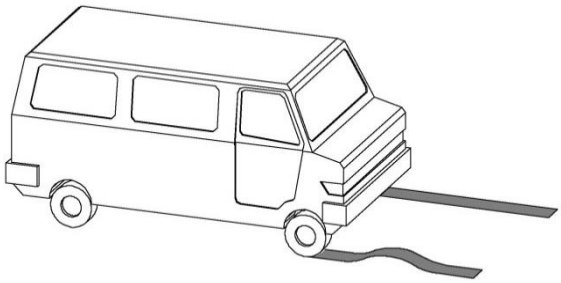
Motivational Example: Optimization of the Rear Suspension of a Vehicle

technical system



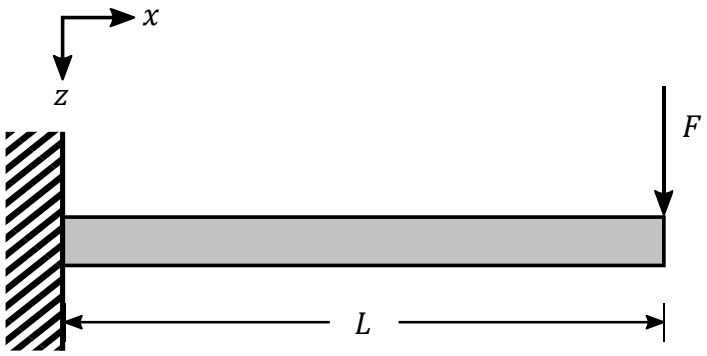
mechanical model

M : vehicle body mass
m : wheel and axle mass
c : stiffness of suspension
d : damping of suspension
c_W : stiffness of tire
s : street profile

mathematical model

$$\begin{aligned} M\ddot{x} &= -c(x - x_\mathrm{w}) - d(\dot{x} - \dot{x}_\mathrm{w}) - Mg \\ m\ddot{x}_\mathrm{w} &= \quad c(x - x_\mathrm{w}) + d(\dot{x} - \dot{x}_\mathrm{w}) - c_\mathrm{w}\big(x_\mathrm{w} - s(t)\big) - mg \end{aligned}$$

Motivational Example: Parameter Identification of a Flexible Beam



The Euler-Bernoulli equation describes the relationship between a beam's deflection and the applied load on the beam and is given by

$$EI\frac{\mathrm{d}^4 w(x)}{\mathrm{d}x} = q(x)$$

with the deflection in the z-direction $w(x)$ and the distributed load $q(x)$. The constant parameters of the equation are Young's modulus E and the area moment of inertia I. For a cantilever beam with a single force F acting on the free end, the solution of the Euler-Bernoulli equation is given by

$$w(x) = \beta F\left(Lx^2 - \frac{1}{3}x^3\right)$$

with $\beta = (2EI)^{-1}$.

In practice, the parameter β, incorporating E and I, might be unknown for the given beam. However, it is possible to identify the missing parameter through measurements, by applying a known force F at the end of the beam and measuring the bending $z = w(x)$. Multiple different measurements result in the data set $\mathcal{D} = \{(x_1, F_1, z_1), \dots, (x_N, F_N, z_N)\}$. The goal is to find a value for the unknown parameter $\beta > 0$ that minimizes the mean squared error (MSE) of the model prediction from the measurements, which can be formulated as a least-squares optimization problem of the form

$$\beta^\star = \underset{\beta}{\operatorname{argmin}} \_\_\_\_\_$$

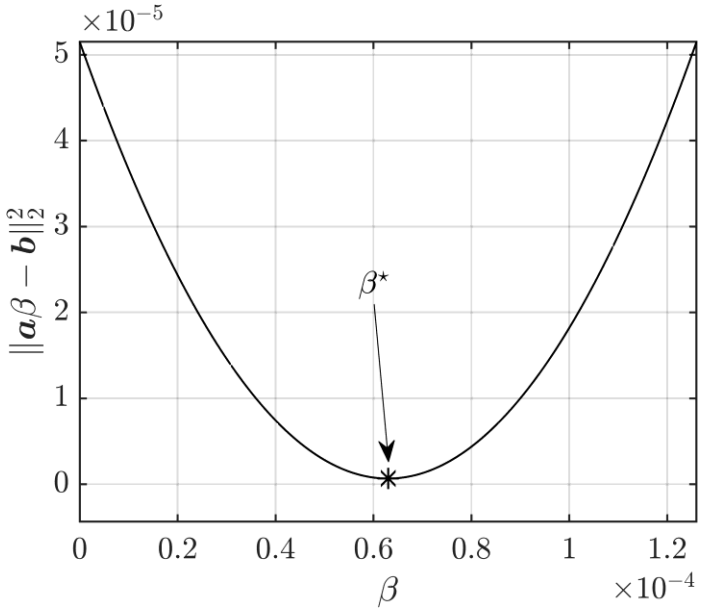
$$\text{s.t.} \quad \_\_\_\_\_$$

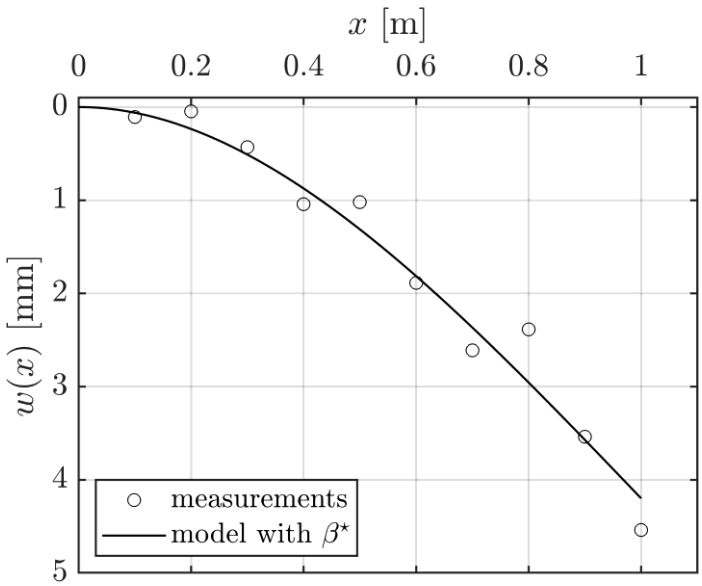
with $\boldsymbol{a} = \left[F_1\left(Lx_1^2 - \frac{1}{3}x_1^3\right) \quad \cdots \quad F_N\left(Lx_N^2 - \frac{1}{3}x_N^3\right)\right]^\top$ and $\boldsymbol{b} = [z_1 \quad \cdots \quad z_N]^\top$.

Data Set from Measurements, $L = 1\,\mathrm{m}$

#	force F $[N]$	position x $[m]$	bending $w(x)$ $[mm]$
1	100	0.1	0.1081
2	100	0.2	0.0465
3	100	0.3	0.4295
4	100	0.4	1.0426
5	100	0.5	1.0188
6	100	0.6	1.8866
7	100	0.7	2.6091
8	100	0.8	2.3853
9	100	0.9	3.5378
10	100	1.0	4.5390

Result of the Least-Squares Optimization





Chebyshev Approximation Problem

Instead of minimizing the mean squared error, it may be desirable to minimize the maximum deviation of the model prediction from the measurements, i.e., to solve the optimization problem

$$\beta^{\star,\mathrm{c}} = \operatorname*{argmin}_{\beta} \max_{k\in\{1,\dots,N\}} \_\_\_\_\_$$

$$\text{s.t.} \quad \_\_\_\_\_,$$

which is a non-smooth problem. However, using $\boldsymbol{p} \coloneqq [\beta \quad t]^\top$, it can be rewritten as a linear program (LP) in the form

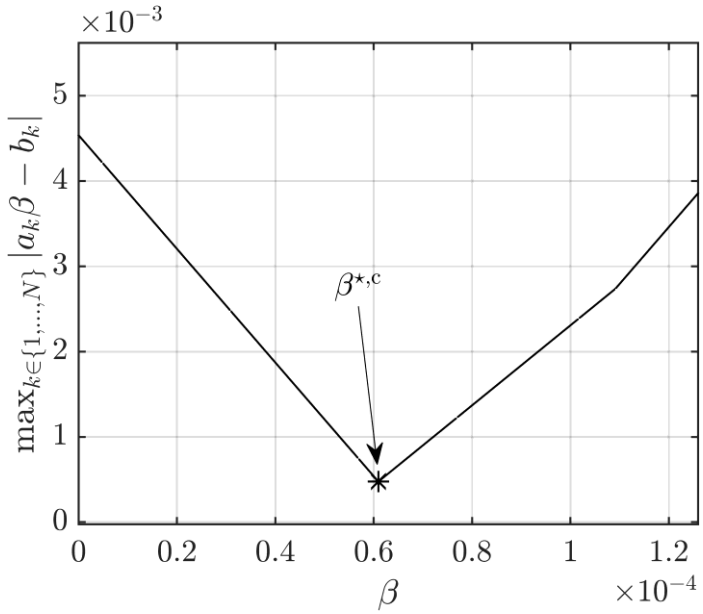
$$\min_{\boldsymbol{p}\in\mathbb{R}^2} \_\_\_\_\_$$

