with Matlab, the `Load()` function will find it.
- For convenience, convert all `char` arrays in the dataset into `string` arrays. Remove leading and trailing whitespace.

4.2 Basics (3 Points)

In `analyze_cars.m`:

- Find the manufacturers (field `Mfg`) and models (field `Model`) of all cars that share the minimum power (field `Horsepower`).
- Calculate the median weight (field `Weight`) by manufacturer (field `Mfg`).

4.3 Boxplots (2 Points)

In `analyze_cars.m`:

Choose two manufacturers and create side-by-side boxplots of the miles per gallon (field `MPG`) distribution of their models for comparison.

4.4 Distribution Fitting (2 Points)

In `analyze_cars.m`:

- Fit a normal distribution and a Weibull distribution to the miles per gallon (considering the full dataset).
- Use `negloglik()` (see Matlab documentation) to determine programmatically which fits the data better.

5 Data Fitting / Nonlinear Least Squares (10 Points)

In this exercise, we fit a parameterized model function to a noisy dataset.

Scenario: An experiment is run and measurements are taken at various sample points. The values include noise. We need to derive an analytical function that yields a good approximation of the experimental results over the whole domain. The experiment will be rerun regularly in the future, so our solution needs to be designed such that we can quickly rerun the analysis with an updated dataset. We expect the structure of the functional relationship to remain the same, though the exact behavior will change slightly.

Note: In this task your code will depend on the functional relationship represented by the dataset. However, do not make unnecessary assumptions. In particular, the location, order or number of data points may change.

Note: Your grade is determined by your success in solving the application problem and the quality of your implementation. It does not matter if you do not find the same function structure used to generate the sample dataset, but you should obtain a good fit with a reasonable model.

5.1 Load and Visualize the Dataset (2 Points)

In `main.m`:

- Load the example dataset from `taskDataset.mat`.
- Plot the point cloud (`x`, `y`, `z`) using either `scatter3()` or `plot3()`.

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

In `main.m`:

- Use the point cloud visualization to get a good impression of the dataset behavior.
- Identify a function structure with unknown parameters that can approximate `z` as function of `x`, `y`.
- Make an initial guess `p_guess` for your parameter vector.

In `modelFunction.m`:

- Implement the identified function structure with unknown parameters.
- Allow the caller to pass the parameter values as a vector argument.

Hint: A fairly simple model with approximately six parameters is sufficient.

5.3 Implement the Residual Function (2 Points)

In `main.m`:

- Implement a function handle `residual = @(p) …` calculating the residual vector (!) for all samples in the dataset.
- The input of the residual function is the parameter vector `p`. The output is the difference between the output of your model and the dataset, evaluated at each point in the dataset.

5.4 Identify the Optimal Parameters (2 Points)

In `main.m`:

- 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.
- You may need to refine the initial guess for the parameters to achieve convergence.

5.5 Visualize the Fitted Surface (2 Points)

In `main.m`:

- 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. If it is not, rethink the design of your model function and/or adjust the initial guess.