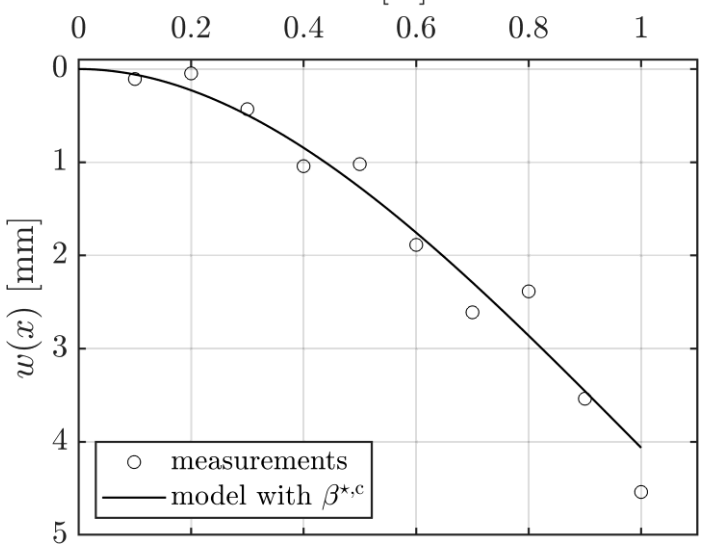
$$\text{s.t.} \quad \_\_\_\_\_,$$

$$\_\_\_\_\_,$$

$$\_\_\_\_\_.$$

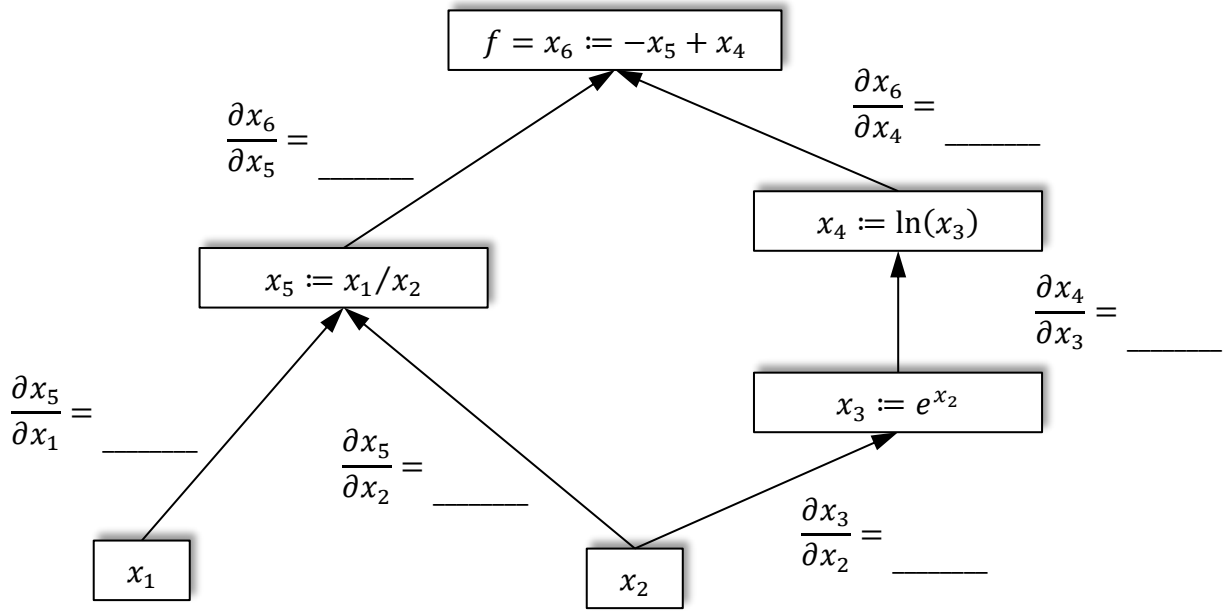
This LP can be solved efficiently with solvers such as the Simplex algorithm. For the data set from above, this yields the results shown below with the solution denoted as $\boldsymbol{p}^\star = [\beta^{\star,\mathrm{c}} \quad t^\star]^\top$.





Automatic Differentiation

Graph for $f(x_1, x_2) = -\frac{x_1}{x_2} + \ln(e^{x_2})$



Forward mode:

$$\nabla x_1 = \begin{bmatrix} \quad \end{bmatrix}, \qquad \nabla x_2 = \begin{bmatrix} \quad \end{bmatrix}$$

$$x_3 = \underline{\quad\quad\quad\quad}, \quad \nabla x_3 = \sum_{j \in J_3 = \{2\}} \frac{\partial x_3}{\partial x_j} \nabla x_j = \frac{\partial x_3}{\partial x_2} \nabla x_2 = \begin{bmatrix} \quad \end{bmatrix}$$

$$x_4 = \underline{\quad\quad\quad\quad}, \quad \nabla x_4 = \sum_{j \in J_4 = \{\ \}} \frac{\partial \_\_\_}{\partial \_\_\_} \nabla x_j = \underline{\quad\quad\quad\quad\quad\quad\quad\quad} = \begin{bmatrix} \quad \end{bmatrix}$$

$$x_5 = \underline{\quad\quad\quad\quad}, \quad \nabla x_5 = \sum_{j \in J_5 = \{\ \}} \frac{\partial \_\_\_}{\partial \_\_\_} \nabla x_j$$

$$= \underline{\quad\quad\quad\quad\quad\quad\quad\quad} + \underline{\quad\quad\quad\quad\quad\quad\quad\quad} = \begin{bmatrix} \quad \end{bmatrix}$$

$$x_6 = \underline{\quad\quad\quad\quad}, \quad \nabla x_6 = \sum_{j \in J_6 = \{\ \}} \frac{\partial \_\_\_}{\partial \_\_\_} \nabla x_j$$

$$= \underline{\quad\quad\quad\quad\quad\quad\quad\quad} + \underline{\quad\quad\quad\quad\quad\quad\quad\quad} = \begin{bmatrix} \quad \end{bmatrix}$$

Reverse mode:

$x_3 =$ _____ , $x_4 =$ _____ ,

$x_5 =$ _____ , $f = x_6 =$ _____

$\overline{x}_6 = \dfrac{\partial x_6}{\partial x_6} =$ _____ ,

$\overline{x}_5 = \sum_{j \in I_5 = \{6\}} \overline{x}_j \dfrac{\partial x_j}{\partial x_5} =$ _____ $=$ _____ ,

$\overline{x}_4 = \sum_{j \in I\_\_ = \{\ \}}$ _____ $=$ _____ $=$ _____ ,

$\overline{x}_3 = \sum_{j \in I\_\_ = \{\ \}}$ _____ $=$ _____ $=$ _____ ,

$\overline{x}_2 = \sum_{j \in I\_\_ = \{\ \}}$ _____ $=$ _____ $+$ _____

$=$ _____ ,

$\overline{x}_1 = \sum_{j \in I\_\_ = \{\ \}}$ _____ $=$ _____ $=$ _____ ,

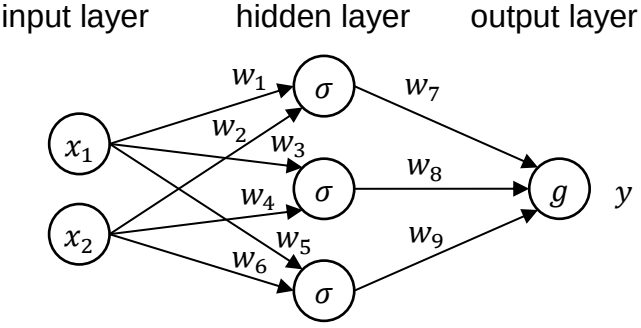
$\dfrac{\mathrm{d}f}{\mathrm{d}x} = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \overline{x}_1 \\ \overline{x}_2 \end{bmatrix} = \begin{bmatrix} \_\_\_\_\_ \\ \_\_\_\_\_ \end{bmatrix}.$

Artificial Neural Networks

Artificial neural networks (ANNs) are nowadays a popular machine learning technique. The goal is to identify or learn a function that fits a given data set, similar to A2. The structure and design of the network predefine a certain structure of the function that will be identified. In the end, the identification or learning task comes down to determining optimal values for a (usually large) set of network parameters (so-called weights) to optimally represent the data or, better yet, the process/relationship represented by the data. Classic linear regression like on A2 fits an ansatz consisting of a linear combination (i.e., weighted sum) of basis functions to the data. In contrast, (deep) artificial neural networks owe their expressiveness to the fact that nonlinear functions are composed/nested many times. To train the network, i.e., to optimize its parameters so that it fits a training data set, one usually uses gradient-based optimization algorithms. However, the network's structure, which is dominated by function compositions, makes it cumbersome to obtain gradient information by hand, which is why automatic differentiation is used for that. This is demonstrated subsequently for a very simple artificial neural network, with two input neurons, one output neuron, one hidden layer with three neurons and a classic feedforward structure. The sigmoid function

$$\sigma(z) \coloneqq \frac{1}{1+e^{-z}} \quad \Longrightarrow \frac{\mathrm{d}}{\mathrm{d}z}\sigma(z) = \sigma(z)\big(1-\sigma(z)\big)$$

is used as the activation function and, for simplicity, no bias neurons are considered.



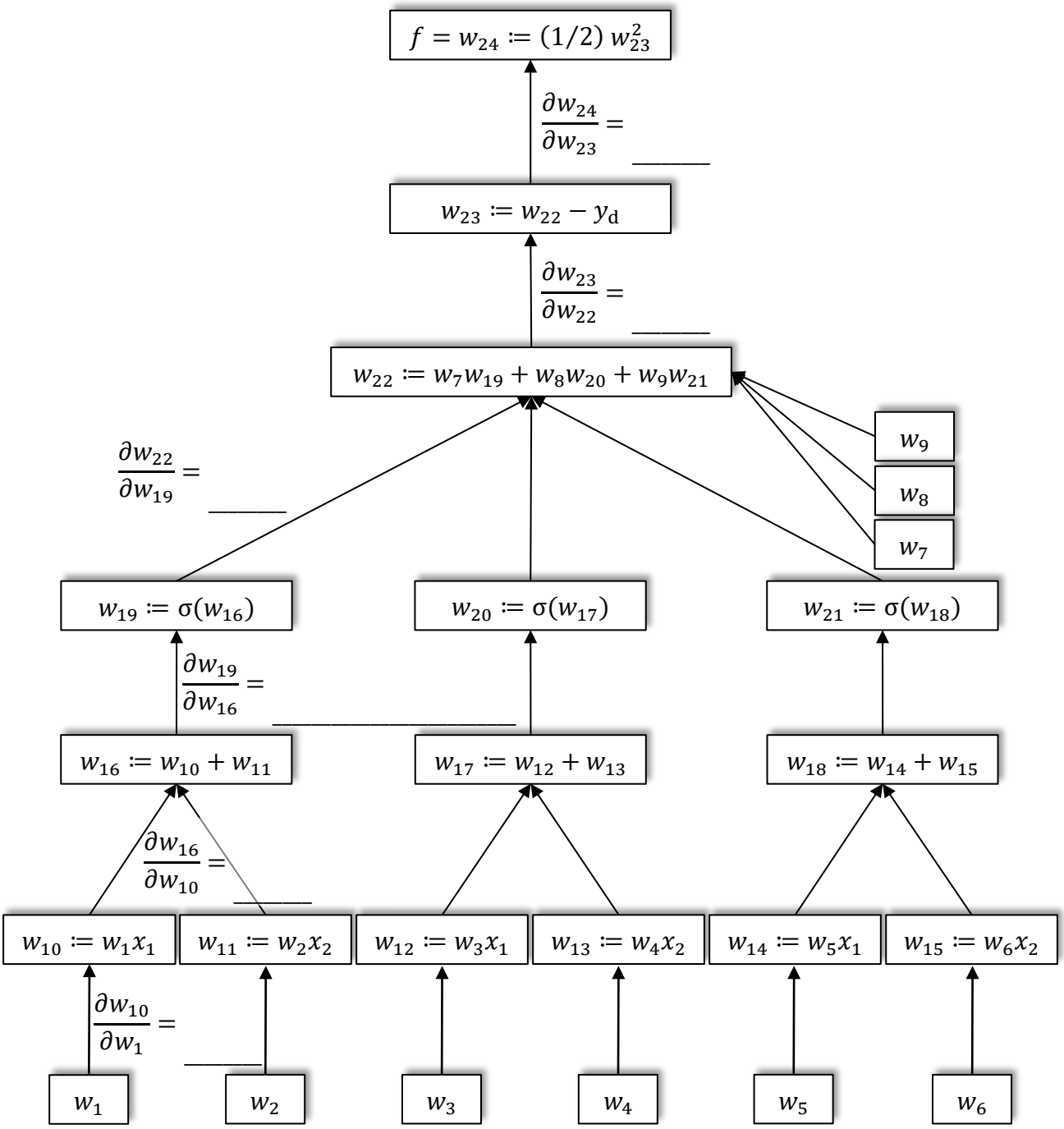
For simplicity, we choose $g(z) \coloneqq z$, which yields the output

$$y = w_7\,\sigma(w_1x_1 + w_2x_2) + w_8\,\sigma(w_3x_1 + w_4x_2) + w_9\,\sigma(w_5x_1 + w_6x_2).$$

For training purposes, i.e., to optimize the weights $\boldsymbol{w} \coloneqq [w_1 \quad \cdots \quad w_9]^\mathsf{T}$ so that the network fits a data set, a cost function (also called loss function in machine learning) is needed, which, here, is set to

$$f(\boldsymbol{w}) = \frac{1}{2}(y - y_\mathrm{d})^2.$$

Therein, y_d represents a desired output. Usually, for training in practical tasks, the cost function would consider all data points in the data set, e.g., the mean squared output error over all data points. Here, for shortness of notation, an individual data point is considered. The goal is now to obtain the gradient of the loss $\nabla f(\boldsymbol{w})$ so that gradient-based optimization can be used to optimize the weights. We focus on obtaining $\partial f/\partial w_1$; the rest of the gradient is obtained analogously.



Automatic Differentiation (Reverse Mode)

a) Give the index sets I_i, $i \in \{1, \ldots, 24\}$, for the input quantities and the introduced (intermediate) quantities as far as they are necessary to obtain $\partial f / \partial w_1$.

b) Compute the scalar gradients $\overline{w}_j = \partial f / \partial w_j$ for the output quantity and those intermediate quantities that are necessary to obtain $\partial f / \partial w_1$.

Hint: Use the relationship $\overline{w}_j = \sum_{i \in I_j} \overline{w}_i \frac{\partial w_i}{\partial w_j}$ for the intermediate quantities.

c) Give $\partial f / \partial w_1$ in terms of the input quantities $\boldsymbol{w} = [w_1 \quad \cdots \quad w_9]^\mathsf{T}$.

$$\frac{\partial f}{\partial w_1} =$$

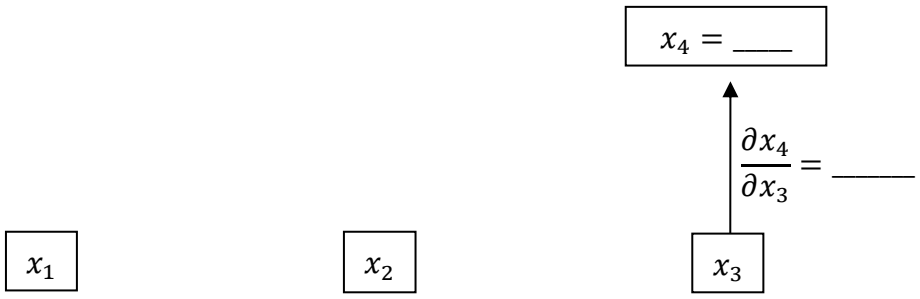
$$= \cdots$$

$$= x_1 (w_7\, \sigma(w_1 x_1 + w_2 x_2) + w_8\, \sigma(w_3 x_1 + w_4 x_2) + w_9\, \sigma(w_5 x_1 + w_6 x_2) - y_\mathrm{d})\, w_7\, \sigma(w_1 x_1 + w_2 x_2)\, (1 - \sigma(w_1 x_1 + w_2 x_2))$$

Automatic Differentiation

For optimization purposes the sensitivities of the function $f(\boldsymbol{x}) = x_1 \sin(x_2) - 5x_3$, with $\boldsymbol{x} = [x_1 \quad x_2 \quad x_3]^\top$, shall be calculated.

a) Complete the graph for the function $f(\boldsymbol{x})$ and write down the derivatives along the arcs.



Firstly, use the **forward mode** to compute the gradient of the function.

b) Give the index sets J_i for your introduced intermediate and output quantities.

c) Compute the gradients for all input and intermediate quantities.

d) Give the gradient of the function $f(\boldsymbol{x})$ in terms of the input quantities $\boldsymbol{x} = [x_1 \quad x_2 \quad x_3]^\top$.

Use also the **reverse mode** to compute the gradient of the function.

e) Give the index sets I_i for the input quantities and the introduced intermediate quantities.

f) Compute the scalar gradients $\overline{x}_j$ for the output and intermediate quantities.

g) Give the gradient of the function $f(\boldsymbol{x})$ in terms of the input quantities $\boldsymbol{x} = [x_1 \quad x_2 \quad x_3]^\top$.

h) Check your result by direct differentiation of the function $f(\boldsymbol{x})$.

Tool 1: CasADi for MATLAB

CasADi[1] is an open-source tool for nonlinear optimization and automatic differentiation. It is written in C++ but is most conveniently used via full-featured interfaces to MATLAB or Python. Applications range from academic teaching to fields such as optimal control, robotics, and aerospace. In order to use CasADi, an appropriate version can be downloaded[2] and added to the MATLAB search path.

```
1   import casadi.*                          % add CasADi toolbox
2   x       = SX.sym('x',2);                 % create 2-D variable x
3   f       = x(1)*x(2) - sin(x(2));         % define the function expression
4   gf      = jacobian(f,x);                 % calculate gradient expression
5   Hf      = jacobian(gf,x);                % calculate Hessian expression
6   gf_fun = Function('gf',{x},{gf});        % create functions for …
7   Hf_fun = Function('Hf',{x},{Hf});        % … evaluation of expressions
8   x0      = [1, 0];                        % define evaluation point
9   y1      = gf_fun(x0)                     % evaluate gradient
10  y2      = Hf_fun(x0)                     % evaluate Hessian

Output:     y1 = [[0, 0]]                    % value of gradient at x0
            y2 = [[0, 1],                    % value of Hessian at x0
                  [1, 0]]
```

Tool 2: autograd in Python

Autograd[3] is a Python package that can automatically differentiate native Python and Numpy code, the typical Python package for scientific computing. The main intended application of Autograd is gradient-based optimization. It can be installed via the package manager pip:

```
>> pip install autograd
```

```
1   import autograd.numpy as np              # import numpy
2   from autograd import jacobian            # import jacobian for AD
3   def f(x):                                # define the function
4       return x[0]*x[1] - np.sin(x[1])
5   gf_fun = jacobian(f)                     # calculate gradient expression
6   Hf_fun = jacobian(gf_fun)                # calculate Hessian expression
7   x0     = np.array([1., 0.])              # define evaluation point
8   print(gf_fun(x0))                        # evaluate gradient
9   print(Hf_fun(x0))                        # evaluate Hessian

Output:     [0.,0.]                          # value of gradient at x0
            [[0., 1.],                       # value of Hessian at x0
             [1., 0.]]
```

A very similar alternative developed from Autograd but also supporting, e.g., just-in-time compilation of gradients and functions is JAX[4].

[1] Andersson, A.E.A; Gillis, J.; Horn, G.; Rawlings, J.B. and Diehl, M.: CasADi – A software framework for nonlinear optimization and optimal control. Mathematical Programming Computation 11(1), pp. 1-36, 2019

[2] https://web.casadi.org/get/

[3] https://github.com/HIPS/autograd

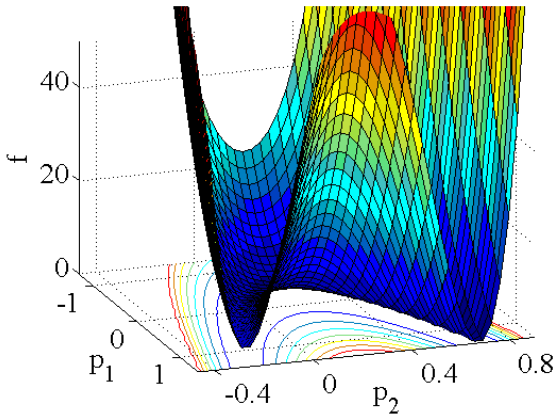
[4] https://github.com/google/jax

Minima of the Rosenbrock function

A common model problem for optimization algorithms is the minimization of the Rosenbrock function

$$f(\boldsymbol{p}) = 100(p_2 - p_1^2)^2 + (1 - p_1)^2.$$

For this function, many optimization algorithms show convergence problems. This is due to the slight curvature of the “valley” and the very slowly decreasing “bottom of the valley”.



a) Complete the gradient and the Hessian matrix for an arbitrary point $\boldsymbol{p}$.

$$\nabla f(\boldsymbol{p}) = \begin{bmatrix} \text{---------------------} \\ \text{---------------------} \end{bmatrix}$$

$$\nabla^2 f(\boldsymbol{p}) = \begin{bmatrix} \text{---------------} & \text{---------------} \\ \text{---------------} & \text{---------------} \end{bmatrix}$$

b) Give the necessary condition of first order for minima. Compute from this a potential minimizer $\boldsymbol{p}^\star$.

c) Does $\boldsymbol{p}^\star$ satisfy the necessary condition of second order?

Minimization of a Quadratic Function

Minimize the quadratic function

$$f(\boldsymbol{p}) = 10p_1^2 + p_2^2.$$

Analytical Solution

Using the gradient and the Hessian matrix,

$$\nabla f(\boldsymbol{p}) = \begin{bmatrix} \_\_\_\_\_\_\_ \\ \_\_\_\_\_\_\_ \end{bmatrix}, \qquad \nabla^2 f(\boldsymbol{p}) = \begin{bmatrix} \_\_\_\_ & \_\_\_\_ \\ \_\_\_\_ & \_\_\_\_ \end{bmatrix},$$

we get the solution from the necessary conditions for a local minimizer, i.e.

$$\_\_\_\_\_\_\_\_\_\_\_\_, \quad \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.$$

Here, the solution $\boldsymbol{p}^\star = \_\_\_\_$ is also a global minimizer of the function.

Newton – Method

Starting from the initial point $\boldsymbol{p}^{(0)} = [0.1 \quad 1]^\mathsf{T}$ the search direction is

$$\boldsymbol{s}^{(0)} = -\left(\nabla^2 f^{(0)}\right)^{-1} \nabla f^{(0)} = -\begin{bmatrix} \_\_\_\_ & \_\_\_\_ \\ \_\_\_\_ & \_\_\_\_ \end{bmatrix} \cdot \begin{bmatrix} \_\_\_ \\ \_\_\_ \end{bmatrix} = \begin{bmatrix} \_\_\_\_\_ \\ \_\_\_\_\_ \end{bmatrix}.$$

Along the corresponding line

$$\boldsymbol{p}(\alpha) = \boldsymbol{p}^{(0)} + \alpha \boldsymbol{s}^{(0)} = \begin{bmatrix} \_\_\_\_\_\_\_\_ \\ \_\_\_\_\_\_\_\_ \end{bmatrix},$$

the criterion function is

$$\bar{f}(\alpha) = f\big(\boldsymbol{p}(\alpha)\big) = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.$$

The minimum along this line follows from

$$\bar{f}' = \_\_\_\_\_\_\_\_\_\_ \overset{!}{=} 0 \quad \Rightarrow \quad \alpha^{(0)} = \_\_\_\_$$

which yields the improved solution

$$\boldsymbol{p}^{(1)} = \boldsymbol{p}\left(\alpha^{(0)}\right) = \_\ \_\ \_\ \_\ \_$$

and so we got the minimizer in the first step.

Conjugate Gradient Method

Starting from the initial point $\boldsymbol{p}^{(0)} = [0.1 \quad 1]^{\mathsf{T}}$ we get the following iteration steps:

ν	$\boldsymbol{p}^{(\nu)}$	$\nabla f^{(\nu)}$	exact line search
0	$\begin{bmatrix} - - - \\ - - - \end{bmatrix}$	$\begin{bmatrix} - - - \\ - - - \end{bmatrix}$	$\boldsymbol{s}^{(0)} = -\nabla f^{(0)} = \begin{bmatrix} - - - \\ - - - \end{bmatrix}$ $\boldsymbol{p}(\alpha) = \begin{bmatrix} - - - - - - - \\ - - - - - - - \end{bmatrix}$ $\overline{f}(\alpha) = \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_$ $\overline{f}' = \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_ \overset{!}{=} 0 \quad \Rightarrow \quad \alpha^{(0)} = \_\ \_\ \_$
1	$\begin{bmatrix} - - - \\ - - - \end{bmatrix}$	$\begin{bmatrix} - - - \\ - - - \end{bmatrix}$	$\boldsymbol{s}^{(1)} = -\nabla f^{(1)} + \frac{\left\|\nabla f^{(1)}\right\|^2}{\left\|\nabla f^{(0)}\right\|^2}\boldsymbol{s}^{(0)} = \begin{bmatrix} \\ \end{bmatrix}$ $\boldsymbol{p}(\alpha) = \begin{bmatrix} \\ \end{bmatrix}$ $\overline{f}(\alpha) = \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_$ $\overline{f}' = \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_ \overset{!}{=} 0 \quad \Rightarrow \quad \alpha^{(1)} = \_\ \_\ \_$
2	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	convergence in two steps	

Minimization of a Quadratic Function

Minimize the quadratic function

$$f(\boldsymbol{p}) = 10p_1^2 + p_2^2.$$

Quasi – Newton Method

Starting from the initial point $\boldsymbol{p}^{(0)} = [0.1 \quad 1]^\mathsf{T}$ we get the following iteration steps:

ν	$\boldsymbol{p}^{(\nu)}$	$\nabla f^{(\nu)}$	exact line search
0	$\begin{bmatrix} - - - \\ - - - \end{bmatrix}$	$\begin{bmatrix} - - - \\ - - - \end{bmatrix}$	$\boldsymbol{H}^{(0)} := \boldsymbol{I} = \begin{bmatrix} - - - & - - - \\ - - - & - - - \end{bmatrix}$ $\boldsymbol{s}^{(0)} = -\boldsymbol{H}^{(0)} \cdot \nabla f^{(0)} = \begin{bmatrix} - - - \\ - - - \end{bmatrix}$ $\boldsymbol{p}(\alpha) = \begin{bmatrix} - - - - - - - \\ - - - - - - - \end{bmatrix}$ $\overline{f}(\alpha) = \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_$ $\overline{f}' = \_ \_ \_ \_ \_ \_ \_ \_ \_ \stackrel{!}{=} 0 \implies \alpha^{(0)} = \_ \_ \_$
1	$\begin{bmatrix} - - - \\ - - - \end{bmatrix}$	$\begin{bmatrix} - - - \\ - - - \end{bmatrix}$	$\delta\boldsymbol{p} = \boldsymbol{p}^{(1)} - \boldsymbol{p}^{(0)} = \_ \_ \_ \begin{bmatrix} - - - \\ - - - \end{bmatrix}$ $\boldsymbol{\gamma} = \nabla f^{(1)} - \nabla f^{(0)} = \_ \_ \_ \begin{bmatrix} - - - \\ - - - \end{bmatrix}$ $\delta\boldsymbol{p} \cdot \boldsymbol{\gamma} = \_ \_, \quad \boldsymbol{H}^{(0)} \cdot \boldsymbol{\gamma} = \_ \_, \quad \boldsymbol{\gamma} \cdot \boldsymbol{H}^{(0)} \cdot \boldsymbol{\gamma} = \_ \_ \_ \_$

ν	$\boldsymbol{p}^{(\nu)}$	$\nabla f^{(\nu)}$	exact line search
			$\boldsymbol{H}^{(1)} = \boldsymbol{H}^{(0)} + \frac{\delta\boldsymbol{p}\delta\boldsymbol{p}^{\mathsf{T}}}{\_\_\_\_\_} - \frac{\_\_\_\_\_}{\_\_\_\_\_}$ $\boldsymbol{H}^{(1)} = \_\_\_\_\begin{bmatrix} \_\_\_\_ & \_\_\_\_ \\ \_\_\_\_ & \_\_\_\_ \end{bmatrix}$ $\boldsymbol{s}^{(1)} = -\boldsymbol{H}^{(1)} \cdot \nabla f^{(1)} = \_\_\_\begin{bmatrix} \_\_\_ \\ \_\_\_ \end{bmatrix}$ $\boldsymbol{p}(\alpha) = \begin{bmatrix} \_\_\_\_\_ \\ \_\_\_\_\_ \end{bmatrix}$ $\overline{f}(\alpha) = \_\_\_\_\_$ $\overline{f}' = \_\_\_\_\_ \overset{!}{=} 0 \quad \Rightarrow \quad \alpha^{(1)} = \_\_\_$
2	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	convergence in two steps	

Constrained Minimization of a Quadratic Function

Minimize the quadratic function

$$f(\boldsymbol{p}) = p_1^2 + p_2^2$$

subject to the constraint

$$-p_1 - p_2 \leq -4,$$

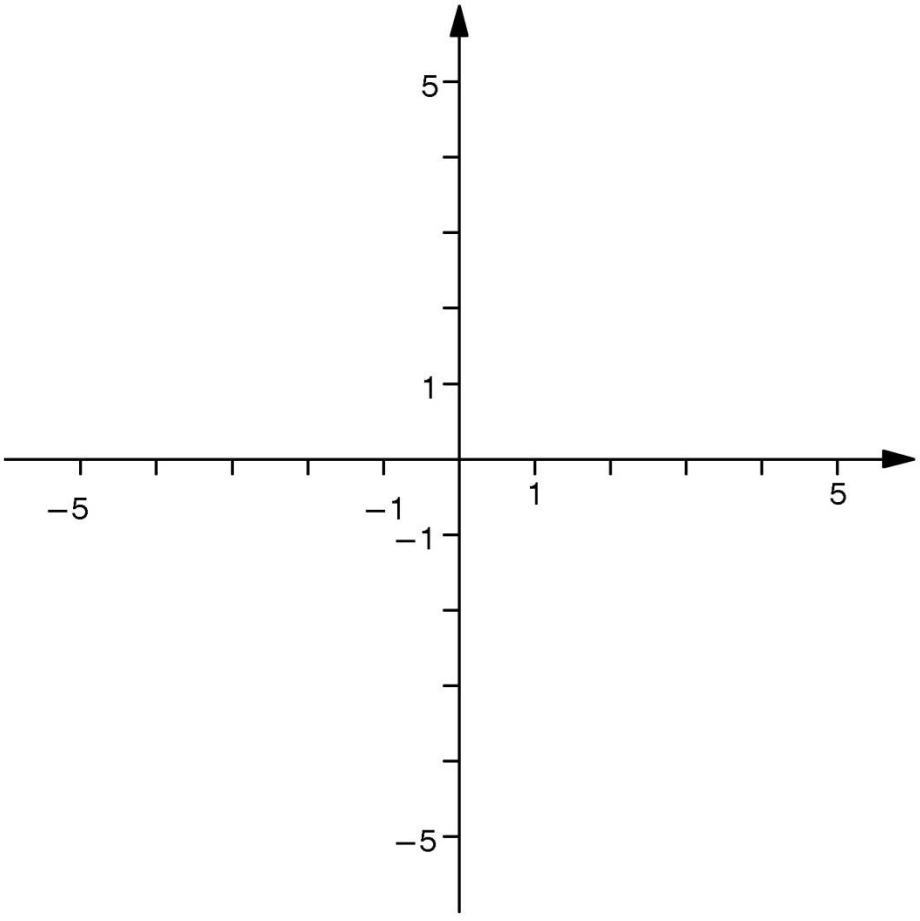
or, written in standard form

$$h_1(\boldsymbol{p}) = \_\_\_\_\_ \leq 0.$$

Graphical Solution

For this simple problem, the solution can be found graphically in the design parameter space. The dimension of the design parameter space is _____.

Plot the contour lines of the function f and the constraint h_1 and mark the optimal solution $\boldsymbol{p}^\star$.



Analytical Solution

The inequality constraint can be handled as an equality constraint by introducing the offset (slack) variable u_1

$$\overline{h}_1(p) = \_\_\_\_\_ = 0.$$

The Lagrangian function for an optimization problem with constraints generally reads as

$$L(\boldsymbol{p}, \mu, u) = f(\boldsymbol{p}) - \sum_{i=1}^{m} \mu_i (h_i + u_i^2).$$

It follows that

$$L(\boldsymbol{p}, \mu_1, u_1) = \_\_\_\_\_ .$$

The necessary condition of 1st order for a minimum is $\nabla L(\boldsymbol{p}, \mu_1, u_1) \stackrel{!}{=} 0$, which leads to the following four equations

$$\frac{\partial L}{\partial \boldsymbol{p}} = \begin{bmatrix} \_\_\_\_\_ \\ \_\_\_\_\_ \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} \_\_\_\_ \\ \_\_\_\_ \end{bmatrix},$$

$$\frac{\partial L}{\partial \mu_1} = \_\_\_\_\_ \stackrel{!}{=} \_\_\_\_ ,$$

$$\frac{\partial L}{\partial u_1} = \_\_\_\_\_ \stackrel{!}{=} \_\_\_\_ .$$

From the last equation we can distinguish two cases, either $\_\_\_\_\_$ or $\_\_\_\_\_$.

Case 1: _ _ _ _ _

If $\mu_1 = 0$, the constraint is not considered and therefore inactive. This is only true if the constraint is fulfilled. It follows from $\nabla L(\boldsymbol{p}, \mu_1, u_1) \overset{!}{=} 0$ that

$p_1 =$ _ _ _ _ _, $p_2 =$ _ _ _ _ _, and _ _ _ _ _ _ _ _ _ _ _ _ _ _ _.

This last equation cannot be satisfied and subsequently the constraint is not fulfilled at the minimum of f at $\boldsymbol{p} = \begin{bmatrix} \_ & \_ & \_ & \_ & \_ & \_ \end{bmatrix}^\top$ and case 1 is not true.

Case 2: _ _ _ _ _

If $u_1 = 0$, the distance from the limit of the constraint is 0 and at $\boldsymbol{p}^\star$ we have an active constraint.

The condition $\nabla L(\boldsymbol{p}, \mu_1, u_1) \overset{!}{=} 0$ leaves us with three equations

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _,

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _,

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _,

and it follows that

$p_1 =$ _ _ _ _ _, $p_2 =$ _ _ _ _ _, $\mu_1 =$ _ _ _ _ _.

Constrained Minimization of a Nonlinear Function

Minimize the nonlinear objective function

$$f(\boldsymbol{p}) = (p_1^3 + 10p_2)^2 - 100p_1$$

subject to the equality constraint

$$g(\boldsymbol{p}) = p_1^2 + p_2^2 - 9 = 0.$$

Solution by the Lagrange–Newton–Method

The Lagrangian function for this optimization problem can be written as

$$L(\boldsymbol{p}, \lambda) = -\lambda$$

The Karush-Kuhn-Tucker (KKT) conditions represent necessary conditions for an optimum. They can be written as

$$\frac{\partial L}{\partial \boldsymbol{p}} = \begin{bmatrix} \\ \end{bmatrix} \stackrel{!}{=} \begin{bmatrix} \\ \end{bmatrix},$$

$$\frac{\partial L}{\partial \lambda} = \stackrel{!}{=} .$$

These nonlinear equations can be solved by the Newton-Raphson method. This yields the update equation of the Lagrange-Newton method

$$\underbrace{\begin{bmatrix} \dfrac{\partial^2}{\partial} & \left(\dfrac{\partial g(\boldsymbol{p})}{\partial \boldsymbol{p}}\right)^{\top} \\ \dfrac{\partial g(\boldsymbol{p})}{\partial \boldsymbol{p}} & \end{bmatrix}_{\boldsymbol{p}^{(\nu)}, \lambda^{(\nu)}}}_{=\boldsymbol{A}(\boldsymbol{p},\lambda)} \begin{bmatrix} \delta \boldsymbol{p}^{(\nu+1)} \\ -\lambda^{(\nu+1)} \end{bmatrix} = \underbrace{\begin{bmatrix} -\dfrac{\partial f(\boldsymbol{p})}{\partial \boldsymbol{p}} \\ -g(\boldsymbol{p}) \end{bmatrix}_{\boldsymbol{p}^{(\nu)}}}_{=\boldsymbol{b}(\boldsymbol{p})}$$

With the objective and constraint function given above, the matrix $\boldsymbol{A}$ and the right hand side $\boldsymbol{b}$ follows as

$$\boldsymbol{A}(\boldsymbol{p},\lambda) = \begin{bmatrix} 30p_1^4 + & & \\ & 200 - 2\lambda & 2p_2 \\ & & 0 \end{bmatrix},$$

$$\boldsymbol{b}(\boldsymbol{p}) = \begin{bmatrix} \\ -20p_1^3 - \\ \\ \end{bmatrix}.$$

Starting with $\boldsymbol{p}^{(0)} = [\,1 \quad 0\,]^\top$ and $\lambda^{(0)} = 0$ the update equation reads as

$$\begin{bmatrix} 30 & & \\ 60 & 200 & \\ & & \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{p}^{(1)} \\ -\lambda^{(1)} \end{bmatrix} = \begin{bmatrix} 94 \\ \\ 8 \end{bmatrix}.$$

From these three scalar equations, the updated optimization variables and the new Lagrange multiplier estimate are given by

$$\boldsymbol{p}^{(1)} = \boldsymbol{p}^{(0)} + \delta\boldsymbol{p}^{(1)} = \begin{bmatrix} \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \end{bmatrix}^{\top}, \quad \lambda^{(1)} = \_ \; \_ \; \_ \; \_ \; \_$$

The next iteration yields

$$\begin{bmatrix} 18022 & & \\ \hline & & 1500 \\ \hline & -2.6 & \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{p}^{(2)} \\ \hline -\lambda^{(2)} \end{bmatrix} = \begin{bmatrix} -16700 \\ \hline -2240 \\ \hline -17.69 \end{bmatrix},$$

which results in

$$\boldsymbol{p}^{(2)} = [3.73482 \quad 0.63777] \text{ and } \lambda^{(2)} = -319.442.$$

The following table summarizes the first 10 iterations of the Lagrange-Newton method:

iteration	p_1	p_2	λ	f	g
0	1,0000E+00	0,0000E+00	0,0000E+00	-9,9000E+01	-8,0000E+00
1	5,0000E+00	-1,3000E+00	-2,6000E+01	1,2044E+04	1,7690E+01
2	3,7348E+00	6,3777E-01	-3,1944E+02	3,0458E+03	5,3556E+00
3	3,1496E+00	-1,3357E-01	2,5548E+01	5,7946E+02	9,3751E-01
4	2,8840E+00	-2,8855E+00	-1,1348E+02	-2,6471E+02	7,6436E+00
5	2,5959E+00	-1,8490E+00	-3,4899E+01	-2,5859E+02	1,1574E+00
6	2,5655E+00	-1,5787E+00	-1,0998E+01	-2,5534E+02	7,3970E-02
7	2,5667E+00	-1,5533E+00	-8,8899E+00	-2,5478E+02	6,4691E-04
8	2,5667E+00	-1,5531E+00	-8,8713E+00	-2,5477E+02	4,6864E-08
9	2,5667E+00	-1,5531E+00	-8,8713E+00	-2,5477E+02	3,5527E-15
10	2,5667E+00	-1,5531E+00	-8,8713E+00	-2,5477E+02	1,0011E-15

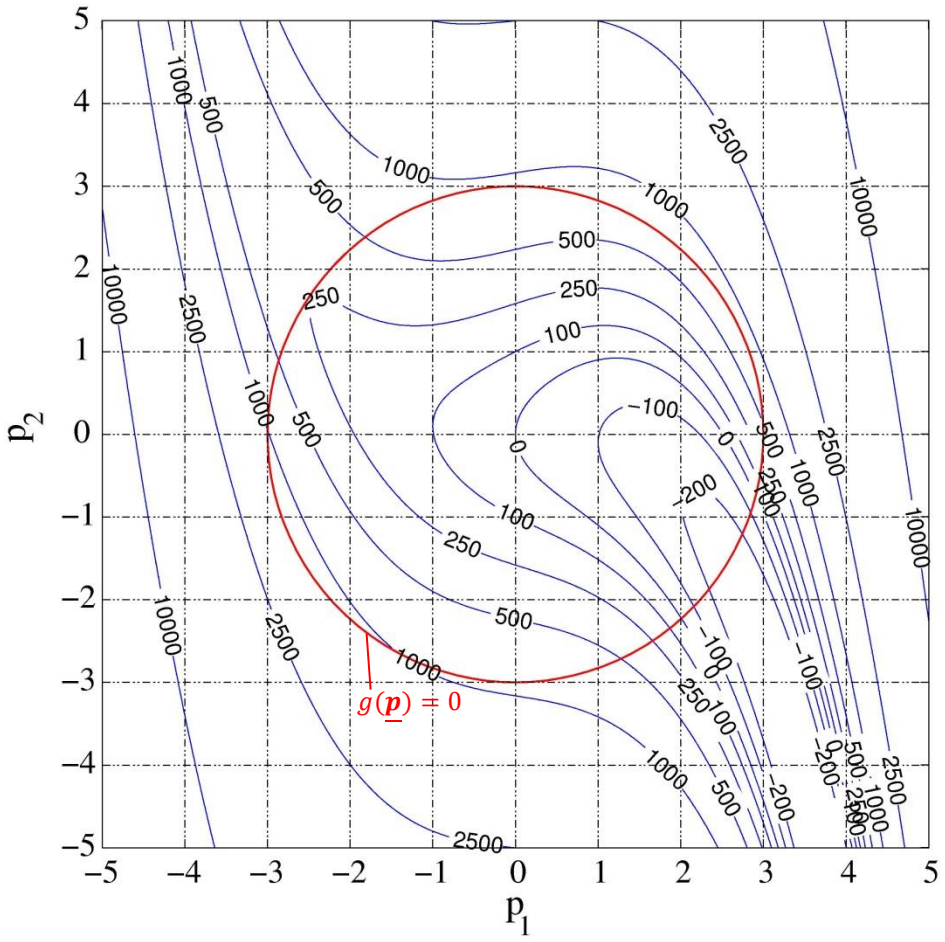
Draw the points $\boldsymbol{p}^{(0)}$ to $\boldsymbol{p}^{(4)}$ and the final point $\boldsymbol{p}^{(10)}$ in the contour plot on the next page.

Another solution process is started with the initial point $\boldsymbol{p}^{(0)} = [-1 \quad 0]^\top$ and $\lambda^{(0)} = 0$. The following table summarizes the corresponding optimization history:

iteration	p_1	p_2	λ	f	g
0	-1,0000E+00	0,0000E+00	0,0000E+00	1,0100E+02	-8,0000E+00
1	-5,0000E+00	1,3000E+00	7,4000E+01	1,3044E+04	1,7690E+01
2	-3,7693E+00	-7,7022E-01	-1,9290E+02	4,1289E+03	5,8005E+00
3	-3,2153E+00	2,8425E-01	8,7719E+01	1,2455E+03	1,4188E+00
4	-2,6628E+00	4,0384E+00	-3,0435E+02	7,2870E+02	1,4399E+01
5	-2,3117E+00	2,4872E+00	-8,3576E+01	3,8788E+02	2,5296E+00
6	-2,4700E+00	1,8314E+00	-8,2745E+00	2,5753E+02	4,5508E-01
7	-2,4825E+00	1,6904E+00	8,1362E+00	2,5082E+02	2,0039E-02
8	-2,4838E+00	1,6826E+00	8,9300E+00	2,5064E+02	6,3143E-05
9	-2,4838E+00	1,6825E+00	8,9323E+00	2,5064E+02	4,5733E-10
10	-2,4838E+00	1,6825E+00	8,9323E+00	2,5064E+02	-3,5527E-15

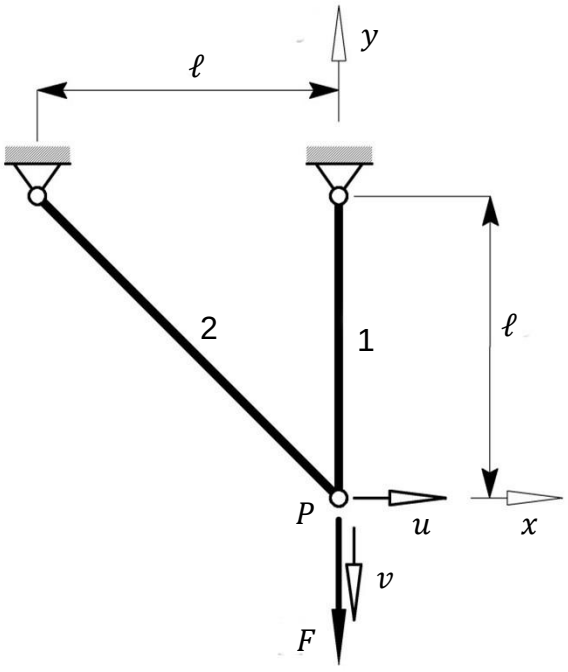
Draw the points $\boldsymbol{p}^{(0)}$ to $\boldsymbol{p}^{(4)}$ and the final point $\boldsymbol{p}^{(10)}$ of this run in the contour plot, too.

Contour plot of the constrained problem:



As we can see, the solution depends on the starting point $\boldsymbol{p}^{(0)}$.

Multicriteria Optimization of a Truss



For the presented truss the cross sections A_1 and A_2 of the two bars are to be designed. Simultaneously, the overall mass and the vertical displacement of the point P shall be minimized, while an external load F is applied at P.

This is a typical problem of multicriteria optimization since the overall mass corresponds to the costs of material and the vertical displacement is a measure for the function of the framework.

Using the density ρ the overall mass is

$$M = \_\_\_\_\_$$

For the computation of the vertical displacement, the mass of the truss can be neglected in comparison to the external load. Using finite element modelling we get for the displacement $\boldsymbol{y}$ of the joint

$$\boldsymbol{K}\boldsymbol{y} = \boldsymbol{q} \quad \text{where} \quad \boldsymbol{K} = \frac{E}{2\sqrt{2}\ell}\begin{bmatrix} A_2 & A_2 \\ A_2 & 2\sqrt{2}A_1 + A_2 \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} u \\ v \end{bmatrix}, \qquad \boldsymbol{q} = \begin{bmatrix} 0 \\ F \end{bmatrix}$$

which yields the vertical displacement

$$v = \_\_\_\_\_.$$

If A is a given minimal cross section of the bars, we get dimensionless design variables by normalization:

$$p_1 := \_\_\_\_\_, \quad p_2 := \_\_\_\_\_$$

and the dimensionless optimization criteria

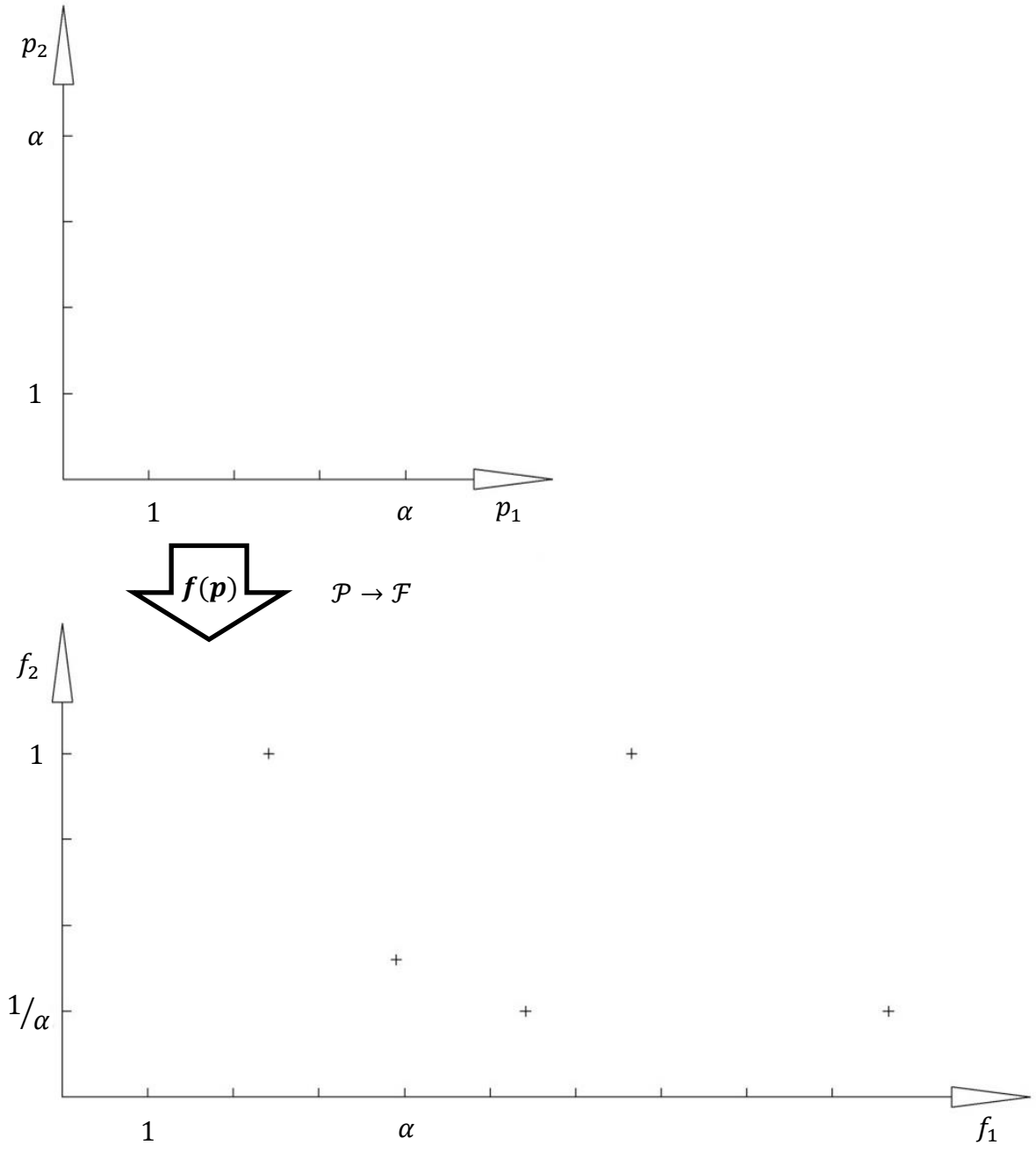
$$f_1 := \frac{M}{\rho\ell A} = \_\_\_\_\_, \quad f_2 := \frac{v}{F\ell/EA} = \_\_\_\_\_.$$

If the possible cross section has a maximum of αA with $\alpha > 1$, then we get the vector optimization problem

$$\underset{\boldsymbol{p}\in\mathcal{P}}{\text{opt}} \begin{bmatrix} \_\_\_\_\_ \\ \_\_\_\_\_ \end{bmatrix}$$

where $\mathcal{P} = \left\{ \boldsymbol{p} \in \mathbb{R}^2 \middle| \_\_\_\_\_ \right\}$.

Sketch the problem in the design space and in the criteria space for $\alpha = 4$.



From these sketches we get the Edgeworth–Pareto optimal solutions

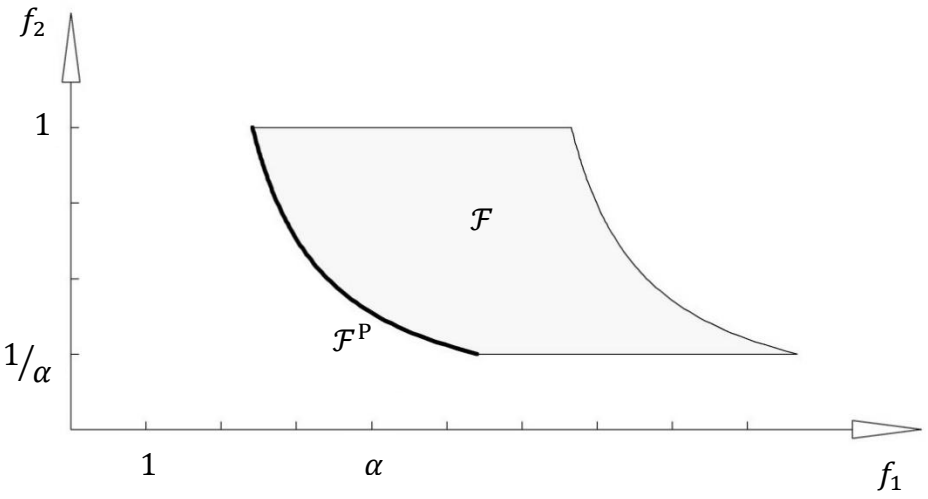
$$\mathcal{P}^{\mathrm{P}} = \left\{ \left[\quad \right], \; \_\_\_\_\_ \right\}$$

For the numerical determination of the EP–optimal solutions, we can formulate several scalar optimization problems. Formulate the following scalar optimization problems and determine the corresponding solution graphically.

1) Weighted Criteria Method

(Weighting factors: $w_1 = w_2 = 1/2$)

$$\min_{\boldsymbol{p} \in \mathcal{P}} \; \_\_\_\_\_$$

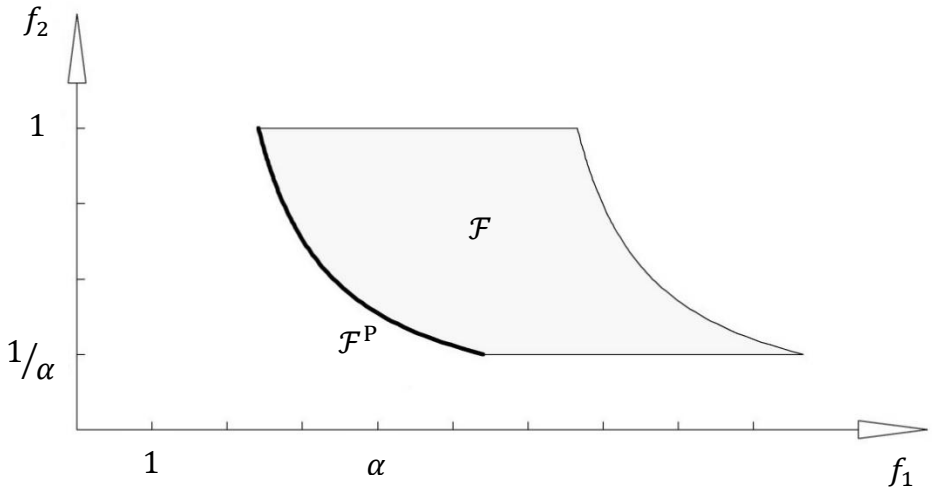


2) Distance Method

(Minimization of the Euklidean distance to the ideal solution

$$\boldsymbol{f}^0 = \left[\_\_\_\_\_ \right]^{\top})$$

$$\min_{\boldsymbol{p} \in \mathcal{P}} \; \_\_\_\_\_$$



3) **Hierarchical Method**

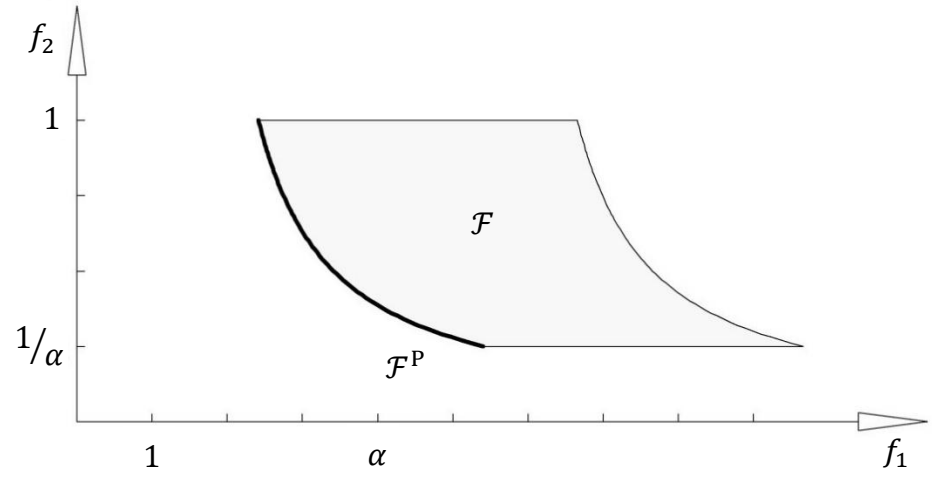
If the minimal vertical displacement is more important than the minimal overall mass of the truss, then the priorities are $l_1 =$ _ _ _ and $l_2 =$ _ _ _ .

In **step 1**, the minimization of the vertical displacement only leads to

$$\overline{f}_2 = \min_{\boldsymbol{p}\in\mathcal{P}^0} f_2(\boldsymbol{p}) = \_\ \_\ \_\ \_\ \_\ \_ \quad \text{where} \quad \mathcal{P}^0 = \_\ \_\ \_\ \_$$

In **step 2**, we try to additionally minimize the overall mass. For this, we may allow some increase of the vertical displacement, e.g., $\varepsilon_1 = 100\%$. The second optimization step then reads

$$\overline{f}_1 = \min_{\boldsymbol{p}\in\mathcal{P}^1} f_1(\boldsymbol{p}) \quad \text{where} \quad \mathcal{P}^1 = \{\_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\}$$



Application: Optimization of a Double Pendulum

An actively controlled double pendulum is described by the generalized coordinates

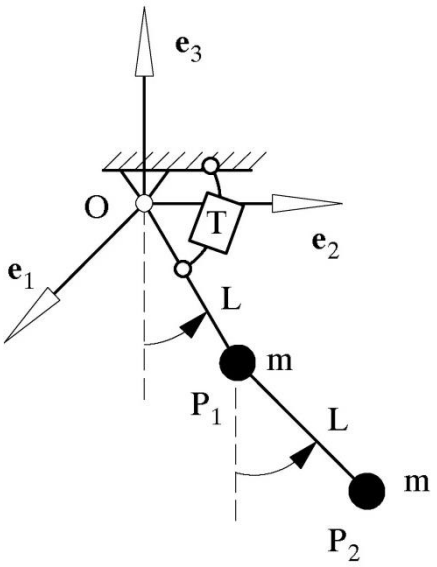
$$\boldsymbol{y} = [\alpha \quad \beta]^{\top}.$$

Without the torque actuator T, the dynamic behavior is described by the nonlinear equations of motion

$$\boldsymbol{M}(t, \boldsymbol{y})\ddot{\boldsymbol{y}} + \boldsymbol{k}(t, \boldsymbol{y}, \dot{\boldsymbol{y}}) = \boldsymbol{q}(t, \boldsymbol{y}, \dot{\boldsymbol{y}})$$

and the initial conditions are

$$\boldsymbol{y}(0) = \boldsymbol{y}_0, \quad \dot{\boldsymbol{y}}(0) = \boldsymbol{0}.$$



Linearization of the equations of motion about the equilibrium position results in

$$\boldsymbol{y}(t) = \boldsymbol{0} + \boldsymbol{\eta}(t), \quad \dot{\boldsymbol{y}}(t) = \boldsymbol{0} + \dot{\boldsymbol{\eta}}(t), \quad \ddot{\boldsymbol{y}}(t) = \boldsymbol{0} + \ddot{\boldsymbol{\eta}}(t)$$

and the following system of linear equations

$$\boldsymbol{M}(t)\ddot{\boldsymbol{\eta}}(t) + \boldsymbol{P}(t)\dot{\boldsymbol{\eta}}(t) + \boldsymbol{Q}(t)\boldsymbol{\eta}(t) = \boldsymbol{h}(t),$$

where

$$\boldsymbol{M} = \begin{bmatrix} 2mL^2 & mL^2 \\ mL^2 & mL^2 \end{bmatrix}, \quad \boldsymbol{P} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{Q} = \begin{bmatrix} 2mgL & 0 \\ 0 & mgL \end{bmatrix}.$$

The generalized force vector $\boldsymbol{q}$ has to be considered with the control torque $T(t)$ of the torque actuator

$$\boldsymbol{h}(t) = \begin{bmatrix} T(t) \\ 0 \end{bmatrix}$$

and the PD–control law is

$$T(t) = -k_1\alpha - k_2\dot{\alpha}$$

By an optimization process the control parameters k_1 and k_2 are to be optimized in such a way, that the vibrations of the pendulum, evaluated by the integral of $\|\boldsymbol{y}\|^2 = \boldsymbol{y}^\top \boldsymbol{y}$ within the first 10 s, are optimally damped. In order to ensure the stability of the system, both of the control parameters must not be negative.

First we formulate the problem in standard form:

design variables $\boldsymbol{p} = [\_\ \_\ \_\ ,\ \_\ \_\ \_]$

kinetics $\_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_ = \_\ \_\ \_\ \_\ \_$

initial conditions $\boldsymbol{\phi}^0 = \begin{bmatrix} \_\ \_\ \_\ \_\ \_\ \_\ \_ \\ \_\ \_\ \_\ \_\ \_\ \_\ \_ \end{bmatrix} = \mathbf{0}, \quad \dot{\boldsymbol{\phi}}^0 = \begin{bmatrix} \_\ \_\ \_\ \_\ \_\ \_\ \_ \\ \_\ \_\ \_\ \_\ \_\ \_\ \_ \end{bmatrix} = \mathbf{0}.$

final condition $\boldsymbol{H}^1 = \_\ \_\ \_\ \_\ \_\ \_\ \_ = 0$

criteria functions $\Psi_1 = \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_$

$\Psi_2 = \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_$

$\Psi_3 = \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_$

optimization criterion $f = \_\ \_\ \_\ \_\ \_\ \_\ \_$

inequality constraints $h_1 = \_\ \_\ \_\ \_\ \_\ \_\ \_$

$h_2 = \_\ \_\ \_\ \_\ \_\ \_\ \_$

The constrained optimization task then reads as

find $\_\ \_\ \_\ \_\ \_$, where $\_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_$

and $\_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_$

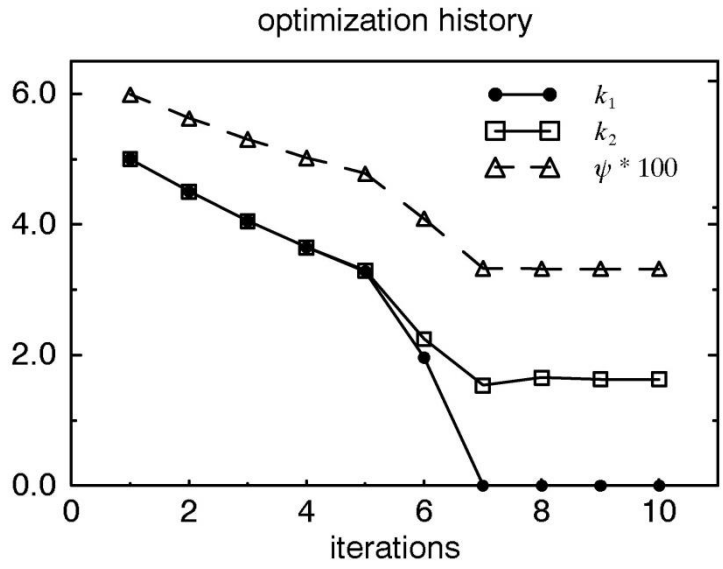
The described optimization task is solved by a SQP–method. The parameters of the double pendulum are

$m = 1\text{kg}$, $L = 0.3\text{m}$, $\boldsymbol{y}_0 = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$, and the initial design variable vector is $\boldsymbol{p}^{(0)} = \begin{bmatrix} 5.0 \\ 5.0 \end{bmatrix}$.

The optimization algorithm needs 10 gradient evaluations and 16 function evaluations in order to find the optimizer

$$\boldsymbol{p}^{\star} = \begin{bmatrix} 0.0 \\ 1.63 \end{bmatrix}.$$

The optimization history shows the convergence after 10 iterations:



The optimal dynamic behaviour shows a significantly improved damping in comparison to the passive and the initial design: