Optimization of Static and Dynamic Systems

Summary

Fabian Damken

November 8, 2023

TECHNISCHE UNIVERSITÄT DARMSTADT

Contents

List of Figures

List of Tables

List of Algorithms

1 Einführung

1.1 Beispiele

1.2 Fragestellungen

1.3 Allgemeine Formulierung eines Optimierungsproblems

1.4 Statische vs. Dynamische Optimierung

1.5 Klassifizierung von Optimierungsverfahren

1.6 Typische Struktur

2 Gradient-Based Optimization without Constraints

2.1 Solution Characterization

This section covers the theoretical results for solving a nonlinear optimization problem using calculus.

2.1.1 One-Dimensional Optimization

For a one-dimensional function $\varphi(p) : \mathbb{R} \to \mathbb{R}$ the first-order necessary condition for a minimum is that the derivative of $\varphi(p)$ w.r.t. the parameter p vanishes:

$$\frac{\mathrm{d}\varphi(p^*)}{\mathrm{d}p} = 0$$

Where p^* denotes the optimal solution, i.e. the minimum.

All solutions that fulfill this condition are *candidates* for a minimum. If φ is twice continuous differentiable, the sufficient condition for a minimum is that the second-order derivative is positive:

$$\frac{\mathrm{d}\varphi(p^*)}{\mathrm{d}p} > 0$$

Then p^* is called a *strict minimum*. This condition is sufficient, but not necessary! The second-order necessary condition for a minimum is that the second-order derivative is non-negative, i.e. $\frac{\mathrm{d}\varphi(p^*)}{\mathrm{d}p} \geq 0$.

Possibilities for a Minimum

There are three cases for a minimum:

- $\varphi(p)$ is twice continuously differentiable everywhere
- $\varphi'(p)$ is not continuous everywhere but at p^*
- $\varphi'(p)$ is not continuous everywhere, not even at p^*

While the latter case is common, it is problematic as the solution can typically not be determined analytically (if a function is not continuous at one point, it is rarely invertible).

2.1.2 Multi-Dimensional Optimization

For multi-dimensional objective functions $\varphi : \mathbb{R}^{n_p} \to \mathbb{R}$, where n_p is the dimensionality of the parameters, the first-order necessary condition is that the gradient vanishes:

$$\boldsymbol{\nabla}\varphi(\boldsymbol{p}^*) = \begin{bmatrix} \frac{\partial\varphi}{\partial p_1} \\ \vdots \\ \frac{\partial\varphi}{\partial p_{n_p}} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

If $\varphi(\boldsymbol{p})$ is twice continuously differentiable, the second-order sufficient condition is that the Hessian of $\varphi(\boldsymbol{p})$ is positive definite. Analogous to the one-dimensional case, the second-order necessary condition is that the Hessian is positive semi-definite, i.e.:

$$\boldsymbol{H}_\varphi(\boldsymbol{p}^*) = \begin{bmatrix} \frac{\partial^2 \varphi}{\partial p_1^2} & \cdots & \frac{\partial \varphi^2}{\partial p_{n_p} p_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \varphi^2}{\partial p_1 p_{n_p}} & \cdots & \frac{\partial^2 \varphi}{\partial p_{n_p}^2} \end{bmatrix} > 0 \quad \text{or respectively} \quad \boldsymbol{H}_\varphi(\boldsymbol{p}^*) \geq 0$$

Example

2.2 Numerical Gradient-Based Methods

2.2.1 Starting Point

Structure of Gradient-Based Methods

Given a initial approximation $\boldsymbol{p}^{(0)}$, an approximation of the minimum $\boldsymbol{p}^*$ is wanted. Gradient-based methods are iteration methods based on the iteration rule

$$\boldsymbol{p}^{(k+1)} = \boldsymbol{p}^{(k)} + \alpha^{(k)} \boldsymbol{d}^{(k)}, \quad k = 0, 1, 2, \cdots$$

where

- $\boldsymbol{d}^{(k)}$ is the search direction found as the solution of a linear sub problem and
- $\alpha^{(k)}$ is the step size found by a one-dimensional *line search*.

The iteration terminates once $\boldsymbol{p}^{(k+1)}$ is "close to" $\boldsymbol{p}^*$, e.g. when the gradient nearly vanishes.

Descent Direction

Gradient-based methods have to ensure the the local search direction $\boldsymbol{d}^{(k)}$ really is a descent direction (the algorithm shall not "run up the hill"). This property is ensured iff the angle δ between the search direction and the gradient $\boldsymbol{\nabla}\varphi\left(^{(k)}\right)$ greater than $90°$, i.e.

$$\cos\delta = \frac{\left(\boldsymbol{d}^{(k)}\right)^T \left(\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)\right)}{\left\|\boldsymbol{d}^{(k)}\right\| \cdot \left\|\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)\right\|} < 0 \quad \Longleftrightarrow \quad \left(\boldsymbol{d}^{(k)}\right)^T \left(\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)\right) < 0 \tag{2.1}$$

This is called the "necessary descent condition".

Algorithmic Structure

The algorithm 1 shows the basic structure of any gradient-based optimization algorithm.

Algorithm 1: Algorithmic structure of a gradient-based optimization algorithms.

Initialization: Choose an initial approximation $\boldsymbol{p}^{(0)}$, set $k \leftarrow 0$
while *not converged* **do**
  Determine new search direction: $\boldsymbol{d}^{(k)} \in \mathbb{R}^{n_p}$
  Determine new step size: $\alpha^{(k)} \in \mathbb{R}^{+}$
  Update the approximation: $\boldsymbol{p}^{(k+1)} \leftarrow \boldsymbol{p}^{(k)} + \alpha^{(k)} \boldsymbol{d}^{(k)}$
  $k \leftarrow k + 1$

2.2.2 Steepest Descent

Steepest descent is the straightforward way for getting a search direction. The search direction is just set to the negative of the gradient:

$$\boldsymbol{d}^{(k)} = -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)$$

- Advantages:
  - Often quickly reaches areas around the local minimum.
  - No second derivatives needed.
- Disadvantages:
  - Very slow in areas around the local minimum compared to (Quasi-) Newton Methods.

2.2.3 Conjugate Gradient

Basic approach for conjugate gradient:

$$\begin{aligned}\boldsymbol{d}^{(0)} &= -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(0)}\big)\\ \boldsymbol{d}^{(k)} &= \text{Component of } -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big) \text{ that is conjugate to } \boldsymbol{d}^{(0)}, \boldsymbol{d}^{(1)}, \cdots, \boldsymbol{d}^{(k-1)}\end{aligned}$$

For a quadratic objective function

$$\varphi(\boldsymbol{p}) = \frac{1}{2}\boldsymbol{p}^T\boldsymbol{A}\boldsymbol{p} - \boldsymbol{b}^T\boldsymbol{p}$$

with a positive semi-definite matrix $\boldsymbol{A}$ and constant $\boldsymbol{A}$, $\boldsymbol{b}$, the search direction is given as the solution of:

$$\big(\boldsymbol{d}^{(k)}\big)^T\boldsymbol{A}\boldsymbol{d}^{(j)} = 0, \quad j = 1, \cdots, k-1$$

With an optimal step size $\alpha^{(k)}$, i.e.

$$\alpha^{(k)} = \arg\min_{\alpha} \varphi\big(\boldsymbol{p}^{(k)} + \alpha\boldsymbol{d}^{(k)}\big) \quad \Longrightarrow \quad \alpha^{(k)} = -\frac{1}{\big(\boldsymbol{d}^{(k)}\big)^T\boldsymbol{A}\boldsymbol{d}^{(k)}}\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big)^T\big(\boldsymbol{d}^{(k)}\big)$$

the minimum of φ is reached in n_p steps.

The extension for nonlinear objective functions is given in algorithm 2.

- Exact line search necessary.

Algorithm 2: Conjugate Gradient for nonlinear Objective Function.

Initialization: Choose an initial approximation $\boldsymbol{p}^{(0)}$, set $\boldsymbol{d}^{(0)} \leftarrow -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(0)}\big)$ and $k \leftarrow 0$
while *not converged* **do**
  $\alpha^{(k)} \leftarrow \arg\min_{\alpha} \varphi\big(\boldsymbol{p}^{(k)} + \alpha \boldsymbol{d}^{(k)}\big)$
  $\boldsymbol{p}^{(k+1)} \leftarrow \boldsymbol{p}^{(k)} + \alpha^{(k)} \boldsymbol{d}^{(k)}$
  $\beta^{(k+1)} \leftarrow \dfrac{\big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k+1)}\big)\big)^T \big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k+1)}\big)\big)}{\big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\big)^T \big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\big)}$
  $\boldsymbol{d}^{(k+1)} \leftarrow -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k+1)}\big) + \beta^{(k+1)} \boldsymbol{d}^{(k)}$
  $k \leftarrow k + 1$

- Different variants of GC-algorithms mainly distinguish in the choice of of $\beta^{(k)}$.
- Advantages:
  - Faster then steepest descent.
  - No explicit storing of the Hessian $\boldsymbol{H}_{\varphi}\big(\boldsymbol{p}^{(k)}\big)$ necessary.
  - No explicit matrix-vector multiplication.
  - Useful even for extreme high dimensions n_p.
  - Exact for quadratic objectives $\varphi(\boldsymbol{p})$.
- Disadvantages:
  - A lot slower then (Quasi-) Newton Methods.
  - In general not useful for optimizing simulation models.

2.2.4 Newton Method

Assuming the approximation of each iteration, $\boldsymbol{p}^{(k)}$, is close to the minimum $\boldsymbol{p}^*$, the gradient $\boldsymbol{\nabla}\varphi(\boldsymbol{p}^*)$ can be taylor-expanded around $\boldsymbol{p}^{(k)}$:

$$\boldsymbol{\nabla}\varphi(\boldsymbol{p}^*) \overset{T\left(\boldsymbol{p}^{(k)}\right)}{=} \boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big) + \boldsymbol{H}_{\varphi}\big(\boldsymbol{p}^{(k)}\big)\big(\boldsymbol{p}^* - \boldsymbol{p}^{(k)}\big) + \cdots \overset{!}{=} \boldsymbol{0}$$

By leaving our the higher order terms the search direction $\boldsymbol{d}^{(k)} \coloneqq \boldsymbol{p}^* - \boldsymbol{p}^{(k)}$ is given by the solution of the system of linear equations

$$\boldsymbol{H}_{\varphi}\big(\boldsymbol{p}^{(k)}\big)\boldsymbol{d}^{(k)} = -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)$$

The realization is shown in algorithm 3.

When plugging the search direction into the necessary descent condition (2.1)

$$\big(\boldsymbol{d}^{(k)}\big)^T \Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big) = -\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)^T\Big)\Big(\boldsymbol{H}_{\varphi}\big(\boldsymbol{p}^{(k)}\big)\Big)^{-1}\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big) < 0$$

it is clear that this is only fulfilled iff the Hessian is positive definite. But this is only the case in a region around the minimum! If the approximation is far away from the minimum, the search direction might also be an ascent direction causing the Newton method to diverge. There are two main solutions to this problem:

Algorithm 3: Newton Method

Initialization: Choose an initial approximation $\boldsymbol{p}^{(0)}$, set $k \leftarrow 0$
while *not converged* **do**
  Solve $\boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k)}\big)\boldsymbol{d}^{(k)} = -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)$ for $\boldsymbol{d}^{(k)}$
  $\alpha^{(k)} \leftarrow \arg\min_\alpha \varphi\big(\boldsymbol{p}^{(k)} + \alpha\boldsymbol{d}^{(k)}\big)$
  $\boldsymbol{p}^{(k+1)} \leftarrow \boldsymbol{p}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}$
  $k \leftarrow k + 1$

1. If the Hessian is not positive definite, replace it by an identity matrix. That is, set the search direction to the steepest descent.
2. Regularize the equation system with a weight $\nu > 0$ such that the new matrix is positive definite (this "rotates" the matrix in the direction of the steepest descent such that the new search direction always fulfills the descent condition):

$$\Big(\boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k)}\big) + \nu\boldsymbol{I}\Big)\boldsymbol{d}^{(k)} = -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)$$

- Advantages:
  - Near to strong local minima of twice continuous differentiable objective, the Newton method is quadratic convergent.
- Disadvantages:
  - Computationally expensive as a linear system has to be solved in every iteration.
  - Not only first, but also second-order derivatives have to be available. This is a big disadvantage:
    * In practice, the first derivative is rarely and the second derivative is never available.
    * Even a single wrong component in the gradient or the Hessian destroys the quadratic convergence.

Availability of Second-Order Derivatives

The obvious idea is to approximate the Hessian using finite differences. The approximated Hessian is then given as

$$\boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k)}\big) = \frac{1}{2}\big(\tilde{\boldsymbol{H}} + \tilde{\boldsymbol{H}}^T\big)$$

where $\tilde{\boldsymbol{H}}$ is given by

$$\tilde{\boldsymbol{H}}_i = \frac{\partial}{\partial p_i}\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big) \approx \frac{1}{h_i}\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)} + h_i\boldsymbol{e}_i\big) - \boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big)$$

where $\tilde{\boldsymbol{H}}_i$ is the i-th column of $\tilde{\boldsymbol{H}}$. The equation (KKT.i) is used to force the Hessian to be symmetric.
Problems:

- The Hessian $\boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k)}\big) = \frac{1}{2}\big(\tilde{\boldsymbol{H}} + \tilde{\boldsymbol{H}}^T\big)$ is not necessarily positive definite.

- In every iteration the Gradient has to be evaluated n_p times more.
- The linear system still needs to be solved.
- Only useful for high-dimensional problems with sparse gradients!

Another possibility are *Quasi-Newton Methods*.

2.2.5 Quasi-Newton Methods

Quasi-Newton methods are equivalent to the Newton method, however, the Hessian (or its inverse) is approximated by a positive definite matrix

$$\hat{\boldsymbol{H}}^{(k)} \approx \boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k)}\big)$$

that is updated in every iteration. This yields a lot of advantages over the classic Newton method:

- Only first-order derivatives needed.
- As $\hat{\boldsymbol{H}}$ constructed positive definite, the descent condition is fulfilled anytime.
- If the inverse Hessian is directly approximated, only $\mathcal{O}(n_p^2)$ multiplications instead of $\mathcal{O}(n_p^3)$ for solving the linear system.

But how to do the Hessian update? By Taylor-expanding the gradient $\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)$ around $\boldsymbol{p}^{(k+1)}$

$$\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)^{T\left(\boldsymbol{p}^{(k+1)}\right)}_{=} \boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k+1)}\big) + \boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k+1)}\big)\big(\boldsymbol{p}^{(k)} - \boldsymbol{p}^{(k+1)}\big) + \cdots \overset{!}{=} \boldsymbol{0}$$

and cutting off the higher-order terms, the following approximation holds:

$$\boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k+1)}\big)\boldsymbol{d}^{(k)} \approx \boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k+1)}\big) - \boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)$$

The approximation of the Hessian must therefore fulfill the *secant condition*

$$\tilde{\boldsymbol{H}}^{(k+1)}\boldsymbol{d}^{(k)} = \boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k+1)}\big) - \boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)$$

There exist a lot of different approaches for doing the Hessian updates $\tilde{\boldsymbol{H}}^{(k+1)} = \tilde{\boldsymbol{H}}^{(k)} + \boldsymbol{U}(k)$ for rank-1 or rank-2 matrices $\boldsymbol{U}^{(k)}$:

- Approach for rank-1 corrections: $\tilde{\boldsymbol{H}}^{(k+1)} = \tilde{\boldsymbol{H}}^{(k)} + \beta_1 \boldsymbol{u}\boldsymbol{u}^T$
- Approach for rank-2 corrections: $\tilde{\boldsymbol{H}}^{(k+1)} = \tilde{\boldsymbol{H}}^{(k)} + \beta_1 \boldsymbol{u}\boldsymbol{u}^T + \beta_2 \boldsymbol{v}\boldsymbol{v}^T$

The vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{n_p}$ and scalars $\beta_1, \beta_2 \in \mathbb{R}$ must have to be chosen such that $\tilde{\boldsymbol{H}}^{(k+1)}$ is

- positive definite,
- symmetric,
- fulfills the secant condition and
- adding up the matrices is efficient and robust.

BFGS-Update

The most known rank-2 update for the Hessian is the *BFGS-Update*[1]

$$\boldsymbol{u} = \tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)} \qquad \beta_1 = -\frac{1}{\left(\boldsymbol{d}^{(k)}\right)^T \tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)}}$$

$$\boldsymbol{v} = \boldsymbol{g}^{(k)} \qquad \beta_2 = \frac{1}{\left(\boldsymbol{g}^{(k)}\right)^T \boldsymbol{d}^{(k)}}$$

where $\boldsymbol{g}^{(k)} = \boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k+1)}\right) - \boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)$. Plugging that into the general approach for rank-2 updates yields the update rule for BFGS-approximations:

$$\tilde{\boldsymbol{H}}^{(k+1)} = \tilde{\boldsymbol{H}}^{(k)} - \frac{1}{\left(\boldsymbol{d}^{(k)}\right)^T \tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)}} \tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)}\left(\tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)}\right)^T + \frac{1}{\left(\boldsymbol{g}^{(k)}\right)^T \boldsymbol{d}^{(k)}} \boldsymbol{g}^{(k)}\left(\boldsymbol{g}^{(k)}\right)^T$$

- The direct approximation of the Hessian inverse is not really robust (e.g. for a non-optimal step size rule).
- A better alternative is to directly approximate a useful factorization, e.g. the Cholesky decomposition. This is more robust and equally efficient ($\mathcal{O}(n_p^2)$).

The pseudo code for the BFGS update is shown in algorithm 4.

Algorithm 4: Quasi-Newton Method with BFGS Update.

Initialization: Choose an initial approximation $\boldsymbol{p}^{(0)}$, set $\tilde{\boldsymbol{H}}^{(0)} = \boldsymbol{I}$ and $k \leftarrow 0$
while *not converged* **do**
  Solve $\tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)} = -\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)$ for $\boldsymbol{d}^{(k)}$
  $\alpha^{(k)} \leftarrow \arg\min_\alpha \varphi\left(\boldsymbol{p}^{(k)} + \alpha\boldsymbol{d}^{(k)}\right)$
  $\boldsymbol{p}^{(k+1)} \leftarrow \boldsymbol{p}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}$
  $\boldsymbol{g}^{(k)} \leftarrow \boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k+1)}\right) - \boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)$
  $\tilde{\boldsymbol{H}}^{(k+1)} \leftarrow \tilde{\boldsymbol{H}}^{(k)} - \frac{1}{(\boldsymbol{d}^{(k)})^T \tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)}} \tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)}\left(\tilde{\boldsymbol{H}}^{(k)}\boldsymbol{d}^{(k)}\right)^T + \frac{1}{(\boldsymbol{g}^{(k)})^T \boldsymbol{d}^{(k)}} \boldsymbol{g}^{(k)}\left(\boldsymbol{g}^{(k)}\right)^T$
  $k \leftarrow k + 1$

2.2.6 Comparison

2.2.7 Notes and Discussion

- The convergence of gradient-based methods can be shown under weak preconditions.
- As the search direction is only a local descent direction, gradient-based algorithms only yields local minima.
- There is no algorithm that can guarantee to find the global minimum!
- Some approaches for determining a global minimum:

[1] "BFGS" stands for the authors Broyden, Fletcher, Goldfarb and Shanno.

  - Choose the initialization well, i.e. close to the global minimum.
  - Execute the algorithm multiple times with different starting points.
  - Validate the solution against properties of the original problem.
  - Execute direct search methods beforehand to find promising regions for the local minimum search.

- Advantages:
  - If gradient-based algorithms converge, they converge utterly fast.
  - Efficient also for high-dimensional problems, i.e. a large n_p.

- Disadvantages:
  - Only applicable for functions that are differentiable almost everywhere.
  - Require gradient information exact up to four to eight decimal points.
  - Convergence to a local minimum near the initialization $\boldsymbol{p}^{(0)}$.
  - Require some expert knowledge.

2.3 Step Size Rules, Line Search

In every iteration of gradient-based algorithms, the step size has to be determined by minimizing the one-dimensional function:

$$\psi(\alpha) = \varphi\big(\boldsymbol{p}^{(k)} + \alpha \boldsymbol{d}^{(k)}\big)$$

As the necessary first-order condition for a minimum, the derivative w.r.t. α has to vanish:

$$\frac{\mathrm{d}\psi\big(\alpha^{(k)}\big)}{\mathrm{d}\alpha} = \frac{\mathrm{d}}{\mathrm{d}\alpha}\varphi\big(\boldsymbol{p}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}\big) = \Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}\big)\Big)^T \boldsymbol{d}^{(k)} \stackrel{!}{=} 0$$

Thus the gradient of φ at the minimum $\alpha^{(k)}$ as to be orthogonal to the search direction $\boldsymbol{d}^{(k)}$. Intuitively, the optimal step size has to be chosen such that the iteration step cannot go any further without ascending again ("hitting an ascending contour line").

Goal of the line search is to reach the minimum of ψ with least invocations of ψ as possible. Most of the existing search methods can be classified into

- *Polynomial approximation*, e.g. quadratic or cubic interpolation
- *Direct search methods*, e.g. Fibonacci-search, golden ratio search
- *Optimal vs. non-optimal search methods*, e.g. by finding an improvement but not the minimum
- Usage of the gradient information ψ' or not.

Requirements:

- Finding the $\alpha^{(k)}$ with a minimal value of ψ.
- Do not waste too much computation time on the line search.

In general, an exact line search requires lots of ψ-evaluations. But how far does ψ need to be reduced in order to guarantee convergence? In general, the condition $\varphi\big(\boldsymbol{p}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}\big) < \varphi\big(\boldsymbol{p}^{(k)}\big)$ is not enough!

2.3.1 Inexact Line Search

Procedure: Generate and inspect a series of candidates for $\alpha^{(k)}$ and terminate once one of the candidates fulfills specific criteria, e.g. the Armijo rule or Wolfe conditions.

Armijo Rule

The *Armijo rule* guarantees a sufficient reduction in φ:

$$\varphi\big(\boldsymbol{p}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}\big) \leq \varphi\big(\boldsymbol{p}^{(k)}\big) + c_1\alpha^{(k)}\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big)^T\boldsymbol{d}^{(k)} = \varphi\big(\boldsymbol{p}^{(k)}\big) + c_1\alpha^{(k)}\psi'(0)$$

Where $0 < c_1 < 1$ is any constant, e.g. $c_1 = 10^{-4}$.

Hence, the minimal reduction has to be proportional to $\alpha^{(k)}$ and the derivative $\psi'(0)$.

Curvature Condition

But a sufficient descent condition is not enough as the step sizes must not be too small (otherwise progress would stop). Thus a second condition has to be employed, the *curvature condition* that requires a minimum curvature on ψ:

$$\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}\big)\Big)^T\boldsymbol{d}^{(k)} \geq c_2 \cdot \Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big)^T\boldsymbol{d}^{(k)} = c_2\psi'(0) \quad \Longleftrightarrow \quad \psi'\big(\boldsymbol{p}^{(k)}\big) \geq c_2\psi'(0)$$

Where $c_1 < c_2 < 1$ is any constant, e.g. $c_2 = 0.9$.

Wolfe and Goldstein Conditions

Combining the Armijo rule and the curvature condition yields the Wolfe conditions that guarantee both a minimal reduction and a minimal curvature. They are especially useful for Quasi-Newton methods as the Wolfe conditions are scale invariant, i.e. independent of

- multiplying φ with any constant and
- affine transformations of $\boldsymbol{p}$.

There are other possibilities are, e.g. the *Goldstein conditions*

$$\varphi\big(\boldsymbol{p}^{(k)}\big) + (1-c)\alpha^{(k)}\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big)^T\boldsymbol{d}^{(k)} \leq \varphi\big(\boldsymbol{p}^{(k)} + \alpha^{(k)}\boldsymbol{d}^{(k)}\big) \leq \varphi\big(\boldsymbol{p}^{(k)}\big) + c\alpha^{(k)}\Big(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\Big)^T\boldsymbol{d}^{(k)}$$

with any $0 < c < 1/2$, that are useful for Newton, but not for Quasi-Newton methods.

2.3.2 Notes

- For gradient-based methods a step size $\alpha^{(k)} > 1$ is in general not useful because:
  - The search direction $\boldsymbol{d}^{(k)}$ is determined using a linear or quadratic Taylor approximation.
  - The Taylor approximation is only valid in a small region around the current approximation $\boldsymbol{p}^{(k)}$, i.e. for $0 < \alpha^{(k)} \leq 1$.
- The local quadratic or super-linear convergence of Newton-type methods is visible in practice as the last step can be executed with full step size $\alpha^{(k)} = 1$.

2.4 Trust Region Methods

Gradient-based methods with line search determine a fixed search direction and adjust the step size $\alpha^{(k)}$ according to that search direction to reach global convergence.

Another approach is to determine both the length and direction of $\boldsymbol{d}^{(k)}$. The iteration step then becomes

$$\boldsymbol{p}^{(k+1)} = \boldsymbol{p}^{(k)} + \boldsymbol{d}^{(k)}$$

without any explicit step size. The length and direction of $\boldsymbol{d}^{(k)}$ are then determined as a solution of the quadratic sub-problem

$$\begin{aligned} \min_{\boldsymbol{d}\in\mathbb{R}^{n_p}} & \left(\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)\right)^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k)}\big)\boldsymbol{d} \\ \text{subject to} \quad & \|\boldsymbol{d}\|_2 \leq \delta \end{aligned}$$

where δ describes the area around the current approximation $\boldsymbol{p}^{(k)}$ where the quadratic approximation of $\varphi\big(\boldsymbol{p}^{(k)} + \boldsymbol{d}\big)$ makes sense, i.e. the *trust region*.

The value of δ is extremely important for the efficiency of the method.

- If δ is too small, opportunities for large steps are missed.
- If δ is too large, the minimum of the quadratic approximation might be far off the minimum of the objective if the Hessian is indefinite or negative definite.

It is possible to add a regularization parameter $\beta \geq 0$ to the quadratic approximation

$$\Big(\boldsymbol{H}_\varphi\big(\boldsymbol{p}^{(k)}\big) + \beta\boldsymbol{I}\Big)\boldsymbol{d} = -\boldsymbol{\nabla}\varphi\big(\boldsymbol{p}^{(k)}\big)$$

such that the matrix is positive semidefinite. The solution of this regularized problem also solves the trust region problem if either $\beta = 0$, $\|\boldsymbol{d}\| \leq \delta$ or $\beta \geq 0$, $\|\boldsymbol{d}\| = \delta$.

2.5 Rate of Convergence

A common criteria to measure the performance of a gradient method are *rates of convergence*. These provide information on how fast an algorithm converges, i.e. how fast $\boldsymbol{p}^{(k)} \to \boldsymbol{p}^*$ or $\big\|\boldsymbol{p}^{(k)} - \boldsymbol{p}^*\big\| \to 0$.

Definition: Let $\big(\{\, \boldsymbol{p}^{(k)} \,\}$ be the series of approximations produced by an optimization method. Then this series has rate of convergence r if r is the greatest positive number such that the limit

$$0 \leq \lim_{k\to\infty} \frac{\big\|\boldsymbol{p}^{(k+1)} - \boldsymbol{p}^*\big\|}{\big\|\boldsymbol{p}^{(k)} - \boldsymbol{p}^*\big\|^r} = \gamma < \infty$$

converges (where $\boldsymbol{p}^* = \lim_{k\to\infty} \boldsymbol{p}^{(k)}$). If $r = 1$, then $\gamma < 1$ has to hold for the method to converge.

A sequence is said to converge superlinearly if

$$\lim_{k\to\infty} \frac{\big\|\boldsymbol{p}^{(k+1)} - \boldsymbol{p}^*\big\|}{\big\|\boldsymbol{p}^{(k)} - \boldsymbol{p}^*\big\|} = 0$$

holds. Even though technically this condition holds for $r > 1$, in practice only methods with $1 < r < 2$ are said to converge superlinearly (e.g. for $r = 2$, the sequence is called to converge quadratically).

2.5.1 Gradient-Based Methods

Under ideal conditions (i.e. φ is twice continuously differentiable and $\boldsymbol{H}_\varphi(\boldsymbol{p}^*)$ is positive definite), all of the following hold:

- Steepest descent is (locally) linearly convergent (with exact line search).
- The Newton method is (locally) quadratically convergent.
- Quasi-Newton methods with BFGS-update are (locally) superlinearly convergent (for inexact line search using the Wolfe conditions).

But: Even a single wrong component in the Hessian reduced the quadratic convergence of the Newton method to linear convergence!

3 Gradient-Free Optimization without Constraints

This chapter covers different types of sampling methods (direct search methods):

1. Metaheuristics (random search),
2. Deterministic Sampling Methods (pattern search) and
3. Surrogate optimization.

These optimization methods only use evaluations of the objective φ and do not user gradient information (neither analytically nor using numerical differentiation). That is, φ is used as a "Black Box" for function evaluations.

Even for φ that are not differentiable and have lots of local minima, gradient-free algorithms are remarkably robust, but can also fail fast.

3.1 Introduction

The objective φ that is to be optimized often has suboptimal properties, e.g.:

- the evaluation is noisy
- not differentiable
- high computation time for evaluation (e.g. when a simulation has to be run for each evaluation)

But gradient-free techniques have a wide range of applications, e.g. in automobile, aerospace industry, robotics, financial, etc. The goal is to reduce the objective function as much as possible and find regions in which the objective is differently sensitive w.r.t. changes in the optimization variables.

3.1.1 Simulation-Based Optimization

In a simulation-based setting, the objective is evaluated by running a simulation (e.g. by solving a set of differential equations or running a real experiment). This yields a suboptimal setting for optimization, yet a common one:

- Only function values are computable and not gradient information is available.
- Source code of the optimization is often not available. Hence, no information about how the simulation is computed.
- Discontinuous/non-differentiable systems and model properties, e.g. due to collisions.
- Non-differentiable structure inside the simulation (e.g. if-else).
- Discontinuities caused by subprograms, heuristics, table data, ...

- Function evaluations are computationally expensive.
- Numerical “noise” overlay the actual system properties.
- Non-deterministic simulations (e.g. due to complex friction).

Hence, simulation-based optimization is a black box problem and can be solved (or approximated) using gradient-free methods!

3.1.2 Black-Box Optimization

Naturally, gradient-based optimization methods are not well-suited for problems with “low” differentiability and computationally expensive function evaluations. These problems may be solved using gradient-free optimization methods. But…

- Gradient-based techniques are really slow in comparison to gradient-based ones for differentiable optimization problems.
- They need a lot of function evaluations for high-dimensional problems and are thus practically only applicable for problems with dimensions $n_p < 100$, better $n_p < 20$.
- Have lots of problems with nonlinear equality constraints!
- The theory of gradient-free methods is not as mature as the theory for gradient-based methods.

3.2 Metaheuristics

3.2.1 Evolutionary Algorithm (EA)

A *evolutionary algorithm* mimics the biological evolutionary strategy with random search methods.

1. Initialization: Choose one “Parent” $\boldsymbol{p}^{(0)}$ and a number of descendants ℓ.
2. Iteration: Create ℓ descendants via “mutation”

$$\boldsymbol{p}^{(k,i)} = \boldsymbol{p}^{(k)} + \alpha_i^{(k)} \boldsymbol{d}_i, \quad i = 1, \cdots, \ell$$

where $\boldsymbol{d}_i \in \mathbb{R}^{n_p}$ are vectors of Gaussian distributed variables and $\alpha_i \in \mathbb{R}$ art suitable “mutation step sizes” Then select the descent with the lower φ value and repeat.

3.2.2 Genetic Algorithms (GA)

Genetic algorithms are inspired by biological evolutionary strategies the positive properties caused by mutation are kept through natural selection. GAs are applicable for both real and discrete optimization variables $\boldsymbol{p}$.

1. Initialization: Choose a suitable set of different “individuals” (first generation).
2. Evaluation: Determine the “fitness” of each candidate using the objective/fitness function.
3. Selection: Randomly select candidates of the current generation (the higher the fitness, the higher the probability to be chosen).

4. Recombination: Combine values (genomes) of the selected individuals and create new individuals.
5. Mutation: Randomly change the genomes.
6. New Generation: Select new individuals as the new generation and continue with step 2.

Example

3.2.3 Further Metaheuristics

Further metaheuristics based on real representations of $\boldsymbol{p}$ like in evolutionary algorithms:

- Particle Swam: Population method, uses idea of combining local and swarm knowledge, direction and speed for particles are adjusted
- ...

Further metaheuristics based on binary representations of $\boldsymbol{p}$ like in genetic algorithms:

- Tabu Search: A list of possible manipulations is given, another lists dynamically the inverses of them, these cannot be applied any more
- ...

3.3 Deterministic Sampling Methods (Pattern Search Methods)

Deterministic sampling methods can be further categorized into

- Qualitative methods: Only ranking w.r.t. to the function value.
  - Simplex Methods
  - Coordinate- or compass-search
  - Multidirectional search
  - Pattern search methods
  - ...
- Quantitative methods: Consideration of the real function values.
  - Implicit filtering (based on the simplex method)
  - DIRECT (dividing rectangles)
  - ...

3.3.1 Nelder-Mead Simplex Method

A *simplex* is a simple object that consists of $n_p + 1$ points $\boldsymbol{p}^{(i)}$ in the parameter space (which is n_p-dimensional). In a 2D space, the three points form a triangle. In the iteration phase the values of φ at the corners are compared and the simplex is transformed according to specific rules (see subsubsection 3.3.1). The algorithm terminates once the simplex contracts onto a single point.

Iteration Phase

The iteration phase starts by sorting the edges according to its φ-values:

$$\varphi\big(\boldsymbol{p}^{(1)}\big) \leq \varphi\big(\boldsymbol{p}^{(\mathbf{2})}\big) \leq \cdots \leq \varphi\big(\boldsymbol{p}^{(n_p+1)}\big)$$

where $\boldsymbol{p}^{(1)}$ is called the *best* point and $\boldsymbol{p}^{(n_p+1)}$ is called the *worst*. The algorithm now tried to replace the worst point $\boldsymbol{p}^{(n_p+1)}$ with another point of the form

$$\boldsymbol{p}(\mu) = (1+\mu)\bar{\boldsymbol{p}} - \mu\boldsymbol{p}^{(n_p+1)}$$

Where $\bar{\boldsymbol{p}}$ is the centroid of the of all points *except the worst*, i.e.:

$$\bar{\boldsymbol{p}} = \frac{1}{n_p}\sum_{i=1}^{n_p}\boldsymbol{p}^{(i)}$$

This corresponds to a reflection of the worst point over the centroid with a weight μ that specifies "how far the point point gets pushed out", i.e. the ratio of the original distance of the worst point to the centroid that is preserved while reflecting. If $\mu = 1$, the point is mirrored.

In every iteration, the value μ is chosen of a set of four values

$$-1 < \mu_{ic} < 0 < \mu_{oc} < \mu_r < \mu_e$$

for example $(\mu_{ic}, \mu_{oc}, \mu_r, \mu_e) = (-0.5, 0.5, 1, 2)$.

Algorithm

Some termination criteria are for example:

- Exactness in the objective: $\varphi\big(\boldsymbol{p}^{(n_p+1)}\big) - \varphi\big(\boldsymbol{p}^{(1)}\big) \leq \varepsilon$
- Maximum number of function evaluations: $k = k_{\max}$
- Sufficient small distance on the simplex corners.
- Initialization: Choose a start simplex, evaluate the objective at the corners, sort them and set $k = n_p + 1$.
- Iteration: While $\varphi\big(\boldsymbol{p}^{(n_p+1)}\big) - \varphi\big(\boldsymbol{p}^{(1)}\big) > \varepsilon$ and $k < k_{\max}$, do:
  (a) Calculate the centroid $\bar{\boldsymbol{p}} = \frac{1}{n_p}\sum_{i=1}^{n_p}\boldsymbol{p}^{(i)}$.
  (b) *Reflection:* If $\varphi\big(\boldsymbol{p}^{(1)}\big) \leq \varphi\big(\boldsymbol{p}(\mu_r)\big) < \varphi\big(\boldsymbol{p}^{(n_p)}\big)$, replace $\boldsymbol{p}^{(n_p+1)}$ with $\boldsymbol{p}(\mu_r)$; go to (g).
  "Use the reflected point if it is better than the second worst, but not better than the best."
  (c) *Expansion:* If $\varphi\big(\boldsymbol{p}(\mu_r)\big) < \varphi\big(\boldsymbol{p}^{(1)}\big)$, then:
  "If the reflected point is better than the best, …"
  – If $\varphi\big(\boldsymbol{p}(\mu_e)\big) < \varphi\big(\boldsymbol{p}(\mu_r)\big)$, replace $\boldsymbol{p}^{(n_p+1)}$ with $\boldsymbol{p}(\mu_e)$; go to (g).
  "…and the expanded point is better than the reflected point, use the expanded point."
  – If $\varphi\big(\boldsymbol{p}(\mu_r)\big) < \varphi\big(\boldsymbol{p}(\mu_e)\big)$, replace $\boldsymbol{p}^{(n_p+1)}$ with $\boldsymbol{p}(\mu_r)$; go to (g).
  "…and the expanded point is worst than the reflected point, use the reflected point."
  (d) *Outer Contraction:* If $\varphi\big(\boldsymbol{p}^{(n_p)}\big) \leq \varphi\big(\boldsymbol{p}(\mu_r)\big) < \varphi\big(\boldsymbol{p}^{(n_p+1)}\big)$, then:
  "If the reflected point is better than the worst, but worst than second worst, …"

- If $\varphi\big(\boldsymbol{p}(\mu_{oc})\big) < \varphi\big(\boldsymbol{p}(\mu_r)\big)$, replace $\boldsymbol{p}^{(n_p+1)}$ with $\boldsymbol{p}(\mu_{oc})$; go to (g).
  "…and the outer contraction point is better than the reflected point, use the outer contraction point."
- Else, go to (f).

(e) *Inner Contraction:* If $\varphi\big(\boldsymbol{p}^{(n_p+1)}\big) \leq \varphi\big(\boldsymbol{p}(\mu_r)\big)$, then:
"If the reflected point is worst than the worst point, …"
- If $\varphi\big(\boldsymbol{p}(\mu_{ic})\big) < \varphi\big(\boldsymbol{p}^{(n_p+1)}\big)$, replace $\boldsymbol{p}^{(n_p+1)}$ with $\boldsymbol{p}(\mu_{ic})$; go to (g).
  "…and the inner contraction point is better than the worst, use the inner contraction point."
- Else, go to (f).

(f) *Shrink:* For all $2 \leq i \leq n_p + 1$, set $\boldsymbol{p}^{(i)} = \boldsymbol{p}^{(1)} - \frac{1}{2}\big(\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(1)}\big)$.

(g) *Sort:* Sorting the current simplex corners such that $\varphi\big(\boldsymbol{p}^{(1)}\big) \leq \varphi\big(\boldsymbol{p}^{(\mathbf{2})}\big) \leq \cdots \leq \varphi\big(\boldsymbol{p}^{(n_p+1)}\big)$ holds again. Repeat.

The Nelder-Mead method therefore always tries to create a big simplex and only shrink if every other action would yield a worst corner/simplex.

Notes

- The method is not guaranteed to converge. But in practice, it yields good results.
- Can get stuck on a suboptimal point such that the algorithm has to be restarted with other initial simplex corners.

Simplex-Gradient It is possible to detect stagnation using a *Simplex-Gradient* $\boldsymbol{D}^{(k)} \in \mathbb{R}^{n_p}$, $\boldsymbol{D}^{(k)} = \big(\boldsymbol{V}^{(k)}\big)^{-T}\boldsymbol{\delta}^{(k)}$ where $\boldsymbol{V}^{(k)}$ is the matrix of the simplex directions

$$\boldsymbol{V}^{(k)} = \begin{bmatrix}\boldsymbol{p}^{(2)} - \boldsymbol{p}^{(1)} & \boldsymbol{p}^{(3)} - \boldsymbol{p}^{(1)} & \cdots & \boldsymbol{p}^{(n_p+1)} - \boldsymbol{p}^{(1)}\end{bmatrix} =: \begin{bmatrix}\boldsymbol{v}^{(1)} & \boldsymbol{v}^{(2)} & \cdots & \boldsymbol{v}^{(n_p)}\end{bmatrix} \in \mathbb{R}^{n_p \times n_p}$$

and $\boldsymbol{\delta}^{(k)}$ is the vector of the objective differences:

$$\boldsymbol{\delta}^{(k)} = \begin{bmatrix}\varphi\big(\boldsymbol{p}^{(2)}\big) - \varphi\big(\boldsymbol{p}^{(1)}\big) \\ \varphi\big(\boldsymbol{p}^{(3)}\big) - \varphi\big(\boldsymbol{p}^{(1)}\big) \\ \vdots \\ \varphi\big(\boldsymbol{p}^{(n_p+1)}\big) - \varphi\big(\boldsymbol{p}^{(1)}\big)\end{bmatrix} \in \mathbb{R}^{n_p}$$

Analogous to a gradient-based method, this yields a condition for *minimum progress*

$$\hat{\varphi}^{(k+1)} - \hat{\varphi}^{(k)} < -\alpha\big\|\boldsymbol{D}^{(k)}\big\|^2, \qquad \hat{\varphi} = \frac{1}{n_p+1}\sum_{i=1}^{n_p+1}\varphi\big(\boldsymbol{p}^{(i)}\big)$$

with a small $\alpha > 0$. One approach for a condition on when to restart is to restart if both

$$\hat{\varphi}^{(k+1)} - \hat{\varphi}^{(k)} > -\alpha\big\|\boldsymbol{D}^{(k)}\big\|^2 \quad \text{and} \quad \hat{\varphi}^{(k+1)} - \hat{\varphi}^{(k)} < 0$$

hold.

3.3.2 Multidirectional Search Methods

- In the Nelder-Mead method a bad conditioning of the simplex, i.e. the matrix $\boldsymbol{V}^{(k)}$, leads to problems that cannot be avoided.
- In multidirectional search methods this problem is avoided by making every simplex congruent to its predecessors.
- The algorithm uses similar steps for reflection, expansion and contraction, but possibly needs a lot more function evaluations.

3.3.3 Asynchronous Parallel Pattern Search (APPS)

- *Asynchronous Parallel Pattern Search* is a pattern-based search method on a grid.
- The direction of the pattern determines the descent direction.
- Patterns can be varied while maintaining their mathematical properties.
- It is "naturally" parallelizable.

3.3.4 Implicit Filtering

Implicit filtering is a descent method using "smooth" approximations of the gradients. It uses a central approximation of the simplex gradient

$$\boldsymbol{D}_C^{(k)} = \frac{1}{2}\left(\boldsymbol{D}^{(k)} + \boldsymbol{D}_R^{(k)}\right)$$

where $\boldsymbol{D}_R^{(k)}$ is the gradient of the simplex that is reflected around $\boldsymbol{p}^{(k)}$.

The structure of implicit filtering is sketched inalgorithm 5.

Algorithm 5: Implicit Filtering.

Initialization: Choose $\alpha, \beta \in (0,1)$ and set $\boldsymbol{H} = \boldsymbol{I}$
while *not converged* **do**
  Calculate $\varphi\left(\boldsymbol{p}^{(k)}\right)$, $\boldsymbol{D}_C^{(k)}$ and the search direction $\boldsymbol{d}^{(k)} = -\boldsymbol{H}^{-1}\boldsymbol{D}_C^{(k)}$
  Inexact line search for $j = 1, \cdots, j_{\max}$, $\lambda := \beta^j$ until the following holds:

$$\varphi(\boldsymbol{p} + \lambda\boldsymbol{d}) - \varphi(\boldsymbol{p}) \leq \alpha\lambda\boldsymbol{\nabla}_{\Delta\boldsymbol{p}_k}\varphi^T(\boldsymbol{p})\boldsymbol{d}^{(k)}$$

  $\boldsymbol{p}^{(k+1)} \leftarrow \boldsymbol{p}^{(k)} + \lambda\boldsymbol{d}^{(k)}$
  if *line search successful* **then**
    Quasi-Newton update of the Hessian $\boldsymbol{H}$ with $\boldsymbol{p}^{(k+1)} - \boldsymbol{p}^{(k)}$ and $\boldsymbol{D}_C^{(k+1)} - \boldsymbol{D}_C^{(k)}$
  else
    $\boldsymbol{H} \leftarrow \boldsymbol{I}$
    Shrink the simplex

3.4 Surrogate Optimization

In *surrogate optimization methods*, the (complex) objective is replaced with a simpler approximation that maintains the key properties of the objective (e.g. a noisy measurement might be replaced by a simpler regression model). This surrogate function is then minimized and adjusted in order to find a good approximation of the solution of the original problem (this can also be applied for constraint functions).

Requirements for the surrogate function $\hat{\varphi} : \mathbb{R}^{n_p} \to \mathbb{R}$: For all function evaluation (or "sampling") points $\left(\boldsymbol{p}^{(i)}, \varphi\left(\boldsymbol{p}^{(i)}\right)\right)$, $i = 1, \cdots, m$ it must hold that

$$\varphi\left(\boldsymbol{p}^{(i)}\right) = \hat{\varphi}\left(\boldsymbol{p}^{(i)}\right) + \epsilon$$

where $\epsilon \in \mathbb{R}$ is some "slack" constant that determines how exact the surrogate function shall be.

- For $\epsilon = 0$, the problem is the same as interpolation.
- For $\epsilon > 0$, the surrogate function does not perfectly reproduce the objective, but might be smoother.

Further requirements are that $\hat{\varphi}$ should be fast to compute and the gradients of $\hat{\varphi}$ should be available in closed form.

This raises some questions:

1. φ might be too complex for a simple approximation $\implies$ which approximation method should be used?
2. How to generate the data basis of the function evaluations? Generating all in one point will not yields good results...
3. Which method is feasible to minimize $\hat{\varphi}$?

3.4.1 Approximation Methods

Response Surface Methods (RSMs)

Response surface methods use simple polynomials of a low degree as the model function $\hat{\varphi}$, e.g.:

- Degree one (linear): $\hat{\varphi}(\boldsymbol{p}) = \beta_0 + \beta_1^T \boldsymbol{p}$
- Degree one with mixed terms: $\hat{\varphi}(\boldsymbol{p}) = \beta_0 + \boldsymbol{\beta}_1^T \boldsymbol{p} + \sum_i \sum_{\substack{j \\ j \neq i}} \beta_2^{i,j} p_i p_j$
- Degree two (quadratic): $\hat{\varphi}(\boldsymbol{p}) = \beta_0 + \boldsymbol{\beta}_1 \boldsymbol{p} + \boldsymbol{p}^T \boldsymbol{\beta}_2 \boldsymbol{p}$
- Higher degree: ...

The unknown parameters $\beta_1 \in \mathbb{R}$, $\boldsymbol{\beta}_2 \in \mathbb{R}^{n_p}$ and $\boldsymbol{\beta}_2 \in \mathbb{R}^{n_p \times n_p}$ can be approximated using least squares:

$$\min_{\beta_1, \boldsymbol{\beta}_2, \boldsymbol{\beta}_2} \sum_i \left(\varphi\left(\boldsymbol{p}^{(i)}\right) - \hat{\varphi}\left(\boldsymbol{p}^{(i)}\right)\right)$$

- Advantage: Simple and the approximations are easy to compute.
- Disadvantage: The RSMs cover only the global behavior and not local accuracy.

Radial Basis Functions (RBFs)

Now the surrogate function $\hat{\varphi}$ uses a linear combination of *radial basis functions*:

$$\hat{\varphi}(\boldsymbol{p}) = \sum_{i=1}^{m} \gamma_i h\Big(\big\|\boldsymbol{p} - \boldsymbol{p}^{(i)}\big\|\Big)$$

with basis function $h(\cdot)$ based only on the euclidean distance from the interpolation point, e.g.

- Linear: $h(r_i) = r_i$
- Cubic: $h(r_i) = r_i^3$
- Thin-Plate: $h(r_i) = r_i^2 \log r$

where $r_i = \big\|\boldsymbol{p} - \boldsymbol{p}^{(i)}\big\|$.

A suitable combination of RSM and RBF yield cubic spline-approximation:

$$\hat{\varphi}(\boldsymbol{p}) = \beta_0 + \boldsymbol{\beta}_1 \boldsymbol{p} + \sum_i \gamma_i h(\boldsymbol{p}), \quad \beta_1 \in \mathbb{R}, \beta_2 \in \mathbb{R}^{n_p}, \gamma_i \in \mathbb{R}$$

- Univariate: $h(\boldsymbol{p}) = \frac{1}{12} \sum_i r_i^3$
- Bivariate: $h(\boldsymbol{p}) = \frac{1}{16\pi} \sum_i r_i^2 \log r_i$

Design and Analysis of Computer Experiments (DACE)

Assuming the model function is a realization of a stochastic process

$$\hat{\varphi}(\boldsymbol{p}) = \boldsymbol{v}^T(\boldsymbol{p})\boldsymbol{\beta} + Z(\boldsymbol{p})$$

where $\boldsymbol{v}(\boldsymbol{p})$ is a vector of basis functions (e.g. RSM, RBF) and $Z(\boldsymbol{p})$ is a stationary random variable that is Gaussian distributed with zero mean. The covariance between two points $\boldsymbol{p}^{(l)}$ and $\boldsymbol{p}^{(k)}$ is given as

$$\mathrm{Cov}\Big[Z\big(\boldsymbol{p}^{(l)}\big), Z\big(\boldsymbol{p}^{(k)}\big)\Big] = \sigma^2 R\big(\boldsymbol{p}^{(l)}, \boldsymbol{p}^{(k)}\big) \quad \text{with} \quad R\big(\boldsymbol{p}^{(l)}, \boldsymbol{p}^{(k)}\big) = \prod_{i=1}^{n_p} e^{-\theta_i d_i^2}, \quad d_i = \big\|\boldsymbol{p}_i^{(l)} - \boldsymbol{p}_i^{(k)}\big\|_2$$

Die unknown parameters $\boldsymbol{\beta}$, $\boldsymbol{\theta}$ and σ^2 are then estimated using statistical estimators, e.g. maximum likelihood.

3.4.2 Select of the Sampling Points

Design of Experiments (DoE)

The classical strategy for selection sampling points, *design of experiments* is mainly designed for physical experiments, not for deterministic ones! Typical approach:

- Classical selection
  - orthogonal arrays
  - latin hypercubes
  - combinations

- Metric-bases methods
  - MiniMax: minimizing the maximal distance between the sampling points
  - MaxiMin: maximizing the minimal distance
- Stochastic selection for Gaussian processes
  - Entropy Design or D-Opt: maximizing the determinant of the covariance matrix
  - A-Opt: depends on the trace of the covariance matrix
  - G-Opt: minimize the maximum mean squared error

3.4.3 Minimizing the Surrogate Function

To successfully minimize the original objective function, the data basis of the surrogate function has to be expanded sequentially. The following sections describe two methods for this, the Strawman and the Shoemaker method.

Strawman

1. Calculate the current minimum of the surrogate function (e.g. using gradient descent).
2. Add the minimum of the surrogate function as a sampling point.
3. Calculate the new surrogate function. Repeat.

Shoemaker

1. Calculate the minimum with a minimal distance to all sampling points.
2. Extend the sampling points by this point.
3. Determine the new surrogate function. Repeat.

DACE-Based, Sequential Update Strategy

- The the expected mean error as a criteria for the quality of the surrogate function.
- Weigh small function values and uncertainties in the approximations.
- There exist different strategies following this basic idea.
- Termination criteria:
  - Number of function evaluations
  - No more improvements in the objective function
  - $\varphi(\boldsymbol{p})$ close to the minimal possible function value
- Sequential methods are better on normal computers, for parallel computers a special scheme should be used.

3.4.4 Discussion

- Independent of the approximation method, the surrogate function has to be minimized one or more times per iteration (depending on the actual method).
- But the effort for the minimization is negligible as one simulation run for the evaluation of φ often takes a lot longer.
- Every method of **??** can be used for minimizing $\hat{\varphi}$ as the gradients are available by design.
- Advantages:
  - $\hat{\varphi}$ is given in closed form as well as the gradientd.
  - Easy to compute, Newton-type methods applicable!
  - "Smooth" surrogate function.
- Disadvantages:
  - Approximation accuracy is limited.
  - The number of sample points rises a lot for high-dimensional problems (curse of dimensionality).

3.5 Comparison

3.5.1 Magnetic Bearing Design

3.5.2 Walking Optimization of a Humanoid Robot

3.6 Discussion

- Advantages:
  - Application is easy, no or little prior knowledge required.
  - Robust toward discontinuities of φ or $\boldsymbol{\nabla}\varphi$.
  - No calculation of gradients necessary.
  - No need to start "close to" a solution.
  - Some methods (e.g. evolutionary algorithms) are highly parallelizable, some are not (e.g. Nelder-Mead).
- Disadvantages:
  - Slow convergence, high number of steps needed and high computation time due to many φ-evaluations.
  - Inefficient for large n_p.
  - Major difficulties for nonlinear constraints in $\boldsymbol{p}$.

4 Gradient-Based Optimization with Constraints

This chapter covers the optimization problems with nonlinear equality- and inequality-constraints:

$$\begin{aligned} &\min_{\boldsymbol{p}\in\mathbb{R}^{n_p}} \varphi(\boldsymbol{p}) \\ \text{subject to} \quad &\boldsymbol{a}(\boldsymbol{p}) = \boldsymbol{0}, \quad \boldsymbol{a} : \mathbb{R}^{n_p} \to \mathbb{R}^{n_a} \\ &\boldsymbol{b}(\boldsymbol{p}) \geq \boldsymbol{0}, \quad \boldsymbol{b} : \mathbb{R}^{n_p} \to \mathbb{R}^{n_b} \end{aligned}$$

Such an optimization problem is called a *nonlinear programming problem* (NLP).

- A point $\boldsymbol{p} \in \mathbb{R}^{n_p}$ that fulfills the constraints is called a *feasible point*.
- The set of all feasibly points is called the *feasible region*.
- For a local minimum $\boldsymbol{p}^*$ it holds that $\varphi(\boldsymbol{p}^*) \leq \varphi(\boldsymbol{p})$ for all feasible points $\boldsymbol{p}$ in a neighborhood of $\boldsymbol{p}^*$.
- All constraints that are *active* at a point $\boldsymbol{p}$ make up the *active set*. This set includes all equality constraints and the active inequality constraints $A(\boldsymbol{p}) \coloneqq \{\, j \in \mathbb{N} : 1 \leq j \leq n_b,\, b_j(\boldsymbol{p}) = 0 \,\}$.

4.1 Solution Characterization

By a geometric view it is clear that the gradient of the active constraints must be parallel to the gradient of the objective function, i.e. there has to be a constant μ_i^* for each active constraint a_i such that

$$\boldsymbol{\nabla}\varphi(\boldsymbol{p}^*) = \mu_i^* \boldsymbol{\nabla} a_i(\boldsymbol{p}^*) \quad \iff \quad \boldsymbol{\nabla}\varphi(\boldsymbol{p}^*) - \mu_i^* \boldsymbol{\nabla} a_i(\boldsymbol{p}^*) = \boldsymbol{0}$$

This leads directly leads to the definition of the Lagrangian

$$L(\boldsymbol{p}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \varphi(\boldsymbol{p}) - \boldsymbol{\mu}^T \boldsymbol{a}(\boldsymbol{p}) - \boldsymbol{\sigma}^T \boldsymbol{b}(\boldsymbol{p})$$

with the Lagrange multiplier $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ which encodes the insight above.

For formulating the *first-order optimality conditions* analogous to the ones for unconstrained optimization (see subsection 2.1.2), a *constraint qualification* must hold: The gradients

$$\boldsymbol{\nabla} a_1(\boldsymbol{p}^*), \cdots, \boldsymbol{\nabla} a_{n_p}(\boldsymbol{p}^*) \quad \text{and} \quad \boldsymbol{\nabla} b_j(\boldsymbol{p}^*), j \in A(\boldsymbol{p}^*)$$

of the active constraints have to linearly independent.

4.1.1 First-Order Necessary Optimality Conditions (Karush-Kuhn-Tucker Conditions, KKT)

Let $\varphi : \mathbb{R}^{n_p} \to \mathbb{R}$, $\boldsymbol{a} : \mathbb{R}^{n_p} \to \mathbb{R}^{n_a}$ and $\boldsymbol{b} : \mathbb{R}^{n_p} \to \mathbb{R}^{n_b}$ be continuously differentiable and let the constraint qualification be fulfilled. If $\boldsymbol{p}$ is a feasible local minimum of the NLP, then there exist Lagrange multiplier $\boldsymbol{\mu} \in \mathbb{R}^{n_a}$, $\boldsymbol{\sigma} \in \mathbb{R}^{n_b}$ such that the *Karush-Kuhn-Tucker* conditions hold:

$$\boldsymbol{\nabla}_{\boldsymbol{p}} L(\boldsymbol{p}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \boldsymbol{0} \tag{KKT.i}$$

$$\boldsymbol{\mu}^T \boldsymbol{a}(\boldsymbol{p}) = \boldsymbol{0} \tag{KKT.iia}$$

$$\boldsymbol{\sigma}^T \boldsymbol{b}(\boldsymbol{p}) = \boldsymbol{0} \tag{KKT.iib}$$

$$\boldsymbol{\sigma} \geq \boldsymbol{0} \tag{KKT.iii}$$

Here, $L(\boldsymbol{p}, \boldsymbol{\mu}, \boldsymbol{\sigma})$ is the Lagrangian

$$L(\boldsymbol{p}, \boldsymbol{\mu}, \boldsymbol{\sigma}) \coloneqq \varphi(\boldsymbol{p}) - \boldsymbol{\mu}^T \boldsymbol{a}(\boldsymbol{p}) - \boldsymbol{\sigma}^T \boldsymbol{b}(\boldsymbol{p})$$

and the inequality in (KKT.iii) is element-wise.

4.1.2 Second-Order Necessary Optimality Conditions

The second-order necessary optimality condition is fulfilled if the first-order condition is fulfilled and the Hessian of the Lagrangian has a positive curvature *along the feasible directions*:

$$\boldsymbol{z}^T \boldsymbol{H}_L^{\boldsymbol{p}}(\boldsymbol{p}, \boldsymbol{\mu}, \boldsymbol{\sigma}) \boldsymbol{z} \geq 0$$
$$\text{for all } \boldsymbol{z} \in \mathbb{R}^{n_p} \setminus \{\boldsymbol{0}\} \text{ with } \left(\boldsymbol{z}^T \cdot \boldsymbol{\nabla} a_i(\boldsymbol{p}) = \boldsymbol{0}\right)_{i=1,\cdots,n_a}, \text{ and } \left(\boldsymbol{z}^T \cdot \boldsymbol{\nabla} b_j(\boldsymbol{p}) = \boldsymbol{0}\right)_{j \in A(\boldsymbol{p})}$$

4.1.3 Example

4.2 Simple Bounds, Box Constraints

The most common type of inequality-constraints are simple lower and upper bounds on the optimization variables:

$$\min_{\boldsymbol{p} \in \mathbb{R}^{n_p}} \varphi(\boldsymbol{p})$$
$$\text{subject to} \quad \boldsymbol{p}_{i,\min} \leq \boldsymbol{p}_i \leq \boldsymbol{p}_{i,\max}$$

In practice, these are not only used in nearly all constrained, but also in unconstrained optimization problems as they might increase the efficiency by constraining the search space. They also increase robustness of optimization techniques that ensure the fulfilling of the constrains in every iteration.

Box constraints can also be used to prohibit “insecure values”, e.g. negative variables when the objective function takes square roots.

For gradient-free methods, the constraints can be ensured by clipping the new iteration value to the bounds:

$$p_i^{(k+1)} = \max\left\{ p_{i,\min}, \min\left\{ p_{i,\max}, p_i^{(k+1)} \right\} \right\}$$

For gradient-based methods, box constraints can be considered like every other linear and nonlinear constraint (more on that later).

4.3 Penalty Function

The approach of *penalty functions* is to transform the constrained problem to an unconstrained problem by punishing violations of the constraints. As this arises big problems with lots of and highly nonlinear constraints, penalty functions are rarely used in practice anymore in favor of sequential quadratic programming (see section 4.5).

As of today, penalty functions are mainly used for step size determination for nonlinear programs and in the application of robust, gradient-free methods in the unconstrained optimization for solving NLPs.

4.3.1 Exterior Penalty Functions

The original nonlinear problem is replaced by an unconstrained problem with a penalty function Φ

$$\min_{\boldsymbol{p}\in\mathbb{R}^{n_p}} \Phi(\boldsymbol{p},\rho), \quad \Phi(\boldsymbol{p},\rho) = \varphi(\boldsymbol{p}) + \rho\sum_j \pi_j(\boldsymbol{p}), \quad \rho\in\mathbb{R}^+$$

with a function π_j per constraint that is positive if the constraint is violated and zero otherwise.

Often a series of unconstrained optimization problems $\Phi(\boldsymbol{p},\rho)$ is solved with an increasing ρ, such that the solution $\boldsymbol{p}^*(\rho)$ gets pushed into the feasible region of the NLP step by step in in the hope of

$$\lim_{\rho\to\infty} \boldsymbol{p}^*(\rho) = \boldsymbol{p}^*$$

where $\boldsymbol{p}^*$ is the real minimum.

This approach is called *exterior penalty functions* as the penalty term is only relevant if $\boldsymbol{p}$ violates the according constraints.

Quadratic Penalty Function

The *quadratic penalty function* is the most common exterior penalty function:

$$\Phi_Q(\boldsymbol{p},\rho) = \varphi(\boldsymbol{p}) + \rho\cdot\frac{1}{2}\left(\sum_{k=1}^{n_a}\big(a_k(\boldsymbol{p})\big)^2 + \sum_{jk=1}^{n_b}\big(\hat{b}_k(\boldsymbol{p})\big)^2\right)$$

Here, $\hat{b}_j(\boldsymbol{p})$ denotes the violation of the inequality constraints:

$$\hat{b}_j(\boldsymbol{p}) = \begin{cases} 0 & \text{iff } b_j(\boldsymbol{p})\geq 0 \\ -b_j(\boldsymbol{p}) & \text{iff } b_j(\boldsymbol{p})<0 \end{cases} = \big|\min\{0,\, b_j(\boldsymbol{p})\}\big|$$

But in the case of $\rho\to\infty$, the Hessian of Φ_C gets more and more ill-condition and may even be singular in the limit.

Example

4.3.2 Interior Penalty Functions

When using exterior penalty functions, it is not guaranteed that every solution of the iterating unconstrained problems fulfills the constraints of the original NLP. Therefore, exterior penalty functions are not applicable if fulfilling the constraints in every iteration is required. *Interior penalty functions* guarantee exactly that: Every subproblem yields a feasible solution.

Given a NLP with only inequality constraints, interior penalty functions use *barrier functions* that have the following properties:

- Value of infinity everywhere except inside the feasible region.
- Continuously differentiable inside the feasible region.
- The values go to $+\infty$ as $\boldsymbol{p}$ gets closer to the edge of the feasible region.

Logarithmic Barrier Function

The *logarithmic barrier function* is the most used interior penalty function:

$$\Phi_B(\boldsymbol{p}, r) = \varphi(\boldsymbol{p}) - r \sum_{k=1}^{n_b} \ln b_i(\boldsymbol{p})$$

The barrier parameter $r > 0$ will be decreased step by step and the solution should converge to the minimum as $r \to 0$.

But in the case of $r \to 0$, the Hessian of Φ_B gets more and more ill-condition and may even be singular in the limit.

Example

4.3.3 Exact Penalty Functions

Neither the quadratic penalty function nor the logarithmic barrier function are “exact” penalty function so that, with an appropriate ρ or respectively r, the solution of the unconstrained NLP yields the exact solution.

One important penalty function of this class is the *exact ℓ_1-penalty function*:

$$\Phi_{\ell_1}(\boldsymbol{p}, \rho) = \varphi(\boldsymbol{p}) + \rho \sum_{k=1}^{n_a} \left|a_k(\boldsymbol{p})\right| + \rho \sum_{k=1}^{n_b} \left|\min\{0,\, b_j(\boldsymbol{p})\}\right|$$

However, the ℓ_1-penalty function is not differentiable! This make the numerical solution difficult. But the minimization yields, for adequate big ρ, the minimum of the NLP!

Example 1

Example 2

4.3.4 Augmented Lagrangian

The downsides of exterior, interior and the ℓ_1-penalty function can be avoided by not using the objective directly, but by using the Lagrangian:

$$L_Q(\boldsymbol{p}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \rho) = \varphi(\boldsymbol{p}) - \boldsymbol{\mu}^T \boldsymbol{a}(\boldsymbol{p}) - \boldsymbol{\sigma}^T \boldsymbol{b}(\boldsymbol{p}) + \rho \cdot \frac{1}{2} \left(\sum_{k=1}^{n_a} \big(a_k(\boldsymbol{p})\big)^2 + \sum_{jk=1}^{n_b} \big(\min\{0,\, b_j(\boldsymbol{p})\}\big)^2 \right)$$

But this requires a good approximation of the Lagrange multiplier $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

Example 1

Example 2

Notes

- In practice, the augmented Lagrangian L_Q is used in the following fashion:
  1. Choose Lagrange multipliers $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and the parameter $\rho > 0$.
  2. Calculate a local minimum $\boldsymbol{p}^*(\rho)$ of L_Q using methods of the unconstrained optimization.
  3. Update the Lagrange multipliers and ρ. Repeat with 2.
- The update of the Lagrange multipliers according to $\boldsymbol{\mu}(\rho) = -\rho \boldsymbol{a}\big(\boldsymbol{p}^*(\rho)\big)$ is based on the convergence properties of the quadratic penalty function.
- If
  - $\boldsymbol{p}^*$ is a minimum of the NLP,
  - the constraint qualification hold and
  - the KKT-conditions and the second-order sufficient optimality condition is fulfilled for $\boldsymbol{\mu}^*$, $\boldsymbol{\sigma}^*$,

  then
  - exists a threshold $\hat{\rho}$ for which it holds that for all $\rho \geq \hat{\rho}$ it holds that
  - $\boldsymbol{p}^*$ is a strict local minimum of the (quadratic) augmented Lagrangian $L_Q(\boldsymbol{p}, \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*, \rho)$.

4.4 Constraint Elimination

For NLPs with active constraints it is often tempting to transform the NLP to remove some of the constraints and to reduce the number of optimization variables (the degrees of freedom). But this might yields wrong results! It has to be assured that:

- The original minimum is not eliminated.
- The nonlinearity of the problem is not increased too much (causing problems in finite difference approximations of derivatives).
- No singularities arise in the transformed objective.
- No new discontinuities and non-differentiabilities are added to the objective.

- The Hessian of the surrogate problem is not singular or ill-conditioned near the minimum.
- The transformed problem does not have additional local minima or stationary points.
- And a lot more.

In general: Keep more constraints and keep the function as linear as possible instead of applying transformations that behave badly.

Example 1

Example 2

Example 3

4.5 Sequential Quadratic Programming (SQP)

To take care of highly nonlinear equality and inequality constraints, information about the trajectory of these have to be considered, i.e. gradient and Hessian information. Assuming that the constraints that are active on the solution are known, the optimization problem is given as:

$$\begin{aligned} &\min_{\boldsymbol{p}\in\mathbb{R}^{n_p}} \varphi(\boldsymbol{p}) \\ &\text{subject to} \quad \boldsymbol{a}(\boldsymbol{p}) = \boldsymbol{0}, \quad \boldsymbol{a}: \mathbb{R}^{n_p} \to \mathbb{R}^{n_a} \end{aligned}$$

The KKT-conditions are then equivalent to the simpler formulation

$$\boldsymbol{\nabla} L(\boldsymbol{p},\boldsymbol{\mu}) \coloneqq \begin{bmatrix} \boldsymbol{\nabla}_{\boldsymbol{p}} L(\boldsymbol{p},\boldsymbol{\mu}) \\ \boldsymbol{\nabla}_{\boldsymbol{\mu}} L(\boldsymbol{p},\boldsymbol{\mu}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\nabla}\varphi(\boldsymbol{p}) - \sum_{k=1}^{n_a} \mu_k \cdot \boldsymbol{\nabla} a_k(\boldsymbol{p}) \\ -\boldsymbol{a}(\boldsymbol{p}) \end{bmatrix} = \boldsymbol{0}$$

with the Lagrangian

$$L(\boldsymbol{p},\boldsymbol{\mu}) = \varphi(\boldsymbol{p}) - \boldsymbol{\mu}^T \boldsymbol{a}(\boldsymbol{p})$$

This yields a system of $n_p + n_a$ nonlinear equations for $n_p + n_a$ unknowns $\boldsymbol{p}$, $\boldsymbol{\mu}$.

Taylor-expanding the gradient of the Lagrangian around $\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}\right)$ yields

$$\boldsymbol{\nabla} L(\boldsymbol{p}^*,\boldsymbol{\mu}^*) \overset{T\left(\boldsymbol{p}^{(k)},\boldsymbol{\mu}^{(k)}\right)}{=} \boldsymbol{\nabla} L\left(\boldsymbol{p}^{(k)},\boldsymbol{\mu}^{(k)}\right) + \boldsymbol{H}_L\left(\boldsymbol{p}^{(k)},\boldsymbol{\mu}^{(k)}\right) \begin{bmatrix} \boldsymbol{d}_p^{(k)} \\ \boldsymbol{d}_\mu^{(k)} \end{bmatrix} + \cdots \overset{!}{=} 0$$

with $\boldsymbol{d}_p^{(k)} \coloneqq \boldsymbol{p}^* - \boldsymbol{p}^{(k)}$ and $\boldsymbol{d}_\mu^{(k)} \coloneqq \boldsymbol{\mu}^* - \boldsymbol{\mu}^{(k)}$. Cutting off the higher-order terms yields the *Lagrange-Newton method* where the search direction $\boldsymbol{d}_p^{(k)}$, $\boldsymbol{d}_\mu^{(k)}$ is given as the solution of

$$\boldsymbol{H}_L\left(\boldsymbol{p}^{(k)},\boldsymbol{\mu}^{(k)}\right) \begin{bmatrix} \boldsymbol{d}_p^{(k)} \\ \boldsymbol{d}_\mu^{(k)} \end{bmatrix} = -\boldsymbol{\nabla} L\left(\boldsymbol{p}^{(k)},\boldsymbol{\mu}^{(k)}\right) \tag{4.1}$$

The Iteration equation $k \to k+1$ is analogous to the Newton method:

$$\begin{bmatrix} \boldsymbol{p}^{(k+1)} \\ \boldsymbol{\mu}^{(k+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}^{(k)} \\ \boldsymbol{\mu}^{(k)} \end{bmatrix} + \begin{bmatrix} \boldsymbol{d}_p^{(k)} \\ \boldsymbol{d}_\mu^{(k)} \end{bmatrix}$$

But there is one catch: The set of active constraints at the minimum is generally not known and can change in every iteration.

4.5.1 Finding the Search Direction

Further analysis of the linear system (4.1) with the Lagrangian

$$L(\boldsymbol{p}, \boldsymbol{\mu}) = \varphi(\boldsymbol{p}) - \boldsymbol{\mu}^T \boldsymbol{a}(\boldsymbol{p})$$

yields that the linear system has the following structure:

$$\begin{bmatrix} \boldsymbol{H}_L^{\boldsymbol{p}}\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}\right) & -\boldsymbol{J}_{\boldsymbol{a}}\left(\boldsymbol{p}^{(k)}\right) \\ -\boldsymbol{J}_{\boldsymbol{a}}^T\left(\boldsymbol{p}^{(k)}\right) & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_p^{(k)} \\ \boldsymbol{d}_\mu^{(k)} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right) + \boldsymbol{J}_{\boldsymbol{a}}\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{\mu}^{(k)} \\ \boldsymbol{a}\left(\boldsymbol{p}^{(k)}\right) \end{bmatrix}$$

Where $\boldsymbol{H}_L^p\left(\boldsymbol{p}^{(k)}\right)$ is the Hessian of the Lagrangian w.r.t. $\boldsymbol{p}$ and $\boldsymbol{J}_{\boldsymbol{a}}\left(\boldsymbol{p}^{(k)}\right)$ is the Jacobian

$$\boldsymbol{J}_{\boldsymbol{a}}\left(\boldsymbol{p}^{(k)}\right) = \begin{bmatrix} \frac{\partial a_1}{\partial p_1} & \cdots & \frac{\partial a_{n_a}}{\partial p_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_1}{\partial p_{n_p}} & \cdots & \frac{\partial a_{n_a}}{\partial p_{n_p}} \end{bmatrix}$$

Note that this definition of the Jacobian differs from the usual definition! This one has the gradients of the function has columns!

This linear system can be transformed to

$$\begin{bmatrix} \boldsymbol{H}_L^{\boldsymbol{p}}\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}\right) & -\boldsymbol{J}_{\boldsymbol{a}}\left(\boldsymbol{p}^{(k)}\right) \\ -\boldsymbol{J}_{\boldsymbol{a}}^T\left(\boldsymbol{p}^{(k)}\right) & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_p^{(k)} \\ \underbrace{\boldsymbol{d}_\mu^{(k)} + \boldsymbol{\mu}^{(k)}}_{\boldsymbol{\mu}^{(k+1)}} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right) \\ \boldsymbol{a}\left(\boldsymbol{p}^{(k)}\right) \end{bmatrix}$$

where $\boldsymbol{d}_p^{(k)}$ can be viewed as the solution of a quadratic minimization problem!

Quadratic Problem (QP)

The said quadratic problem is given as:

$$\begin{aligned} \min_{\boldsymbol{d}_p \in \mathbb{R}^{n_p}} \; & \varphi\left(\boldsymbol{p}^{(k)}\right) + \left(\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)\right)^T \boldsymbol{d}_p + \frac{1}{2}\boldsymbol{d}_p^T \boldsymbol{H}_L^{\boldsymbol{p}}\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}\right)\boldsymbol{d}_p \\ \text{subject to} \quad & \boldsymbol{a}\left(\boldsymbol{p}^{(k)}\right) + \boldsymbol{J}_a^T\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{d}_p = \boldsymbol{0} \end{aligned}$$

Where the quadratic objective function of the QP consists of a quadratic Taylor-approximation of the NLP objective plus a weighted curvature condition via the Hessian of the activate conditions:

$$\boldsymbol{H}_L^{\boldsymbol{p}}\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}\right) = \boldsymbol{H}_\varphi\left(\boldsymbol{p}^{(k)}\right) - \sum_{i=1}^{n_a} \mu_i^T \boldsymbol{H}_{a_i}\left(\boldsymbol{p}^{(k)}\right)$$

The linear constraints of the QP also consist of Taylor-approximations of the active NLP constraints.

Solving this quadratic optimization problem is more robust and possibly faster than solving the linear system of equations. There exist special algorithms for solving QPs. But even though they are not active, the inequality constraints must be fulfilled. Therefore, the QP is extended to also determine the Lagrange multipliers $\boldsymbol{\mu}^{(k)}$, $\boldsymbol{\sigma}^{(k)}$ along with the search direction $\boldsymbol{d}_p^{(k)}$:

$$\begin{aligned} \min_{\boldsymbol{d}_p \in \mathbb{R}^{n_p}} \; & \varphi\left(\boldsymbol{p}^{(k)}\right) + \left(\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)\right)^T \boldsymbol{d}_p + \frac{1}{2}\boldsymbol{d}_p^T \boldsymbol{H}_L^{\boldsymbol{p}}\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}, \boldsymbol{\sigma}^{(k)}\right)\boldsymbol{d}_p \\ \text{subject to} \quad & \boldsymbol{a}\left(\boldsymbol{p}^{(k)}\right) + \boldsymbol{J}_{\boldsymbol{a}}^T\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{d}_p = \boldsymbol{0} \\ & \boldsymbol{b}\left(\boldsymbol{p}^{(k)}\right) + \boldsymbol{J}_{\boldsymbol{b}}^T\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{d}_p \geq \boldsymbol{0} \end{aligned} \tag{4.2}$$

Iteration step $k \to k+1$: $\boldsymbol{p}^{(k+1)} = \boldsymbol{p}^{(k)} + \boldsymbol{d}_p^{(k)}$

Notes

- Similar to the Newton method, it can be shown under some assumptions that the SQP method converges to a local minimum.
- Algorithms for solving general QPs can be used to determine the active constraints in each iteration by solving the QP.
  Even though it is not needed, it is useful for efficiency to use as much information as possible from the last iteration ("hot start").
- Other approaches determine the active constraints (working set) outside of the QP solution to only solve QPs with equality constraints which is especially efficient.

4.5.2 Step Size Rules

If the initialization $\boldsymbol{p}^{(0)}$ is "far" away from the minimum (or the QP is a bad, local approximation of the NLP), the convergence can be improved by determining the optimal step size for

$$\boldsymbol{p}^{(k+1)} = \boldsymbol{p}^{(k)} + \alpha^{(k)} \boldsymbol{d}_p^{(k)} \quad \text{or respectively} \quad \begin{bmatrix} \boldsymbol{p}^{(k+1)} \\ \boldsymbol{\mu}^{(k+1)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}^{(k)} \\ \boldsymbol{\mu}^{(k)} \end{bmatrix} + \alpha^{(k)} \begin{bmatrix} \boldsymbol{d}_p^{(k)} \\ \boldsymbol{\mu}_{QP}^{(k+1)} - \boldsymbol{\mu}^{(k)} \end{bmatrix}$$

Common methods for determining the step size are

- the (quadratic) augmented Lagrangian L_Q or
- the exact ℓ_1-penalty function Φ_{ℓ_1}

with step size rules like the Armijo rule similar as in the unconstrained optimization (see section 2.3).

4.5.3 Approximation of the Lagrange Multipliers

If the approximation $\boldsymbol{p}^{(k)}$ is far away from the NLP solution, it is not useful to use the Lagrange multipliers of the QP for the NLP (as the linearization is only valid locally). Another method is to approximate the Lagrange multipliers after calculating $\boldsymbol{p}^{(k+1)}$ (i.e. solving the QP) using minimum least squares:

$$\min_{\mu \in \mathbb{R}^{n_a}} \left\| \nabla \varphi\left(\boldsymbol{p}^{(k+1)}\right) - \sum_{i=1}^{n_a} \mu_i^{(k+1)} \cdot \nabla a_i\left(\boldsymbol{p}^{(k+1)}\right) \right\|_2^2$$

or equivalently model the active inequality constraints explicitly:

$$\min_{\mu \in \mathbb{R}^{n_a}} \left\| \nabla \varphi\left(\boldsymbol{p}^{(k+1)}\right) - \sum_{i=1}^{n_a} \mu_i^{(k+1)} \cdot \nabla a_i\left(\boldsymbol{p}^{(k+1)}\right) - \sum_{i \in A\left(\boldsymbol{p}^{(k+1)}\right)} \sigma_i^{(k+1)} \cdot \nabla a_i\left(\boldsymbol{p}^{(k+1)}\right) \right\|_2^2$$

Solving least squares problem will be discussed in detail in chapter 6.

4.5.4 Termination Criteria

An obvious termination criteria would be to check whether the necessary KKT-conditions are sufficiently fulfilled, i.a. $\nabla L(\boldsymbol{p}, \boldsymbol{\mu}) \approx \mathbf{0}$.

The commonly used SQP method *NPSOL* terminates if all of the following criteria are fulfilled:

1. Old and new approximation do not change any more:

$$\left\|\boldsymbol{p}^{(k+1)} - \boldsymbol{p}^{(k)}\right\| = \alpha^{(k)} \cdot \left\|\boldsymbol{d}_p^{(k)}\right\|_2 \leq \sqrt{\varepsilon_{\text{opt}}}\left(1 + \left\|\boldsymbol{p}^{(k+1)}\right\|_2\right)$$

2. Gradient of the objective function that is projected onto the active constraints vanishes:

$$\left\|\boldsymbol{Z}^T \cdot \nabla\varphi\left(\boldsymbol{p}^{(k+1)}\right)\right\|_2 \leq \sqrt{\varepsilon_{\text{opt}}}\left(1 + \max\left\{1 + \left|\varphi\left(\boldsymbol{p}^{(k+1)}\right)\right|,\, \left\|\nabla\varphi\left(\boldsymbol{p}^{(k+1)}\right)\right\|_2\right\}\right)$$

3. Constraints are sufficiently fulfilled:

$$\begin{aligned} \left|a_i\left(\boldsymbol{p}^{(k+1)}\right)\right| &\leq \varepsilon_{\text{ft}}, \quad i = 1, \cdots, n_a \\ b_j\left(\boldsymbol{p}^{(k+1)}\right) &\geq -\varepsilon_{\text{ft}}, \quad i = 1, \cdots, n_b \end{aligned}$$

Where the constraints ε_{opt}, ε_{ft} have to be chosen by the user.

In the more modern method *SNOPT*, all of the following two criteria have to be fulfilled:

$$\frac{\max_{i=1,\cdots,n_p}\left\{\left|\frac{\partial\varphi\left(\boldsymbol{p}^{(k)}\right)}{\partial p_i} - \sum_{j=1}^{n_a}\mu_j^{(k)} \cdot \frac{\partial a_j\left(\boldsymbol{p}^{(k)}\right)}{\partial p_i} - \sum_{l\in A\left(\boldsymbol{p}^{(k)}\right)}\sigma_j^{(k)} \cdot \frac{\partial b_l\left(\boldsymbol{p}^{(k)}\right)}{\partial p_i}\right|\right\}}{\sqrt{\sum_{j=1}^{n_a}\left(\mu_j^{(k)}\right)^2 + \sum_{l\in A\left(\boldsymbol{p}^{(k)}\right)}\left(\sigma_j^{(k)}\right)^2}} \leq \varepsilon_{\text{opt}}$$

$$\frac{\max\left\{\left|a_j\left(\boldsymbol{p}^{(k)}\right)\right| : i = 1, \cdots, n_p\right\} \cup \left\{\left|\min\left\{0,\, b_l\left(\boldsymbol{p}^{(k)}\right)\right\}\right| : l \in A\left(\boldsymbol{p}^{(k)}\right)\right\}}{\left\|\boldsymbol{p}^{(k)}\right\|_2} \leq \varepsilon_{\text{ft}}$$

A common criteria to detect failures is a maximum number of iterations $k_{\max}$.

4.5.5 Hessian Approximation

The quadratic problem(4.2) needs the $(n_p \times n_p)$-dimensional Hessian of the Lagrangian

$$\boldsymbol{H}_L^{\boldsymbol{p}}\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}, \boldsymbol{\sigma}^{(k)}\right) = \boldsymbol{H}_\varphi\left(\boldsymbol{p}^{(k)}\right) - \sum_{i=1}^{n_a}\mu_i^{(k)}\boldsymbol{H}_{a_i}\left(\boldsymbol{p}^{(k)}\right) - \sum_{i\in A\left(\boldsymbol{p}^{(k)}\right)}\sigma_i^{(k)}\boldsymbol{H}_{b_i}\left(\boldsymbol{p}^{(k)}\right)$$

for (theoretical) quadratic convergence of the SQP method. But in practice, the second-order derivatives (the Hessian) is often not available! Additionally it is assumed that the Hessian has a positive curvature (i.e. is positive definite) along all feasible directions, which is fulfilled near a strict minimum. However, this cannot be assumed in every iteration.

Hence, approximations/modifications of the Hessian are required.

Naïve Approach: BFGS Approximation

It is tempting to use the BFGS update for the Hessian that is really successful in the unconstrained optimization. The update rule is given as:

$$\tilde{\boldsymbol{H}}^{(k+1)} = \tilde{\boldsymbol{H}}^{(k)} - \frac{1}{\left(\boldsymbol{d}^{(k)}\right)^T \tilde{\boldsymbol{H}}^{(k)} \boldsymbol{d}^{(k)}} \tilde{\boldsymbol{H}}^{(k)} \boldsymbol{d}^{(k)} \left(\tilde{\boldsymbol{H}}^{(k)} \boldsymbol{d}^{(k)}\right)^T + \frac{1}{\left(\boldsymbol{g}^{(k)}\right)^T \boldsymbol{d}^{(k)}} \boldsymbol{g}^{(k)} \left(\boldsymbol{g}^{(k)}\right)^T$$
$$\boldsymbol{g}^{(k)} = \boldsymbol{\nabla}_{\boldsymbol{p}} L\left(\boldsymbol{p}^{(k+1)}, \boldsymbol{\mu}^{(k+1)}, \boldsymbol{\sigma}^{(k+1)}\right) - \boldsymbol{\nabla}_{\boldsymbol{p}} L\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k+1)}, \boldsymbol{\sigma}^{(k+1)}\right)$$

But it is not necessary that the full Hessian of the Lagrangian is positive definite! Additionally, it is very inefficient to calculate the full Hessian if the NLP has lots of active constraints. Hence, the approximation has to be modified in order to be useful for SQP methods.

Reduced Hessian

Every active constraints reduces the degrees of freedom by one Hence the degrees of freedom are the number of optimization variables n_p minus the number of active and linearly independent constraints. In all of the following it is assumed that it is known which constraints are active at the solution, yielding the following, simpler, view of the NLP:

$$\begin{aligned} \min_{\boldsymbol{d}_p \in \mathbb{R}^{n_p}} \; & \varphi\left(\boldsymbol{p}^{(k)}\right) + \left(\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)\right)^T \boldsymbol{d}_p + \frac{1}{2} \boldsymbol{d}_p^T \boldsymbol{H}_L^{\boldsymbol{p}}\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}, \boldsymbol{\sigma}^{(k)}\right) \boldsymbol{d}_p \\ \text{subject to} \quad & \boldsymbol{a}\left(\boldsymbol{p}^{(k)}\right) + \boldsymbol{J}_{\boldsymbol{a}}^T\left(\boldsymbol{p}^{(k)}\right) \boldsymbol{d}_p = \boldsymbol{0} \end{aligned} \tag{4.3}$$

where the last Jacobian $\boldsymbol{J}_{\boldsymbol{a}}^T$ might also contain active inequality constraints which are left out here for brevity. But they are handled the same as regular equality constraints and can be considered to be part of it knowing which are active (as assumed). The degrees of freedom of this NLP are therefore $n_p - n_a$.

Assuming the current approximation $\boldsymbol{p}^{(k)}$ fulfills the constraints, $\boldsymbol{a}\left(\boldsymbol{p}^{(k)}\right) = \boldsymbol{0}$, the next approximation also has to fulfill the constraints:

$$\boldsymbol{a}\left(\boldsymbol{p}^{(k)} + \boldsymbol{d}_p\right) = \boldsymbol{0}$$

By Taylor-expanding this equation around $\boldsymbol{p}^{(k)}$

$$\boldsymbol{a}\left(\boldsymbol{p}^{(k)} + \boldsymbol{d}_p\right) \overset{T\left(\boldsymbol{p}^{(k)}\right)}{=} \boldsymbol{a}\left(\boldsymbol{p}^{(k)}\right) + \boldsymbol{J}_{\boldsymbol{a}}^T\left(\boldsymbol{p}^{(k)}\right) \boldsymbol{d}_p + \cdots = 0$$

a linear system for the search direction $\boldsymbol{d}_p$ can be found such that the new iteration is also feasible:

$$\boldsymbol{J}_{\boldsymbol{a}}^T\left(\boldsymbol{p}^{(k)}\right) \boldsymbol{d}_p = \boldsymbol{0}$$

Thus, $\boldsymbol{d}_p$ has to lie in the kernel of $\boldsymbol{J}_{\boldsymbol{a}}^T$ and the dimensionality of the kernel is $n_p - n_a$.

Hence, the kernel is spanned by $n_p - n_a$ basis vectors (which are not uniquely determined). Let $\boldsymbol{Z}^{(k)}$ be the matrix that contains these basis vectors as columns, then

$$\boldsymbol{J}_{\boldsymbol{a}}^T\left(\boldsymbol{p}^{(k)}\right) \boldsymbol{Z}^{(k)} = \boldsymbol{0}$$

holds and the search direction $\boldsymbol{d}_p \in \mathbb{R}^{n_p}$ can be represented as

$$\boldsymbol{d}_p = \boldsymbol{Y}^{(k)} \boldsymbol{d}_p^y + \boldsymbol{Z}^{(k)} \boldsymbol{d}_p^z$$

where $\boldsymbol{Z}^{(k)} \in \mathbb{R}^{n_p \times (n_p - n_a)}$ are the basis vectors of the kernel and $\boldsymbol{Y}^{(k)} \in \mathbb{R}^{n_p \times n_a}$ are the basis vectors of the image space of $\boldsymbol{J}_{\boldsymbol{a}}^T(\boldsymbol{p}^{(k)})$. The vectors $\boldsymbol{d}_p^y \in \mathbb{R}^{n_a}$ and $\boldsymbol{d}_p^z \in \mathbb{R}^{n_p - n_a}$ are unknown and have to be computed to solve the QP. Plugging this formulation of $\boldsymbol{d}_p$ into the constraints of (4.3):

$$\begin{aligned} & & \mathbf{0} &= \boldsymbol{a}(\boldsymbol{p}^{(k)}) + \boldsymbol{J}_{\boldsymbol{a}}^T(\boldsymbol{p}^{(k)})\boldsymbol{d}_p \\ \iff & & -\boldsymbol{a}(\boldsymbol{p}^{(k)}) &= \boldsymbol{J}_{\boldsymbol{a}}^T(\boldsymbol{p}^{(k)})\boldsymbol{d}_p \\ & &&= \boldsymbol{J}_{\boldsymbol{a}}^T(\boldsymbol{p}^{(k)})(\boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y + \boldsymbol{Z}^{(k)}\boldsymbol{d}_p^z) \\ & &&= \boldsymbol{J}_{\boldsymbol{a}}^T(\boldsymbol{p}^{(k)})\boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y + \underbrace{\boldsymbol{J}_{\boldsymbol{a}}^T(\boldsymbol{p}^{(k)})\boldsymbol{Z}^{(k)}}_{=\mathbf{0}}\boldsymbol{d}_p^z \\ & &&= \boldsymbol{J}_{\boldsymbol{a}}^T(\boldsymbol{p}^{(k)})\boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y \end{aligned} \tag{4.4}$$

The vector $\boldsymbol{d}_p^y$ is therefore determined by the active constraints via the linear system (4.4). Now there remain $n_p - n_a$ degrees of freedom for the actual optimization.

Plugging the formulation of $\boldsymbol{d}_p$ into the objective of the QP[1]

$$\begin{aligned} &\varphi(\boldsymbol{p}^{(k)}) + \left(\nabla\varphi(\boldsymbol{p}^{(k)})\right)^T \boldsymbol{d}_p + \frac{1}{2}\boldsymbol{d}_p^T \boldsymbol{H}_L^{\boldsymbol{p}} \boldsymbol{d}_p \\ =&\varphi(\boldsymbol{p}^{(k)}) + \left(\nabla\varphi(\boldsymbol{p}^{(k)})\right)^T (\boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y + \boldsymbol{Z}^{(k)}\boldsymbol{d}_p^z) + \frac{1}{2}(\boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y + \boldsymbol{Z}^{(k)}\boldsymbol{d}_p^z)^T \boldsymbol{H}_L^{\boldsymbol{p}}(\boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y + \boldsymbol{Z}^{(k)}\boldsymbol{d}_p^z) \end{aligned}$$

yields the following objective when leaving out all constant parts w.r.t. $\boldsymbol{d}_p^z$, as that is the optimization variable:

$$\varphi(\boldsymbol{p}^{(k)}) + \left(\nabla\varphi(\boldsymbol{p}^{(k)})\right)^T \boldsymbol{Z}^{(k)}\boldsymbol{d}_p^z + (\boldsymbol{d}_p^y)^T (\boldsymbol{Z}^{(k)})^T \boldsymbol{H}_L^{\boldsymbol{p}}(\boldsymbol{Z}^{(k)}\boldsymbol{d}_p^z) + \frac{1}{2}(\boldsymbol{d}_p^z)^T (\boldsymbol{Z}^{(k)})^T \boldsymbol{H}_L^{\boldsymbol{p}} \boldsymbol{Z}^{(k)}\boldsymbol{d}_p^z$$

The solution of this optimization problem can be computed by solving the following linear system (if the reduced Hessian is positive definite, which it is close to a strict local minimum):

$$\left((\boldsymbol{Z}^{(k)})^T \boldsymbol{H}_L^{(k)} \boldsymbol{Z}^{(k)}\right)\boldsymbol{d}_p^z = -(\boldsymbol{Z}^{(k)})^T \boldsymbol{H}_L^{(k)} \boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y - (\boldsymbol{Z}^{(k)})^T \cdot \nabla\varphi(\boldsymbol{p}^{(k)})$$

Summary Using the representation $\boldsymbol{d}_p = \boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y + \boldsymbol{Z}^{(k)}\boldsymbol{d}_p^z$, the search direction as the solution of the QP (4.3) can be computed by solving two staggered linear systems:

$$\begin{aligned} \boldsymbol{J}_{\boldsymbol{a}}^T(\boldsymbol{p}^{(k)})\boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y &= -\boldsymbol{a}(\boldsymbol{p}^{(k)}) \\ \left((\boldsymbol{Z}^{(k)})^T \boldsymbol{H}_L^{(k)} \boldsymbol{Z}^{(k)}\right)\boldsymbol{d}_p^z &= -(\boldsymbol{Z}^{(k)})^T \boldsymbol{H}_L^{(k)} \boldsymbol{Y}^{(k)}\boldsymbol{d}_p^y - (\boldsymbol{Z}^{(k)})^T \cdot \nabla\varphi(\boldsymbol{p}^{(k)}) \end{aligned}$$

- If $\boldsymbol{a}(\boldsymbol{p}^{(k)}) = \mathbf{0}$, i.e. the constraints are linear, $\boldsymbol{d}_p^y = \mathbf{0}$.
- If additionally $(\boldsymbol{Z}^{(k)})^T \cdot \nabla\varphi(\boldsymbol{p}^{(k)}) = \mathbf{0}$, then $\boldsymbol{d}_p^z = \mathbf{0}$.

First- and Second-Order Conditions Using the matrix $\boldsymbol{Z}$ some of the necessary and sufficient conditions can be formulated equivalent:

- First-order necessary condition:

[1] Note that the parameters of the Hessian are kept implicitly for brevity.

$$\nabla\varphi(\boldsymbol{p}^*) - \sum_{i=1}^{n_a} \mu_i \cdot \nabla a_i(\boldsymbol{p}^*) - \sum_{i \in A(\boldsymbol{p}^*)} = \boldsymbol{0} \quad \Longleftrightarrow \quad \boldsymbol{Z}^T(\boldsymbol{p}^*) \cdot \nabla\varphi(\boldsymbol{p}^*) = \boldsymbol{0}$$

- Second-order necessary condition:
  The reduced Hessian of the Lagrangian $\boldsymbol{Z}^T(\boldsymbol{p}^*) \cdot \boldsymbol{H}_L^{\boldsymbol{p}}(\boldsymbol{p}^*, \boldsymbol{\mu}^*, \boldsymbol{\sigma}^*) \cdot \boldsymbol{Z}(\boldsymbol{p}^*)$ is positive semidefinite.
- Second-order sufficient condition:
  The reduced Hessian of the Lagrangian is positive definite.

Example

Approximation of the Reduced Hessian

The reduced Hessian contains all information that is needed to compute the QP solution! Hence, SQP methods can be built on this reduced Hessian. A Quasi-Newton approximation, e.g. BFGS, can be used to update the reduced Hessian:

$$\begin{aligned}
\tilde{\boldsymbol{H}}^{(k+1)} &= \tilde{\boldsymbol{H}}^{(k)} - \frac{1}{\left(\boldsymbol{d}^{(k)}\right)^T \tilde{\boldsymbol{H}}^{(k)} \boldsymbol{d}^{(k)}} \tilde{\boldsymbol{H}}^{(k)} \boldsymbol{d}^{(k)} \left(\tilde{\boldsymbol{H}}^{(k)} \boldsymbol{d}^{(k)}\right)^T + \frac{1}{\left(\boldsymbol{g}^{(k)}\right)^T \boldsymbol{d}^{(k)}} \boldsymbol{g}^{(k)} \left(\boldsymbol{g}^{(k)}\right)^T \\
\boldsymbol{d}^{(k)} &= \boldsymbol{d}_p^z \\
\boldsymbol{g}^{(k)} &= \left(\boldsymbol{Z}^{(k+1)}\right)^T \cdot \nabla_{\boldsymbol{p}} L\left(\boldsymbol{p}^{(k+1)}, \boldsymbol{\mu}^{(k+1)}, \boldsymbol{\sigma}^{(k+1)}\right) - \left(\boldsymbol{Z}^{(k)}\right)^T \cdot \nabla_{\boldsymbol{p}} L\left(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k+1)}, \boldsymbol{\sigma}^{(k+1)}\right)
\end{aligned}$$

Analogous to unconstrained optimization, it is helpful to replace the rank-2 update with a rank-1 update and directly approximate a Cholesky decomposition.

4.5.6 SQP Method (Algorithm)

A sketch of the implementation of an SQP method is given in algorithm 6.

4.5.7 Notes

- When using the reduced Hessian, the matrix $\boldsymbol{Z}^{(k)}$ has to be updated in every iteration.
- Especially for high-dimensional problems, the gradients and Jacobians are sparse, i.e. only a few entries are nonzero. This behavior can be exploited in order to optimize implementations a lot.
- To rise efficiency and robustness further, sophisticated implementations of SQP methods (e.g. NPSOL, SNOPT) differentiate between constraints like
  - upper and lower bounds
  - linear constraints
  - nonlinear constraints

Algorithm 6: Sequential Quadratic Programming

Initialization: Choose an initial approximation $\boldsymbol{p}^{(0)}$, set $k \leftarrow 0$

while *not converged* **do**

Calculate the Lagrange multipliers $\boldsymbol{\mu}^{(k)}$, $\boldsymbol{\sigma}^{(k)}$ using least squares:

$$\min_{\mu \in \mathbb{R}^{n_a}} \left\| \boldsymbol{\nabla} \varphi\big(\boldsymbol{p}^{(k+1)}\big) - \sum_{i=1}^{n_a} \mu_i^{(k+1)} \cdot \boldsymbol{\nabla} a_i\big(\boldsymbol{p}^{(k+1)}\big) - \sum_{i \in A\left(\boldsymbol{p}^{(k+1)}\right)} \sigma_i^{(k+1)} \cdot \boldsymbol{\nabla} a_i\big(\boldsymbol{p}^{(k+1)}\big) \right\|_2^2$$

if *termination criteria fulfilled* **then**

return

Calculate new search direction $\boldsymbol{d}_p^{(k)}$ by solving the quadratic problem:

$$\begin{aligned} \min_{\boldsymbol{d}_p \in \mathbb{R}^{n_p}} \; & \varphi\big(\boldsymbol{p}^{(k)}\big) + \Big(\boldsymbol{\nabla} \varphi\big(\boldsymbol{p}^{(k)}\big)\Big)^T \boldsymbol{d}_p + \frac{1}{2} \boldsymbol{d}_p^T \boldsymbol{H}_L^{\boldsymbol{p}}\big(\boldsymbol{p}^{(k)}, \boldsymbol{\mu}^{(k)}, \boldsymbol{\sigma}^{(k)}\big) \boldsymbol{d}_p \\ \text{subject to} \quad & \boldsymbol{a}\big(\boldsymbol{p}^{(k)}\big) + \boldsymbol{J}_{\boldsymbol{a}}^T\big(\boldsymbol{p}^{(k)}\big) \boldsymbol{d}_p = \boldsymbol{0} \\ & \boldsymbol{b}\big(\boldsymbol{p}^{(k)}\big) + \boldsymbol{J}_{\boldsymbol{b}}^T\big(\boldsymbol{p}^{(k)}\big) \boldsymbol{d}_p \geq \boldsymbol{0} \end{aligned}$$

Calculate the step size $\alpha^{(k)}$ using by minimizing a merit function, e.g. the augmented Lagrangian

$$\min_{\alpha \in \mathbb{R}^+} \; L_Q\big(\boldsymbol{p}^{(k)} + \alpha \boldsymbol{d}_p^{(k)},\, \boldsymbol{\mu}^{(k)} + \alpha \boldsymbol{d}_\mu^{(k)},\, \boldsymbol{\sigma}^{(k)} + \alpha \boldsymbol{d}_\sigma^{(k)},\, \rho^{(k)}\big)$$

or the exact ℓ_1-penalty function $\min_{\alpha \in \mathbb{R}^+} \; \Phi_{\ell_1}\big(\boldsymbol{p}^{(k)} + \alpha \boldsymbol{d}_p^{(k)},\, \rho^{(k)}\big)$

Calculate the new solution approximation:

$$\boldsymbol{p}^{(k+1)} \leftarrow \boldsymbol{p}^{(k)} + \alpha^{(k)} \boldsymbol{d}_p^{(k)}$$

4.5.8 Examples

Optimal Control of a 6-DoF Industry Robot

Car Drive

4.5.9 Wrap-Up

- Motivation:
  - The Newton method is applied to determine a root (“zero point”) of the gradient of the Lagrangian.
  - This yields a linear equation system to determine a search direction.
  - Requires first- and second-order derivatives of the objective and the constraints.
  - Preliminaries for the derivative where differentiability and knowing which constraints are active.
- Basic structure:
  - The linear system derived as a first step is equivalent to the solution of a quadratic problem (QP).
  - Solving the QP is faster and more robust than solving the linear system directly.
  - This yields a sequence of quadratic problems, thus *sequential quadratic programming* (SQP).
- Improvement of the basic structure:
  - For globalizing the method, a step size determination was introduced using line search on an appropriate test function (penalty function).
  - As the second-order derivatives are commonly not available, Quasi-Newton approximations of the Lagrangian are used.
- SQP for high-dimensional NLPs:
  - If the NLP is high-dimensional, the reduced Hessian shall be used.
  - The QP can be solved faster and the Hessian of the Lagrangian can be approximation using Quasi-Newton approaches!
  - High-dimensional NLPs often have sparse gradients as Jacobians which can be used for further performance improvements.
- Outlook:
  - There are lots of SQP methods, e.g. trust-region SQP that can work with indefinite or negative definite Hessians or methods that allow only feasible approximations in every iteration (feasible SQP methods), while the “classic” SQP only fulfills the constraints at the end (infeasible SQP method).
  - Other numerical methods for solving nonlinear, constrained optimization problems exist, e.g. inner point methods.

5 Calculation of Derivatives

To use efficient gradient-based algorithms, the first-order derivatives

$$\frac{\partial \varphi}{\partial p_i} \qquad \frac{\partial a_k}{\partial p_i} \qquad \frac{\partial b_l}{\partial p_i}$$

of the objective and the constraints are needed. But these are typically not available directly! And even one wrong derivative could destroy the fast convergence properties...

5.1 Finite Difference Approximation (numerical Differentiation)

5.1.1 Forward Difference Approximation

The most known approximation of the first derivative is the *forward difference approximation*

$$\frac{\partial \varphi(\boldsymbol{p})}{\partial p_i} \approx D_{V,i}\varphi(\boldsymbol{p}) = \frac{1}{\delta_i}\big(\varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) - \varphi(\boldsymbol{p})\big)$$

where $\boldsymbol{e}_i \in \mathbb{R}^{n_p}$ is the i-th unit vector and δ_i is an appropriate step size. The complete error is:

Error

The complete error of the approximation is composed of the following:

- Approximation error (theoretical error)
- Function precision
- Rounding error

Approximation Error The (theoretical) approximation error is given as the neglected terms of the Taylor approximation (i.e. the Lagrangian remainder):

$$\varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) \overset{T(\boldsymbol{p})}{=} \varphi(\boldsymbol{p}) + \frac{\partial \varphi(\boldsymbol{p})}{\partial p_i}\delta_i + \frac{1}{2}\frac{\partial^2 \varphi(\tilde{\boldsymbol{p}})}{\partial p_i^2}\delta_i^2, \quad \tilde{\boldsymbol{p}} \in [\boldsymbol{p}, \boldsymbol{p} + \boldsymbol{e}_i\delta_i]$$

$$\iff \quad \varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) - \varphi(\boldsymbol{p}) - \frac{\partial \varphi(\boldsymbol{p})}{\partial p_i}\delta_i = \frac{1}{2}\frac{\partial^2 \varphi(\tilde{\boldsymbol{p}})}{\partial p_i^2}\delta_i^2$$

$$\iff \quad \underbrace{\frac{1}{\delta_i}\big(\varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) - \varphi(\boldsymbol{p})\big)}_{= D_{V,i}\varphi(\boldsymbol{p})} - \frac{\partial \varphi(\boldsymbol{p})}{\partial p_i} = \frac{1}{2}\frac{\partial^2 \varphi(\tilde{\boldsymbol{p}})}{\partial p_i^2}\delta_i$$

$$\iff \quad D_{V,i}\varphi(\boldsymbol{p}) - \frac{\partial \varphi(\boldsymbol{p})}{\partial p_i} = \frac{1}{2}\frac{\partial^2 \varphi(\tilde{\boldsymbol{p}})}{\partial p_i^2}\delta_i =: T_{V,i}(\varphi; \delta_i)$$

In theory, the error should decrease with the step size. But today's computers have finite arithmetic! Other error factors have a serious role demolishing this theoretical result.

Function Precision The function precision takes into account that the target function φ cannot be calculated with machine precision, e.g. because the evaluation depends on other methods or because rounding errors have summed up due to cancellation or ill-conditioning.

This can be taken into account with the absolute errors ε, ε_{δ_i} of the function evaluations:

$$\hat{\varphi}(\boldsymbol{p}) = \varphi(\boldsymbol{p}) + \varepsilon \qquad\qquad \hat{\varphi}(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) = \varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) + \varepsilon_{\delta_i}$$

Where the absolute error can be expressed in terms of a relative error $\varepsilon_R = 10^{-n_d}$ as $\varepsilon = \varepsilon_R\varphi(\boldsymbol{p})$ where n_d are the number of decimal places that are correct. Plugging $\hat{\varphi}$ into the forward approximation yields the function precision error $C(D_{V,i}\varphi; \delta_i)$:

$$D_{V,i}\hat{\varphi}(\boldsymbol{p}) = \frac{1}{\delta_i}\big(\hat{\varphi}(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) - \hat{\varphi}(\boldsymbol{p})\big) = \frac{1}{\delta_i}\big(\varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) - \varphi(\boldsymbol{p})\big) + \frac{\varepsilon_{\delta_i} - \varepsilon}{\delta_i} =: D_{V,i}\varphi(\boldsymbol{p}) + C(D_{V,i}\varphi; \delta_i)$$

Rounding Error Additionally to the function precision and the theoretical error, rounding errors are produced by the subtraction and the division. But if δ_i does not get "too small", these are negligible compared to the approximation error and the function precision.

Total Error Hence, the total error is given as:

$$T_{V,i}(\varphi; \delta_i) + C(D_{V,i}\varphi; \delta_i) = \frac{1}{2}\frac{\partial^2\varphi(\tilde{\boldsymbol{p}})}{\partial p_i^2}\delta_i + \frac{\varepsilon_{\delta_i} - \varepsilon}{\delta_i} \tag{5.1}$$

Choosing the Step Size

Ideally, the step size should be chooses such that the error is minimal. As the error term (5.1) contains second-order derivatives that are not available, an upper bound has to be drawn on the error that more or less independent of the derivatives:

$$\left|\frac{1}{2}\frac{\partial^2\varphi(\tilde{\boldsymbol{p}})}{\partial p_i^2}\delta_i + \frac{\varepsilon_{\delta_i} - \varepsilon}{\delta_i}\right| \le \frac{1}{2}\delta_i\left|\frac{\partial^2\varphi(\tilde{\boldsymbol{p}})}{\partial p_i^2}\right| + \frac{1}{\delta_i}\big|\varepsilon_{\delta_i} - \varepsilon\big| \le \frac{1}{2}\delta_i L_{\varphi'',i} + \frac{2}{\delta_i}\varepsilon_R L_\varphi$$

$$L_{\varphi'',i} := \max\left\{\left|\frac{\partial^2\varphi(\tilde{\boldsymbol{p}})}{\partial p_i^2}\right| : \tilde{\boldsymbol{p}} \in [\boldsymbol{p}, \boldsymbol{p} + \boldsymbol{e}_i\delta_i]\right\}$$

$$L_\varphi := \max\Big\{\big|\varphi(\boldsymbol{p})\big|, \big|\varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i)\big|\Big\}$$

Minimizing this w.r.r. to the step size yields:

$$\min_{\delta_i \in \mathbb{R}^+} \Phi(\delta_i), \quad \Phi(\delta_i) = \frac{1}{2}\delta_i L_{\varphi'',i} + \frac{2}{\delta_i}\varepsilon_R L_\varphi$$

$$\implies \quad \Phi'(\delta_i) = \frac{1}{2}L_{\varphi'',i} - \frac{2}{\delta_i^2}\varepsilon L_\varphi \overset{!}{=} 0 \quad \overset{L_{\varphi'',i} \neq 0}{\iff} \quad \delta_i = \sqrt{\frac{4\varepsilon_R L_\varphi}{L_{\varphi'',i}}}$$

If $L_\varphi / L_{\varphi'',i} \approx$, then $\delta_i \approx 2\sqrt{\varepsilon_R}$. If additionally it is possible to evaluate the function with machine precision, i.e. $\varepsilon_R = \varepsilon_{\text{mach}}$, the optimal step size is simply

$$\delta_i \approx 2\sqrt{\varepsilon_{\text{mach}}}$$

This step size often is a good choice and thus set as the default in most implementations. It also explains the rule of thumb that forward-differences can approximately evaluate half of decimal places correctly.

Notes

- For evaluating the gradient $\nabla\varphi(\boldsymbol{p})$ it is necessary to
  - Determine n_p step sized δ_i and to evaluate φ at least $2n_p$ times to approximate $L_{\varphi'',i}$.
  - Every iteration of a gradient-based method needs the gradients causing high computation times.
  - It is better to use a one-time approximation of *relative step sizes* ε_i with $\delta_i = \varepsilon_i\big(1 + |p_i|\big)$ at the initialization $\boldsymbol{p}^{(0)}$.
  - The value $\varepsilon_i = 5\sqrt{\varepsilon_{\text{mach}}}$ can be used as an initial relative step size.
  - If the optimization fails, restart with a new initialization and re-calculate the step sizes.

5.1.2 Central-Difference Approximation

For forward difference approximation often yields results that are good enough, except the gradients are too small. Additionally, the forward approximation is not sufficient if the optimization step size $\alpha^{(k)}$ such that the changes in $\boldsymbol{p}$ are less than the step size δ or the differences in the function values are "too small" relative to δ.

A potentially more exact approximation are *central differences*:

$$\frac{\partial\varphi(\boldsymbol{p})}{\partial p_i} \approx D_{Z,i}\varphi(\boldsymbol{p}) = \frac{1}{2\delta_i}\big(\varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) - \varphi(\boldsymbol{p} - \boldsymbol{e}_i\delta_i)\big)$$

Analogous to the forward differences, Taylor-expand this formula yields the insight that the order of the approximation error is $\mathcal{O}(\delta_i^2)$ while the order of the forward differences is $\mathcal{O}(\delta_i)$. Analogous to the forward differences, the optimal step size δ_i^Z can be calculated by minimizing an upper bound on the total error, yielding the optimal step size

$$\delta_i^Z = \sqrt[3]{\frac{3\varepsilon_R L_\varphi}{L_{\varphi''',i}}}, \quad L_{\varphi''',i} = \max\left\{ \left| \frac{\partial^3\varphi(\tilde{\boldsymbol{p}})}{\partial p_i^3} \right| : \tilde{\boldsymbol{p}} \in \left[\boldsymbol{p} - \boldsymbol{e}_i\delta_i^Z, \boldsymbol{p} + \boldsymbol{e}_i\delta_i^Z\right] \right\}$$

For functions with $L_\varphi / L_{\varphi''',i} \approx 1$ it follows $\delta_i^Z \approx \sqrt[3]{3\varepsilon_R} \approx \delta_i^{2/3}$. If additionally $L_\varphi \approx 1$ and $L_{\varphi''',i} \approx 1$, it follows:

$$\left| D_{Z,i}\hat{\varphi}(\boldsymbol{p}) - \frac{\partial\varphi(\boldsymbol{p})}{\partial p_i} \right| \leq \varepsilon_R^{2/3}$$

Hence, the rule of thumb that central difference can approximately evaluate two third of decimal places correctly.

But central differences produce much more computational overhead compared to forward/backward differences! Thus they should only be used if needed (by switching from forward to central differences).

To approximate the second derivative, the following schema can be used

$$\frac{\partial^2 \varphi(\boldsymbol{p})}{\partial p_i^2} \approx \frac{1}{\delta_i^2}\big(\varphi(\boldsymbol{p} + \boldsymbol{e}_i\delta_i) - 2\varphi(\boldsymbol{p}) + \varphi(\boldsymbol{p} - \boldsymbol{e}_i\delta_i)\big)$$

that is a combination of forward and backward differences to approximate the second-order derivative. This directly gives the i-th diagonal entry of the Hessian of φ, which can reduce the number of iterations needed.

5.2 Numerical Differentiation of Simulation Models

An important class in optimization is simulation-based optimization where the system state $\boldsymbol{x}(t)$ is given as the numerical solution of ODEs or PDEs. In this setting, the objective φ and the constraints $\boldsymbol{a}$, $\boldsymbol{b}$ are dependent on the state variables $\boldsymbol{x}$ of an ODE/PDE system. Hence, the calculation of $\varphi\big(\boldsymbol{x}(\boldsymbol{p})\big)$ requires solving the ODE/PDE numerically:

- Every calculation of $\varphi\big(\boldsymbol{x}(\boldsymbol{p})\big)$, $\boldsymbol{a}$ and $\boldsymbol{b}$ is computationally expensive.
- The calculation is only possible with simulation errors (i.e. approximation error in the ODE/PDE solver and accumulated rounding errors).
- The gradients (see below) are generally not available and have to me approximated.

$$\boldsymbol{\nabla}\varphi\big(\boldsymbol{x}(\boldsymbol{p})\big) = \frac{\partial \varphi\big(\boldsymbol{x}(\boldsymbol{p})\big)}{\partial \boldsymbol{p}} = \frac{\partial \varphi}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{p}}$$

5.2.1 Derivative of ODE-Simulation Models

Given an IVP (initial value problem) $\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}; \boldsymbol{p})$, $\boldsymbol{x}(0) = \boldsymbol{x}_0$, $0 \leq t \leq t_f$, the derivatives of the (numerical) solution $\boldsymbol{x}(t; \boldsymbol{p})$ w.r.t. to the parameters $\boldsymbol{p}$ are required. Formally, the parameters $\boldsymbol{p}$ can be transformed to initial values $\boldsymbol{x}_0$ and thus the derivatives w.r.t. the parameters to derivatives w.r.t. the initial values. The derivative w.r.t. the initial values is called the *sensitivity matrix*:

$$\frac{\partial \boldsymbol{x}(t; \boldsymbol{x}_0)}{\partial \boldsymbol{x}_0}$$

The IVP with the parameters $\boldsymbol{p}$ transformed to initial values is given as

$$\begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_{n_x} \end{bmatrix} = \dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}, x_{n_x+1}, \cdots, x_{n_x+n_p})$$

$$\begin{bmatrix} \dot{x}_{n_x+1} \\ \vdots \\ \dot{x}_{n_x+n_p} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

with $x_{n_x+1} \coloneqq p_1$, $\cdots$, $x_{n_x+n_p} \coloneqq p_{n_p}$ and the initial values

$$\boldsymbol{x}(0) = \boldsymbol{x}_0 \qquad \begin{bmatrix} x_{n_x+1}(0) \\ \vdots \\ x_{n_x+n_p}(0) \end{bmatrix} = \begin{bmatrix} p_1 \\ \vdots \\ p_{n_p} \end{bmatrix}$$

That is, the “parameter states” are added as time invariant states always equaling the parameters.

5.2.2 External Numerical Differentiation

Naïve Approach

Just use forward differences:

$$\frac{\partial \boldsymbol{x}(t;\boldsymbol{p})}{\partial p_i} \approx \frac{1}{\delta_i}\big(\boldsymbol{x}(t;\boldsymbol{p}+\boldsymbol{e}_i\delta_i) - \boldsymbol{x}(t;\boldsymbol{p})\big)$$

- This approach requires solving n_p additional IVP solutions for calculating the tweaked evaluations.
- To solve the IVP as best as possible, it may use variable step sizes.
- This causes gradient-based algorithms to have extremely big problems in finding the minimum!
- But this is not only caused by the optimization method…

Runge-Kutta Methods When using Runge-Kutta methods of order m, the numerical solutions depends on the integration tolerance and the internal step width of the integration. By just looking at the initial values, the error of RK methods of order m is

$$\tilde{\boldsymbol{x}}(t;\boldsymbol{x}_0,h_1) = \boldsymbol{x}(t;\boldsymbol{x}_0) + \sum_{j=m}^{N} \tilde{c}_j(t,\boldsymbol{x}_0)h_1^j + \mathcal{O}\big(h_1^{N+1}\big)$$

$$\tilde{\boldsymbol{x}}(t;\boldsymbol{x}_0+\boldsymbol{e}_i\delta_i,h_2) = \boldsymbol{x}(t;\boldsymbol{x}_0+\boldsymbol{e}_i\delta_i) + \sum_{j=m}^{N} \tilde{c}_j(t,\boldsymbol{x}_0+\boldsymbol{e}_i\delta_i)h_2^j + \mathcal{O}\big(h_2^{N+1}\big)$$

with differentiable functions $\tilde{c}_j(t,\boldsymbol{x}_0)$. Plugging this into the forward difference scheme yields:

$$\begin{aligned}
&\frac{1}{\delta_i}\big(\tilde{\boldsymbol{x}}(t;\boldsymbol{x}_0+\boldsymbol{e}_i\delta_i,h_2) - \tilde{\boldsymbol{x}}(t;\boldsymbol{x}_0,h_1)\big)\\
&= \frac{\partial \boldsymbol{x}(t;\boldsymbol{x}_0)}{\partial x_{0,i}} + \mathcal{O}(\delta_i) + \sum_{j=m}^{N} \tilde{c}_j(t,\boldsymbol{x}_0+\boldsymbol{e}_i\delta_i)\cdot \underbrace{\frac{h_2^j - h_1^j}{\delta_i}}_{\to\infty} + \sum_{j=m}^{N}\left(\frac{\partial \tilde{c}_j(t,\boldsymbol{x}_0)}{\partial x_{0,i}} + \mathcal{O}(\delta_i)\right)h_1^j + \mathcal{O}\left(\frac{h_1^{N+1}}{\delta_i}\right) + \mathcal{O}\left(\frac{h_2^{N+1}}{\delta_i}\right)
\end{aligned}$$

This is the problem! If the integration steps $h_1 \neq h_2$ are not equal, the error term becomes dominant as δ_i usually is “small”.

A “solution” would be to set $h_1 = h_2$, causing bad integration results.

Coupled Forward Differences Approximation

It is possible to simultaneously integrate an (n_p+1)-times big IVP for each tweaked value guaranteeing $h_1 = h_2$:

$$\begin{aligned}
\dot{\boldsymbol{x}} &= \boldsymbol{f}(t,\boldsymbol{x}), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0\\
\dot{\boldsymbol{x}} &= \boldsymbol{f}(t,\boldsymbol{x})\\
&\vdots\\
\dot{\boldsymbol{x}} &= \boldsymbol{f}(t,\boldsymbol{x})
\end{aligned}$$

5.2.3 Internal Numerical Differentiation

The external numerical differentiation costs a lot of time! One insight: The sensitivity matrix can be expresses as the solution of a matrix-ODE:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial \boldsymbol{x}(t;\boldsymbol{x}_0)}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{f}\big(t,\boldsymbol{x}(t;\boldsymbol{x}_0)\big)}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}(t;\boldsymbol{x}_0)}{\partial \boldsymbol{x}_0}, \qquad \frac{\partial \boldsymbol{x}(0;\boldsymbol{x}_0)}{\partial \boldsymbol{x}_0} = \boldsymbol{I}$$

- Variant 1: Simultaneously integrate the ODE and the matrix-ODE with a standard integrator.
- Variant 2 (better): Differentiated integrate method calculates $\boldsymbol{x}(t;\boldsymbol{p})$, $\frac{\partial \boldsymbol{x}(t;\boldsymbol{p})}{\partial \boldsymbol{p}}$, $\frac{\partial \dot{\boldsymbol{x}}(t;\boldsymbol{p})}{\partial \boldsymbol{p}}$
  - Advantage: Really efficient.
  - Disadvantage: Implementation complexity; especially complication for switching points.

5.3 Symbol Differentiation

If the functions φ, $\boldsymbol{a}$, $\boldsymbol{b}$ are given as explicit formulas, the derivatives $\boldsymbol{\nabla}\varphi(\boldsymbol{p})$, $\boldsymbol{J_a}(\boldsymbol{p})$, $\boldsymbol{J_b}(\boldsymbol{p})$ could be evaluated using a computer algebra system (e.g. Maple, Mathematica, SymPy, ...) in closed form. This is based on a systematic application of chain, product, ... rules and often requires a special input format.

- Advantage: No approximation error, only rounding errors.
- Disadvantages: Can lead to complex functions that are computationally expensive to evaluate.

5.4 Automatic Differentiation

Automatic differentiation refers to a technique to generate first and possible second-order derivatives of an existing program that calculates φ, $\boldsymbol{a}$, $\boldsymbol{b}$. This is based on the insight that every every so complex function can be composed of as a sequence of elementary functions with one or two arguments:

- 1-argument functions: Sine, Cosine, Exponential, Logarithm, ...
- 2-argument functions: Addition, Subtraction, Multiplication, Division, Exponentiation

AD-methods are based on an analysis of the evaluation sequence as a *computation graph* of elementary functions.

There are two main design decisions:

- Pre-compile a program to get the derivative function to be able to evaluation the function and the derivatives simultaneously (forward mode).
- Build the computation graph after the function has been evaluated and evaluate the derivative afterwards (backward mode).

The computational complexity of the gradient in forward mode on a scalar function with multiply variables $\varphi(\boldsymbol{p})$ can be as high as for symbolic differentiation: $\mathcal{O}(n_p$. But for the backward mode, a complexity of $\mathcal{O}(5)$ can be reached independent of n_p! But the memory complexity can rise a lot...

- Advantages:

  - No approximation error, only rounding errors.
  - AD is continuously improving and even today's algorithms are capable of a lot of calculations.

- Disadvantages:
  - It is problematic to handle piecewise functions (if-then-else), approximation of tabular data, approximations of Sine, Cosine, ... by rationale functions.
    But this is problematic even for analytical derivatives...

Even ODE- and PDE-simulations can be handled using AD!

Popular implementations, commonly used in machine learning, are libraries like TensorFlow (static computation graph) and PyTorch (dynamic computation graph).

6 Parameter Estimation

This chapter covers a special type of objective function: the sum of squares, called *least squares* optimization problems:

$$\varphi(\boldsymbol{p}) = \frac{1}{2}\sum_{i=1}^{n_r} r_i^2(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{p})\|_2^2$$

This objective function arises in a lot of optimization problems, e.g.: Given some detected corner points $(x_i, y_i)_{i=1,\cdots,n_r}$ of a ball, what is the radius R_K and the position (x_K, y_K) if the ball? The residual (error) function $\boldsymbol{r}$ can be determined from the circle equation:

$$R_K^2 = (x_i - x_K)^2 + (y_i - y_K)^2 \quad \Longrightarrow \quad r_i(x_K, y_K, R_K) = \sqrt{(x_i - x_K)^2 + (y_i - y_K)^2} - R_K$$

If, in practice, some parametric model is used, this almost always leads to a parameter fitting problem to measure/minimize the differences between the model and measurements. By minimizing the differences, the model with that most correspond to the measurements can be found (e.g. the parameters of a friction model or inertia of a robot).

6.1 Objective Functions

There are various different objective functions that could be used:

- Absolute sum: $\varphi_1(\boldsymbol{p}) = \|\boldsymbol{r}(\boldsymbol{p})\|_1 = \sum_{i=1}^{n_r} |r_i(\boldsymbol{p})|$
- Sum of squares: $\varphi_2(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{p})\|_2^2 = \frac{1}{2}\sum_{i=1}^{n_r} r_i^2(\boldsymbol{p})$
- Maximum difference: $\varphi_\infty(\boldsymbol{p}) = \|\boldsymbol{r}(\boldsymbol{p})\|_\infty = \max\left\{|r_i(\boldsymbol{p})| : i = 1, \cdots, n_r\right\}$

But even if $\boldsymbol{r}$ is differentiable, φ_1 and φ_∞ are generally not. But φ_∞ can be transformed into a NLP with differentiable functions:

$$\begin{aligned} &\min_{\boldsymbol{p}, p_{n_r+1}} \; p_{n_r+1} \\ &\text{subject to } -p_{n_r+1} \leq r_i(\boldsymbol{p}) \leq p_{n_r+1}, \quad i = 1, \cdots, n_r \end{aligned}$$

Anyway, φ_∞ is extremely sensitive towards outliers. On the other hand, φ_2 is less sensitive towards outliers and is differentiable (if $\boldsymbol{r}$ is differentiable)!

Hence, in most cases the least squares objective φ_2 is used. Besides the differentiability, it is also statistically appealing: If the measurement noise ε_{ij} are statistically independent and Gaussian distributed with mean 0 and variance σ^2, the solution of $\varphi_2 \to \min$ is a maximum likelihood estimator!

Additionally, by exploiting the structure of φ_2, the efficiency of gradient-based algorithms can be increased (e.g. due to sparse Jacobians).

6.2 Linear Least Squares

In the special case of a linear residual function $\boldsymbol{r}(\boldsymbol{p}) = \boldsymbol{J}^T\boldsymbol{p} + \boldsymbol{f}$ and the least squares objective

$$\varphi(\boldsymbol{p}) = \varphi_2(\boldsymbol{p}) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{p})\|_2^2$$

the optimization problem is quadratic in $\boldsymbol{p}$:

$$\varphi(\boldsymbol{p}) = \frac{1}{2}\left\|\boldsymbol{J}^T\boldsymbol{p} + \boldsymbol{f}_r\right\| = \frac{1}{2}\left(\boldsymbol{p}^T\boldsymbol{J} + \boldsymbol{f}_r^T\right)\left(\boldsymbol{J}^T\boldsymbol{p} + \boldsymbol{f}_r\right)$$

Zeroing the gradient $\boldsymbol{\nabla}\varphi(\boldsymbol{p}) \stackrel{!}{=} \boldsymbol{0}$ yields the *normal equations*

$$\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{p}^* = -\boldsymbol{J}\boldsymbol{f}_r$$

that a solution must fulfill (where $\boldsymbol{J}\boldsymbol{J}^T$ is symmetric and positive definite). This linear system should not be solved as a normal linear system as it is ill-conditioned! Better use special methods like orthogonalization or single value decomposition.

6.3 Optimality Conditions and Special Methods

Let $\varphi(\boldsymbol{p}) = \varphi_2(\boldsymbol{p})$ be two times continuously differentiable. Then the following first- and second-order necessary conditions can be formulated:

1. The gradient has to vanish:

$$\boldsymbol{\nabla}\varphi_2(\boldsymbol{p}^*) = \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{p}^*) \cdot \boldsymbol{r}(\boldsymbol{p}^*) = \boldsymbol{0}$$

2. The Hessian has to be positive semidefinite:

$$\boldsymbol{H}_{\varphi_2}(\boldsymbol{p}^*) = \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{p}^*)\,\boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{p}^*) + \sum_{i=1}^{n_r}\left(r_i(\boldsymbol{p}^*) \cdot \boldsymbol{H}_{r_i}(\boldsymbol{p}^*)\right) \tag{6.1}$$

6.3.1 Gauss-Quasi-Newton Method

In the Quasi-Newton method, the search direction is determined as a solution of the linear system

$$\boldsymbol{H}_{\varphi}\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{d}^{(k)} = -\boldsymbol{\nabla}\varphi\left(\boldsymbol{p}^{(k)}\right)$$

where the Hessian is approximated, e.g. using a BFGS update. But in the case of a least squares optimization problem, the only part of the Hessian (6.1) depending on second-order derivatives is:

$$\sum_{i=1}^{n_r}\left(r_i(\boldsymbol{p}^*) \cdot \boldsymbol{H}_{r_i}(\boldsymbol{p}^*)\right)$$

But as the objective gets smaller and smaller, this term also vanishes. Hence, the Hessian can be approximated using first-order derivatives only

$$\boldsymbol{H}_{\varphi_2}(\boldsymbol{p}^*) \approx \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{p}^*)\,\boldsymbol{J}_{\boldsymbol{r}}^T(\boldsymbol{p}^*)$$

if the residuals are “small”. This leads to the *Gauss-Newton Method* where the search direction is given as the solution of the normal equations

$$\boldsymbol{J}_r\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{J}_r^T\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{d}^{(k)} = -\boldsymbol{J}_r\left(\boldsymbol{p}^{(k)}\right)\cdot\boldsymbol{r}\left(\boldsymbol{p}^{(k)}\right)$$

or as the solution of a (better conditioned) linear least squares problem:

$$\min_{\boldsymbol{d}\in\mathbb{R}^{n_p}} \frac{1}{2}\left\|\boldsymbol{J}_r^T\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{d} + \boldsymbol{r}\left(\boldsymbol{p}^{(k)}\right)\right\|_2^2$$

If residuals are big or the problem is ill-conditioned, it can be modified with a suitable matrix $\boldsymbol{B}^{(k)}$:

$$\left(\boldsymbol{J}_{\boldsymbol{r}}\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{J}_{\boldsymbol{r}}^T\left(\boldsymbol{p}^{(k)}\right) + \boldsymbol{B}^{(k)}\right)\boldsymbol{d}^{(k)} = -\boldsymbol{J}_r\left(\boldsymbol{p}^{(k)}\right)\cdot\boldsymbol{r}\left(\boldsymbol{p}^{(k)}\right)$$

Additionally a step size rule should be used. This method is implemented, e.g. in *NLSCON* which also allows additional nonlinear inequality constraints.

6.3.2 Levenberg-Marquardt Methods

Instead of a Newton-approach, trust region methods can be used to determine the search direction. In the *Levenberg-Marquardt Method*, the search direction $\boldsymbol{d}^{(k)}$ is given as the solution of:

$$\left(\boldsymbol{J}_{\boldsymbol{r}}\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{J}_{\boldsymbol{r}}^T\left(\boldsymbol{p}^{(k)}\right) + \gamma^{(k)}\boldsymbol{I}\right)\boldsymbol{d}^{(k)} = -\boldsymbol{J}_{\boldsymbol{r}}\left(\boldsymbol{p}^{(k)}\right)\cdot\boldsymbol{r}\left(\boldsymbol{p}^{(k)}\right), \quad \gamma^{(k)} \geq 0$$

That is, the search direction is a mixture of the Gauss-Newton direction and steepest descent. The same search direction can be received by solving a constrained linear least squares problem:

$$\begin{aligned}&\min_{\boldsymbol{d}\in\mathbb{R}^{n_p}} \frac{1}{2}\left\|\boldsymbol{J}_{\boldsymbol{r}}^T\left(\boldsymbol{p}^{(k)}\right)\boldsymbol{d} + \boldsymbol{r}\left(\boldsymbol{p}^{(k)}\right)\right\|_2^2\\ &\text{subject to } \|\boldsymbol{d}\|_2 \leq \delta^{(k)}\end{aligned}$$

Where δ and γ have a connection.

This is implemented, e.g. in *LMDER*, *LMJAC* of *MINPACK*.

6.3.3 Notes

- The Gauss-Newton method needs an appropriate test function that also exploits the structure of the objective function to determine the step size to ensure global convergence.
- In the Levenberg-Marquardt method, globalization happens by choosing an appropriate trust region.
- Levenberg-Marquardt and Gauss-Newton methods are in general a lot more efficient (faster and more precise) than solving least squares problems using general purpose optimization techniques (e.g. Quasi-Newton or SQP).
- When optimization by simulation, the sensitivity matrix $\frac{\partial \boldsymbol{x}(t;\boldsymbol{p})}{\partial \boldsymbol{p}}$ is needed (seesubsection 5.2.1).

6.4 Conditioning of Normal Equations

Often, the Jacobian of the objective function of linear least squares $\boldsymbol{J}$ is ill-conditioned, i.e. $\operatorname{cond} \boldsymbol{J}$ is high, causing round error to be increased. In extreme cases, $\boldsymbol{J}$ does not has full rank, i.e. $\operatorname{cond} \boldsymbol{J} = \infty$. In this case, the solution $\boldsymbol{d}$ of the normal equations

$$\boldsymbol{J}\boldsymbol{J}^T\boldsymbol{d} = -\boldsymbol{J}\boldsymbol{r}$$

is not unique! Common reasons for ill-conditioning are:

- "too few" measurements
  E.g. the measurements are not "dense enough".
- not the "correct" measurements
  E.g. the measurements to not depend or weakly depend on the optimization variables.
- System model does not fit to the measurements (it is incompatible). Then either the measurements or the model is wrong.

6.5 Result Interpretation

6.5.1 Common Problems

Common problems if the residuals are still high after the optimization terminates:

- Derivatives are not precise enough.
- The optimization method is not suitable for the problem, e.g. if the method cannot handle inexact function evaluations or has problems with local minima of φ_2.
- In parameter estimation settings,
  - the system model and measurements might be incompatible (wrong measurements) or
  - the system model and the physical might be incompatible (wrong model).

Common problems for small residuals:

- Some optimization variables might not be uniquely determined.
- Too less measurements, variables are not unique in general (e.g. linearly dependent in the model).

Practical aspects:

- Scaling/balancing of the variables: By the transformation $p_i \to \hat{p}_i = s_i p_i$, $s_i = \text{const} > 0$, the derivative changes to $\frac{\partial \varphi}{\partial \hat{p}_i} = \frac{\partial \varphi}{\partial p_i} \frac{1}{s_i}$.
- Scaling of the residuals by weights $w_i > 0$: $\varphi(\boldsymbol{p}) = \frac{1}{2} \sum_{i=1}^{n_r} w_i r_i^2(\boldsymbol{p})$

6.5.2 The Covariance Matrix

Assuming the measurement errors ε_i (in the measurements r_i) are normally distributed with zero mean and constant variance σ^2. Then the solution of the linear least squares problem is a maximum likelihood estimator for the parameters! The covariance matrix $\boldsymbol{V}$ is then given by $\boldsymbol{V} = \sigma^2 \left(\boldsymbol{J}\boldsymbol{J}^T\right)^{-1}$ where the variance can be approximated by

$$\sigma^2 \approx \frac{\|\boldsymbol{r}(\boldsymbol{p}^*)\|^2}{n_r - n_p}$$

and the mean of the residual squares is given as

$$\mathbb{E}\left[\|\boldsymbol{r}(\boldsymbol{p}^*)\|^2\right] = (n_r - n_p)\sigma^2$$

Hence, a bad conditioning of $\boldsymbol{J}$ implies high variance!

6.6 Optimal Experimental Design

Goal of *optimal experimental design* is a good conditioning of the parameter identification problem, i.e. the optimal experimental parameters $\boldsymbol{s}^*$ is a solution of the optimization problem

$$\min_{\boldsymbol{s}} \phi_{\text{exp}}\left(\boldsymbol{V}(\boldsymbol{s})\right)$$

where $\boldsymbol{V}$ is the covariance matrix. Some objective ϕ_{exp} are:

1. Determinant of $\boldsymbol{V}$.
2. Trace mean, i.e. $\operatorname{tr} \boldsymbol{V}/n_p$
3. Biggest eigenvalue of $\boldsymbol{V}$
4. Absolute length of the biggest confidence interval.
5. Conditional number.

6.7 Examples

6.7.1 Parameter-Dependent Vehicle Dynamics

Simulated Measurements

Real Measurements

Comparison

6.7.2 Parameter Estimation for “BioBiped”

7 Minimization of Functionals

In the setting of *variational problems*, the unknown $\boldsymbol{x}$ is a function of t (where t is the independent variable) and the objective is a functional $J[\boldsymbol{x}]$ of integral type:

$$J[\boldsymbol{x}] = \int L\big(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\big)\,\mathrm{d}t$$

the solution $\boldsymbol{x}^*$ must fulfill given constraints, e.g.:

- Initial and end conditions (boundary conditions)
- Inequality constraints
- Integral-type constraints
- Differential equations

If differential equations are given, the problem is called an *optimal control problem*.

7.1 Euler-Lagrange Equation

Given an optimization problem

$$\begin{aligned} \min_{\boldsymbol{x}} J[\boldsymbol{x}], \quad J[\boldsymbol{x}] &= \int_a^b L\big(\boldsymbol{x}(t), \dot{\boldsymbol{x}}, t\big)\,\mathrm{d}t \\ \text{subject to } & \begin{aligned} \boldsymbol{x}(a) &= \boldsymbol{x}_a \\ \boldsymbol{x}(b) &= \boldsymbol{x}_b \end{aligned} \end{aligned}$$

where $\boldsymbol{x} : \mathbb{R} \to \mathbb{R}^{n_x}$, $L : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times R \to R$ are two times continuously differentiable.

A *stationary* solution $\boldsymbol{x}^*$, i.e. a solution that is a minimum candidate, has to fulfill the *Euler-Lagrange Equation*:

$$\frac{\partial L}{\partial x_i} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{x}_i} = 0, \quad i = 1, \cdots, n_x$$

Additionally, the boundary conditions $\boldsymbol{x}(a) = \boldsymbol{x}_a$, $\boldsymbol{x}(b) = \boldsymbol{x}_b$ must be fulfilled, yielding a second-order ordinary boundary value problem.

7.1.1 Example

As of the *Hamilton's principle* $\boldsymbol{x}(t)$ between t_1 and t_2 is a stationary point of the action functional

$$\int_{t_1}^{t_2} L\big(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\big)\,\mathrm{d}t$$

where L is the Lagrangian of the system, i.e. the difference of kinetic and potential energy:

$$L = T - V$$

For a ball with mass m and gravity acceleration g that is thrown into the air in a straight line (i.e. one-dimensional), the kinetic and potential energy are given as:

$$T = \frac{m}{2}\dot{x}^2 \qquad V = mgx$$

The Lagrangian then is $L = T - V = \frac{m}{2}\dot{x}^2 - mgx$. Plugging that in the Euler-Lagrange equations yields a second-order differential equation for the movement of the ball:

$$\frac{\partial L}{\partial x} = -mg \qquad \frac{\partial L}{\partial \dot{x}} = mx \qquad \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{x}} = m\ddot{x} \quad \Longrightarrow \quad -mg - m\ddot{x} = 0 \quad \Longleftrightarrow \quad \ddot{x} = -g$$

Solving this differential equation with the initial values $x(0) = 0$, $\dot{x}(0) = \dot{x}_0$ yields the equation of movement of the ball:

$$x(t) = -\frac{g}{2}t^2 + \dot{x}_0 t$$

As expected, this equation is a perfect parabola w.r.t. time! The same can be done in two dimensions.

7.1.2 Notes

- The Euler-Lagrange equations are in general not tractable. Hence, numerical methods have to be used.
- The equations can be extended to solutions with non-differentiabilities, (in-) equality constraints, integral-type constraints, ...
- Also, second-order necessary conditions can be formulated.

7.1.3 Derivation

8 Optimal Control

A dynamical system is described by the ODE

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}, \boldsymbol{z}), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0$$

where $\boldsymbol{x}$ is the state, $\boldsymbol{u}$ are the control variables, $\boldsymbol{p}$ are (constant) system parameters and $\boldsymbol{z}$ is noise. An optimal control problem has given $\boldsymbol{x}_0$, $\boldsymbol{p}$ and $\boldsymbol{x}_i(t_f)$ and seeks for an optimal $\boldsymbol{u}$ and $\boldsymbol{x}$.

A complete optimal control problem is given as:

$$\begin{aligned}
\min J[\boldsymbol{u}], \quad J[\boldsymbol{u}] = \phi\big(\boldsymbol{x}(t_f), t_f\big) + \int_0^{t_f} L\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)\,\mathrm{d}t \\
\text{subject to} \qquad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) \\
x_i(0) = x_{i,0} = \text{const}, \quad i \in \{1, \cdots, n_x\} \\
\boldsymbol{r}\big(\boldsymbol{x}(t_f), t_f\big) = \boldsymbol{0}, \quad \text{e.g. } x_j(t_f) = x_{j,f},\ j \in \{1, \cdots, n_x\} \\
\boldsymbol{g}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) \geq \boldsymbol{0} \\
\boldsymbol{g}\big(\boldsymbol{x}(t)\big) \geq \boldsymbol{0} \\
\boldsymbol{r}^i\big(\boldsymbol{x}(t_s - 0), \boldsymbol{x}(t_s + 0), t_s\big) = \boldsymbol{0}
\end{aligned}$$

The constraints (from to bottom) are called:

- Equations of movement, these are the defining property of an optimal control problem over a regular variational problem.
- Initial conditions (optional).
- Final conditions (optional).
- Control constraints (optional, often box constraints).
- State constraints (optional).
- Interior point constraints (optional).

Additionally, the final time t_f might either fixed or free. The objective (called the *Bolza functional*) is split into an endpoint cost (also called *Mayer term*) and a Lagrangian:

$$J[\boldsymbol{u}] = \underbrace{\phi\big(\boldsymbol{x}(t_f), t_f\big)}_{\text{Endpoint Cost}} + \int_0^{t_f} \underbrace{L\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)}_{\text{Lagrangian}}\,\mathrm{d}t$$

Note that the endpoint cost is more general than the Lagrangian in terms that the Lagrangian term can be transformed to an endpoint cost by introducing a new state

$$\dot{x}_{n_x+1} = L\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big), \quad x_{n_x+1}(0) = 0$$

and changing the endpoint cost to:

$$\tilde{\phi}\big(\tilde{\boldsymbol{x}}(t_f), t_f\big) \coloneqq \phi\big(\boldsymbol{x}(t_f), t_f\big) + x_{n_x+1}(t_f)$$

Additionally, non-autonomous problems can be transformed to autonomous problems by introducing a "clock state" $\dot{x}_{n_x+1} = 1$, $x_{n_x+1}(0) = 0$ and changing the ODE accordingly:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(t, \boldsymbol{x}(t), \boldsymbol{u}(t)\big) \quad \rightarrow \quad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(x_{n_x+1}, \boldsymbol{x}(t), \boldsymbol{u}(t)\big)$$

8.1 Necessary Optimality Conditions for the Basis Problem

The basis optimal control problem does not have control, state or interior constraints:

$$\begin{aligned} \min J[\boldsymbol{u}], \quad J[\boldsymbol{u}] &= \phi\big(\boldsymbol{x}(t_f), t_f\big) + \int_0^{t_f} L\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)\,\mathrm{d}t \\ \text{subject to} \qquad \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) \\ x_i(0) &= x_{i,0} = \text{const}, \quad i \in \{1, \cdots, n_x\} \\ \boldsymbol{r}\big(\boldsymbol{x}(t_f), t_f\big) &= \boldsymbol{0}, \quad \text{e.g. } x_j(t_f) = x_{j,f},\ j \in \{1, \cdots, n_x\} \end{aligned}$$

For formulating the necessary optimality conditions, the following auxiliary functions are needed (the latter one is called *Hamiltonian*):

$$\begin{aligned} \Phi(\boldsymbol{x}, t, \boldsymbol{\nu}) &\coloneqq \phi(\boldsymbol{x}, t) + \boldsymbol{\nu}^T \boldsymbol{r}(\boldsymbol{x}, t) \\ H(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}) &\coloneqq L(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \end{aligned}$$

Note that, while kept implicitly, the auxiliary variables $\boldsymbol{\nu}$ and $\boldsymbol{\lambda}$ are time-dependent! The $\boldsymbol{\lambda}$ are also called the *adjunct variables* of the optimal control problem.

8.1.1 Boundary Conditions

To solve the optimal control problem, $2n_x$ boundary conditions are needed (the optimality conditions will yield one first-order ODE for every state and adjunct variable). Additional to the given boundary conditions of the problem formulation, enough conditions have to be found to get $2n_x$ conditions. Common cases:

(i) Fixed initial conditions $x_i(0) = x_{i,0} = \text{const}$:
Either $x_k(0)$ is given or, if not, set $\lambda_i(0) = -\frac{\partial \phi(\boldsymbol{x}(0), \boldsymbol{x}(t_f), t_f)}{\partial x_i(0)}$

(ii) Fixed final conditions $x_i(t_f) = x_{i,f} = \text{const}$:
Either $x_i(t_f)$ is given or, if not and x_i is not part of $\boldsymbol{r}(\cdots)$, set $\lambda_i(t_f) = \frac{\partial \phi(\boldsymbol{x}(t_f), t_f)}{\partial x_i(t_f)}$

(iii) Mixed boundary conditions:

a) General boundary conditions $\boldsymbol{r}\big(\boldsymbol{x}(0), \boldsymbol{x}(t_f), t_f\big) = \boldsymbol{0}$:
If $x_i(0)$ is free, set $\lambda_i(0) + \frac{\partial \Phi}{\partial x_i(0)}\Big|_{t=0} = 0$
If $x_i(t_f)$ is free, set $\lambda_i(t_f) - \frac{\partial \Phi}{\partial x_i(t_f)}\Big|_{t=t_f} = 0$

b) Periodic boundary conditions $x_i(0) - x_j(t_f) = 0$:
Set $\lambda_i(0) - \lambda_j(t_f) + \frac{\partial \phi}{\partial x_i(0)} + \frac{\partial \phi}{\partial x_j(t_f)} = 0$

If the final time t_f is free, an additional condition has to be employed:

$$H\big(\boldsymbol{x}(t_f), \boldsymbol{u}(t_f), \boldsymbol{\lambda}(t_f)\big) = -\frac{\partial \Phi}{\partial t_f}$$

8.1.2 First-Order Necessary Optimality Conditions (Maximum Principle)

Functions $\boldsymbol{x}^*$, $\boldsymbol{u}^*$ and $\boldsymbol{\lambda}^* \neq \mathbf{0}$ are the optimal solution of the basis problem iff the *canonical differential equations*

$$\dot{\boldsymbol{x}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} \qquad \dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{x}}$$

and the boundary conditions are fulfilled and the optimal control $\boldsymbol{u}^*$ minimizes the Hamilton function:

$$H\big(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t)\big) = \min_{\tilde{\boldsymbol{u}} \in U} H\big(\boldsymbol{x}^*(t), \tilde{\boldsymbol{u}}, \boldsymbol{\lambda}^*(t)\big), \quad \forall t \in [0, t_f] \tag{8.1}$$

Where $U \subseteq \mathbb{R}^{n_u}$ is the set of feasible controls. These conditions are called the *maximum principle*.

Definition: The Hamiltonian is called *regular* if and only if it has an unique minimum.

If the Hamiltonian is regular and the control $\boldsymbol{u}$ appears nonlinear in the Hamiltonian, condition (8.1) can be formulated as

$$\frac{\partial H}{\partial \boldsymbol{u}} = \mathbf{0}$$

Derivation

8.1.3 Second-Order Necessary Optimality Condition (Legendre-Clebsch Condition)

For stationary $\boldsymbol{x}^*$, $\boldsymbol{u}^*$, $\boldsymbol{\lambda}^*$, the second-order necessary optimality condition, also called the *Legendre-Clebsch Condition* is that the Hessian of the Hamiltonian is positive semidefinite along the optimal state and control:

$$\boldsymbol{H}_H^{\boldsymbol{p}}\big(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t)\big) \geq 0, \quad \forall t \in [0, t_f]$$

8.1.4 Example

8.1.5 Application: Optimal Robot Control

The Robot Dynamics

Optimal Control

Objective Functionals

Necessary Conditions

8.2 Bang-Bang Singular Control

This chapter covers optimal control functions where the control function appears only linear in the Lagrangian and the motion equations:

$$L\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = L_0\big(\boldsymbol{x}(t)\big) + \boldsymbol{L}_1^T\big(\boldsymbol{x}(t)\big)\boldsymbol{u}(t), \quad L_0 : \mathbb{R}^{n_x} \to R,\ \boldsymbol{L}_1 : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$$
$$\boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = \boldsymbol{f}_0\big(\boldsymbol{x}(t)\big) + \boldsymbol{f}_1^T\big(\boldsymbol{x}(t)\big)\boldsymbol{u}, \quad \boldsymbol{f}_0 : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x},\ \boldsymbol{f}_1 : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u \times n_x}$$

Also, it is assumed to have box constraints for the controls: $\boldsymbol{u}_{\min} \leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{\max}$.

The according Hamiltonian is

$$\begin{aligned} H(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}) &= L_0\big(\boldsymbol{x}(t)\big) + \boldsymbol{L}_1^T(\boldsymbol{x}(t))\boldsymbol{u}(t) + \boldsymbol{\lambda}^T\big(\boldsymbol{f}_0\big(\boldsymbol{x}(t)\big) + \boldsymbol{f}_1^T\big(\boldsymbol{x}(t)\big)\boldsymbol{u}\big) \\ &= \underbrace{\big(\boldsymbol{L}_1^T(\boldsymbol{x}(t)) + \boldsymbol{\lambda}^T\boldsymbol{f}_1^T\big(\boldsymbol{x}(t)\big)\big)}_{\boldsymbol{s}(\boldsymbol{x},\boldsymbol{\lambda}) \coloneqq} \boldsymbol{u}(t) + L_0\big(\boldsymbol{x}(t)\big) + \boldsymbol{\lambda}^T\boldsymbol{f}_0\big(\boldsymbol{x}(t)\big) \end{aligned}$$

where $\boldsymbol{s}(\boldsymbol{x}, \boldsymbol{\lambda})$ is called the *switching function*. As of the maximum principle, the control $\boldsymbol{u}$ is optimal iff it minimizes the Hamiltonian. Hence, the optimal control function $\boldsymbol{u}$ is given as a piecewise function depending on the switching function:

$$u_i(t) = \begin{cases} u_{i,\min} & \text{iff } s_i\big(\boldsymbol{x}(t), \boldsymbol{\lambda}(t)\big) > 0 \\ u_{i,\max} & \text{iff } s_i\big(\boldsymbol{x}(t), \boldsymbol{\lambda}(t)\big) < 0 \\ u_{i,\text{sing}} & \text{iff } s_i\big(\boldsymbol{x}(t), \boldsymbol{\lambda}(t)\big) \equiv 0 \end{cases}$$

Here, $u_{i,\text{sing}}$ is the *singular control* that is only needed if the switching function is zero over an interval. The other two cases are called *bang-bang control*.

All of the following will only cover a single control $u_i(t)$, but the methods can be applied to more than one control variable in a setting. Of course it is possible that some control variables are nonlinear in the Lagrangian, hence a bang-bang or singular control is not needed in that case.

8.2.1 Singular Control

Let t_1 and t_2, $t_1 < t_2$ denote the start and end point of a singular control interval, respectively, i.e. the switching function is zero in that interval:

$$s_i\big(\boldsymbol{x}, \boldsymbol{\lambda}\big) = 0, \quad \forall t \in [t_1, t_2]$$

With the insight that a system has to keep itself in the singular control, also the time derivative $s_i^{(1)} \coloneqq \frac{\mathrm{d}s_i}{\mathrm{d}t}$ of the switching function has to vanish. And also the second-order time derivative $s_i^{(2)}$. In fact, every time derivative has to vanish, yielding a recursive process for calculating the m-th time derivative:

$$s_i^{(m)} = \frac{\mathrm{d}}{\mathrm{d}t} s_i^{(m-1)}\big(\boldsymbol{x}(t), \boldsymbol{\lambda}(t)\big) \equiv 0$$

Let m be the smallest number of which the control u_i appears explicitly in the time derivative, $\frac{\partial s_i^{(m)}}{\partial u_1} \neq 0$. This yields a formula to determine $u_{i,\text{sing}}$:

$$s_i^{(m)}\big(\boldsymbol{x}(t), \boldsymbol{\lambda}(t), \boldsymbol{u}(t)\big) \equiv 0$$

Where u_i is part of the vector $\boldsymbol{u}$.

Second-Order Necessary Optimality Condition for Singular Control

Theorem: If $m < \infty$, then m is even. Let $m = 2p$, where p is called the *order* of the singular control.

The second-order necessary optimality condition for singular control, i.e. the generalized Legendre-Clebsch is:

$$(-1)^p \frac{\partial}{\partial u_i} s_i^{(m)}\big(\boldsymbol{x}^*(t), \boldsymbol{\lambda}^*(t), \boldsymbol{u}^*(t)\big) \geq 0, \quad \forall t \in [t_1, t_2]$$

The condition can be expanded to a sufficient condition by replacing the inequality with a strict inequality, i.e. the left side has to be positive.

If the sufficient condition is fulfilled, two conclusions can be proven:

- If p is odd, u_i is either discontinuous or continuously differentiable in t_1.
- If p is even, u_i is continuous in t_1, but *chattering* is possible. If *chattering* occurs, the control u_i "rings" around zero until reaching t_1, i.e. s_i has infinitely many root before entering the singular section.

8.2.2 Application: Time-Minimal Robot Control

8.2.3 Notes

- If all degrees of freedom would become singular simultaneously, all adjunct variables would be zero (contradicting the maximum principle). Hence, this is not possible! For bang-bang control this means that at least one control input has to operate at its maximum or minimum.
- With the given properties, a numerically calculated solution can be checked for sanity (e.g. if all controls are singular on a section, the solution cannot be optimal).
- There are approaches to eliminate singular control beforehand so that only the number and order of switching points has to be calculated. But often a control problem can be found that necessarily has singular parts, so such methods cannot determine the optimal solution.
- The necessary conditions lead to a fully determined boundary value problem (BVP) for $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ that can be solved using numerical methods (see section 9.2), but the order of bang-bang and singular control has to be known.

8.3 Value Function and Hamilton-Jacobi-Bellman Equation

There is a useful interpretation of the adjunct variables $\boldsymbol{\lambda}$! Note that by varying the initial conditions t_0 and $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$, different trajectories $\boldsymbol{x}^*$, $\boldsymbol{u}^*$, $\boldsymbol{\lambda}^*$ are generated by solving the optimal control problem. The *Value Function* expresses the value (of the objective) for a given initial condition $(t_0, \boldsymbol{x}_0)$:

$$\begin{aligned} V(t_0, \boldsymbol{x}_0) &= \min_{\boldsymbol{u} \in U} \left(\phi\big(\boldsymbol{x}(t_f), t_f\big) + \int_{t_0}^{t_f} L\big(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau)\big)\, \mathrm{d}\tau \right) \\ V(t_f, \boldsymbol{x}) &= \phi(\boldsymbol{x}, t_f) \end{aligned}$$

The *Hamilton-Jacobi-Bellman Equation* is a partial differential equation for the value function:

$$\begin{aligned} -\frac{\partial V(t, \boldsymbol{x})}{\partial t} &= \min_{\tilde{\boldsymbol{u}}} \left(L(t, \boldsymbol{x}, \tilde{\boldsymbol{u}}) + \left(\frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right)^T \boldsymbol{f}(t, \boldsymbol{x}, \tilde{\boldsymbol{u}}) \right) \\ V(t_f, \boldsymbol{x}) &= \phi(\boldsymbol{x}, t_f) \end{aligned}$$

If $V(t, \boldsymbol{x})$ is a solution of the Hamilton-Jacobi-Bellman equation, then the minimization

$$\min_{\tilde{\boldsymbol{u}}} \left(L(t, \boldsymbol{x}, \tilde{\boldsymbol{u}}) + \left(\frac{\partial V(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \right)^T \boldsymbol{f}(t, \boldsymbol{x}, \tilde{\boldsymbol{u}}) \right)$$

generates the optimal control $\boldsymbol{u}^*$ of the basis problem with $t_0 \leq t \leq t_f$ with the adjunct variables:

$$\boldsymbol{\lambda}^*(t) = \frac{\partial V\big(t, \boldsymbol{x}^*(t)\big)}{\partial \boldsymbol{x}}$$

Hence, the adjunct variables are the gradient of the minimal objective value w.r.t. the state.

8.3.1 Derivation

8.3.2 Notes

- If $\lambda_i^*(t) \equiv 0$ for $t \in [t_1, t_2]$, the value of the objective does not depend on $x_i^*(t)$ for the same interval.
- The value function contains all information needed for the optimal feedback control $\boldsymbol{u}^*(t, \boldsymbol{x})$. If the value function would be available, the optimal control could be calculate for arbitrary initial conditions. But most of the time it is not available…
- It is nearly impossible to solve the HJB equation numerically (!) for relevant problems, even without constraints.

8.4 Constraints

It is possible to add state and control constraints to the optimal control problem in the form

$$\boldsymbol{g}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) \geq \boldsymbol{0}$$

where the inequality is element-wise. Such a constraint is called a *control constraint* if $\boldsymbol{u}$ appears explicitly in $\boldsymbol{g}$, i.e. $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} \neq \boldsymbol{0}$.

8.4.1 Mixed Inequality Constraints

Mixed inequality constraints are both dependent on the state and the control:

$$\boldsymbol{g}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) \geq \boldsymbol{0}, \quad \boldsymbol{g} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_g}, \quad t \in [0, t_f]$$

This constraint is added to the Hamiltonian with multipliers $\boldsymbol{\eta}$ yielding the augmented Hamiltonian, similar to the Lagrangian of static optimization:

$$H(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}, \boldsymbol{\eta}) = L(\boldsymbol{x}, \boldsymbol{u}) + \lambda^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) + \boldsymbol{\eta}^T \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{u})$$

Analogous to the normal Hamiltonian, the augmented Hamiltonian is called *regular* along the optimal solution $\boldsymbol{x}^*(t)$, $\boldsymbol{\lambda}^*(t)$, $\boldsymbol{\eta}^*(t)$ iff $H\big(\boldsymbol{x}^*(t), \boldsymbol{u}, \boldsymbol{\lambda}^*(t), \boldsymbol{\eta}^(t)\big)$ has a unique minimum $\boldsymbol{u} = \boldsymbol{u}^*(t)$ for all $t \in [0, t_f]$.

Necessary Conditions (Maximum Principle)

Let the problem be autonomous, $\boldsymbol{f}$, $\boldsymbol{g}$, ϕ be continuously differentiable, U the set of feasible optimal controls and let the inequality constraint $\boldsymbol{g} \geq \boldsymbol{0}$ have an active component g_i with $g_i(t) = 0$ for $t \in [t_1, t_2]$, i.e. let it be active on $[t_1, t_2]$.

If not all boundary conditions are given, they have to be determined analogous tosubsection 8.1.1.

Then the necessary optimality conditions can be defined analogous to the ones of the basis problem: Iff $\boldsymbol{x}^*$, $\boldsymbol{u}^*$, $\boldsymbol{\lambda}^*$, $\boldsymbol{\eta}^*$ are optimal, the canonical differential equations

$$\dot{\boldsymbol{x}} = \frac{\partial H}{\partial \boldsymbol{\lambda}} \qquad \dot{\boldsymbol{\lambda}} = -\frac{\partial H}{\partial \boldsymbol{x}}$$

are fulfilled and

$$\eta_j \begin{cases} = 0 & \text{iff } g_j\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) > 0 \\ \leq 0 & \text{iff } g_j\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = 0 \end{cases}$$

which is, for the special constraint g_i, equivalent to:

$$\eta_j \begin{cases} = 0 & t \in [0, t_1) \cup (t_2, t_f] \\ \leq 0 & t \in [t_1, t_2] \end{cases}$$

Also, the following condition have to be fulfilled on the switching points of the constraints:

$$\lim_{\delta \to 0^+} \boldsymbol{\lambda}(t_\Delta + \delta) = \left(\lim_{\delta \to 0^-} \boldsymbol{\lambda}(t_\Delta + \delta) \right) - \nu_\Delta \cdot \frac{\partial g_i\big(\boldsymbol{x}(t_\Delta), \boldsymbol{u}(t_\Delta)\big)}{\partial \boldsymbol{x}}$$

Where $\eta_\Delta = \text{const} \leq 0$ and t_Δ is the entry/exit point, i.e. $\Delta = 1$ or $\Delta = 2$. This implies that the adjunct variables are in general not continuous at the switching points.

The optimal control than has to minimize the augmented Hamiltonian over U:

$$H\big(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), \boldsymbol{\eta}^*(t)\big) = \min_{\tilde{\boldsymbol{u}} \in U} H\big(\boldsymbol{x}^*(t), \tilde{\boldsymbol{u}}, \boldsymbol{\lambda}^*(t), \boldsymbol{\eta}^*(t)\big), \quad \forall t \in [0, t_f] \tag{8.2}$$

Along the active constraint,

$$g_i\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = 0 \quad \text{and} \quad \frac{\partial g_i(\boldsymbol{x}, \boldsymbol{u})}{\partial u_j} \neq 0$$

yields a condition for determining u_j along the active constraint. All other controls as well as u_j outside of the active section have to fulfill (8.2) and the Legendre-Clebsch condition (iff H is regular) or the corresponding bang-bang/singular control, respectively.

As the problem is autonomous, the Hamiltonian is constant sectionally constant w.r.t. time t and it can change when constraints become active/inactive.

8.4.2 State Inequality Constraints

Simple state inequality constraints

$$\boldsymbol{g}\big(\boldsymbol{x}(t)\big) \geq \boldsymbol{0}$$

that does not contain $\boldsymbol{u}$, but at least one x_i are handled like a mixture of mixed constraints and singular control.

Let a single constraint g_i be active in the interval $[t_1, t_2]$, i.e. $g_i\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = 0$ for $t \in [t_1, t_2]$. Then also the time derivative has to vanish as well as the second-order time derivative[1]:

$$g_i^{(m_g)} = \frac{\mathrm{d}}{\mathrm{d}t} g_i^{(m_g - 1)}\big(\boldsymbol{x}(t)\big) \equiv 0$$

[1] Let $g_i^{(0)} := g_i$.

Let m_g be the smallest number such that $g_i^{(m_g)}$ explicitly contains the control u_j, then

$$g^{(m_g)}(\boldsymbol{x}, \boldsymbol{u}) = 0$$

yields an equation for determining u_j in the interval $[t_1, t_2]$ and m_g is called the *order* of the state constraint. Hence, a single active constraints determined a single control variable!

Augmented Hamiltonian

Let $\boldsymbol{g}(\boldsymbol{x}) = g(\boldsymbol{x})$, i.e. only a single state constraint.

Mathematically, the active constraint $g\big(\boldsymbol{x}(t)\big) = 0$ for $t \in [t_1, t_2]$ is equivalent to

$$g^{(k)}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = 0, \quad t \in [t_1, t_2]$$

for every $k = 1, \cdots, m_g$. Hence, there are multiple, equivalent formulations of the maximum principle for active state constraints by viewing at the corresponding augmented Hamiltonian, e.g.:

$$\begin{aligned}
H^0(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}^0, \eta^0) &= L(\boldsymbol{x}, \boldsymbol{u}) + \big(\boldsymbol{\lambda}^0\big)^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) + \eta^0 g^{(0)}(\boldsymbol{x}) \\
&\vdots \\
H^k(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}^k, \eta^k) &= L(\boldsymbol{x}, \boldsymbol{u}) + \big(\boldsymbol{\lambda}^k\big)^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) + \eta^k g^{(k)}(\boldsymbol{x}) \\
&\vdots \\
H^k(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}^{m_g}, \eta^{m_g}) &= L(\boldsymbol{x}, \boldsymbol{u}) + \big(\boldsymbol{\lambda}^{m_g}\big)^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) + \eta^{m_g} g^{(m_g)}(\boldsymbol{x})
\end{aligned}$$

Maximum Principle

As in the previous section, a single state constraint g is assumed.

If not all boundary conditions are given, they have to be determined analogous tosubsection 8.1.1.

The maximum principle for state constraints is similar to the one for mixed constraints. Let g be active in the interval $[t_1, t_2]$. Iff $\boldsymbol{x}^*$, $\boldsymbol{u}^*$, $\boldsymbol{\lambda}^{k*}$, $\boldsymbol{\eta}^{k*}$ are optimal, the canonical differential equations

$$\dot{\boldsymbol{x}} = \frac{\partial H^k}{\partial \boldsymbol{\lambda}} \qquad \dot{\boldsymbol{\lambda}}^k = -\frac{\partial H^k}{\partial \boldsymbol{x}}$$

are fulfilled and

$$\eta \begin{cases} = 0 & \text{iff } g\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) > 0 \\ \leq 0 & \text{iff } g\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = 0 \end{cases} \quad \Longleftrightarrow \quad \eta \begin{cases} = 0 & t \in [0, t_1) \cup (t_2, t_f] \\ \leq 0 & t \in [t_1, t_2] \end{cases}$$

Additionally η^k have to fulfill the sign conditions,

$$(-1)^j \frac{\mathrm{d}^j}{\mathrm{d}t^j} \eta^k(t) = \eta^{k-j}(t) \leq 0, \quad j = 1, \cdots, k$$

and the final conditions

$$\lim_{\delta \to 0^-} \frac{\mathrm{d}^j}{\mathrm{d}t^j} \eta^k(t_2 + \delta) = 0, \quad j = 0, \cdots, k-2 \text{ if } k \geq 2$$

For $k = 1, \cdots, m_g$ the active constraints have to fulfill the following at the entry point:

$$\lim_{\delta \to 0^+} \boldsymbol{\lambda}^k(t_1 + \delta) = \left(\lim_{\delta \to 0^-} \boldsymbol{\lambda}^k(t_1 + \delta) \right) - \sum_{j=1}^{k} \beta^j \cdot \frac{\partial g^{(j-1)}\big(\boldsymbol{x}(t_1)\big)}{\partial \boldsymbol{x}}$$

with $\beta^j \leq 0$. And on the exit point the condition

$$\lim_{\delta \to 0^+} \boldsymbol{\lambda}^k(t_2 + \delta) = \lim_{\delta \to 0^-} \boldsymbol{\lambda}^k(t_2 + \delta)$$

has to be fulfilled for $k = 1, \cdots, m_g$. That is, the adjunct variables of the x_i that appear in $g^{(j-1)}$ are discontinuous at the entry point and continuous at the exit point of the active constraint.

The optimal control than has to minimize the augmented Hamiltonian over U:

$$H\big(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t), \boldsymbol{\eta}^*(t)\big) = \min_{\tilde{\boldsymbol{u}} \in U} H\big(\boldsymbol{x}^*(t), \tilde{\boldsymbol{u}}, \boldsymbol{\lambda}^*(t), \boldsymbol{\eta}^*(t)\big), \quad \forall t \in [0, t_f]$$

As the problem is autonomous, the Hamiltonian is constant sectionally constant w.r.t. time t and it can change when constraints become active/inactive.

To get back and forth in different formulations for k, the following recursion can be defined. For $m_g \geq 1$:

$$\eta^1(t) = \begin{cases} \nu_2 + \int_t^{t_2} \eta^0(\tau)\,\mathrm{d}\tau & \text{iff } t \in [t_1, t_2] \\ 0 & \text{else} \end{cases}$$
$$\beta^1 = \nu_1 + \eta^1(t_1)$$

And for $m_g \geq 2$ and $k \geq 2$:

$$\eta^k(t) = \begin{cases} \int_t^{t_2} \eta^{k-1}(\tau)\,\mathrm{d}\tau & \text{iff } t \in [t_1, t_2] \\ 0 & \text{else} \end{cases}$$
$$\beta^k = \eta^k(t_1)$$
$$\boldsymbol{\lambda}^k(t) = \boldsymbol{\lambda}^0(t) - \sum_{j=1}^{k} \eta^j(t) \cdot \frac{\partial g^{(j-1)}\big(\boldsymbol{x}(t)\big)}{\partial \boldsymbol{x}}$$

Where $\nu_1, \nu_2 \leq 0$ are given by

$$\lim_{\delta \to 0^+} \boldsymbol{\lambda}^0(t_\Delta + \delta) = \left(\lim_{\delta \to 0^-} \boldsymbol{\lambda}^0(t_\Delta + \delta) \right) - \nu_\Delta \cdot \frac{\partial g_i\big(\boldsymbol{x}(t_\Delta), \boldsymbol{u}(t_\Delta)\big)}{\partial \boldsymbol{x}}$$

for $\Delta = 1$ and $\Delta = 2$.

Notes

- There are three different types of active constraints possible:
  1. Contact point
     The states control directly “into” the infeasible region and have to do an immediate turn when having contact with the constraint (like bumping a car into a wall and bouncing off).
  2. Touch point
     The states just merely touch the border of the infeasible region and no changes have to be made to the control (like a a go-around with merely touching the ground).

Order m_g	Touch Point	Boundary Arc
1	no	yes
even	yes	yes
odd	yes	no

Table 8.1: Possibilities for active state-only constraints in optimal control given the order of the constraint.

3. Boundary arc
   The states have to remain a while on the border (the "active section") until going away again (like sliding along the edge on an ice rink).

- If the Hamiltonian is regular, there are multiple possibilities shown in Table 8.1.
- Typically when working with the necessary conditions directly (analytically), H^0 or H^{m_g} are used.
- The conditions and relations for H^k are primarily useful to analyze for the solutions can be transformed one into another and are as such equivalent.
- The solutions for $\boldsymbol{x}$ and $\boldsymbol{u}$ are the same for all H^k, but the $\boldsymbol{\lambda}$ and $\boldsymbol{\eta}$ might be different on the active sections (and only on these).

8.4.3 Examples

Optimal Robot Control

Energy Minimization Problem

8.4.4 Summary

- The conditions derivable from the maximum principle for yields a multi-point boundary value problem for $\boldsymbol{x}^*$ and $\boldsymbol{\lambda}^*$.
- The interior switching points $t_{S,i}$ result of:
  - Switching points at bang-bang and singular control.
  - The entry and exit into/of active sections of mixed or state constraints.
  - Other constraints at interior time steps.
  - A dynamic that is only defined piecewise.
- The switching structure (i.e. the number of order of switching points) in normally unknown!
- The switching structure has to be determined otherwise, e.g. by solving the unconstrained problem and successively tighten the constraints (continuation/homotopy technique).
- Other things like discontinuities in the system dynamics and state constraints at interior points are not covered here…
- Optimal control is hell more complex than it seems in this summary!

9 Calculating Optimal Trajectories

This chapter covers the numerical calculation of optimal trajectories exploiting the theoretical results of chapter 8.

9.1 First Computation Methods

9.1.1 Dynamic Programming

The technique of *dynamic programming* was introduced by Richard Bellman for numerically solving optimal control problems:

- Start from the final state $\boldsymbol{x}_f$ at t_f, the discretized Bellman Equation is used to successively iterate forward in time until $t = 0$.
- The effort raises exponentially with the dimensions n_x of $\boldsymbol{x}$ (the "curse of dimensionality").
- A lot of current research is based on dynamic programming, especially in the field of reinforcement learning.

9.1.2 Gradient Methods (Min-H Methods)

1. Starting with an approximation for $\boldsymbol{x}$, $\boldsymbol{\lambda}$, $0 \leq t \leq t_f$, calculate an approximation for $\boldsymbol{u}$ by minimizing the Hamiltonian at discrete time steps t_i, e.g. using gradient methods.
2. Use this approximation for $\boldsymbol{u}$ to solve the dynamics numerically using forward integration and calculate the adjunct variables using backward iteration.
3. Repeat with the new approximations for $\boldsymbol{x}$, $\boldsymbol{\lambda}$.

9.2 Indirect Methods

Indirect methods are based on the necessary conditions and try to solve the arising boundary value problem.

- Advantages:
  - As it depends on the necessary conditions, the found solution is likely to be optimal.
  - By using highly precise integration methods, the solution can be determined with a high precision. This is especially useful, e.g. for satellite missions.
  - Increasingly powerful tools like automatic differentiation ease this method.
- Disadvantages:

- A lot of expert knowledge is needed to derive the optimality conditions and formulate the multi-point boundary value problem.
- The explicit derivation of the necessary conditions can be really costly.
- The correct, optimal switching structure is not known a-prior.
- Numerical methods need initial approximations for $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ which are, especially for $\boldsymbol{\lambda}$, hard to get.

9.3 Direct Methods

Direct methods avoid the explicit formulation of the necessary conditions by discretizing the control and possible the state variables. They transform the optimal control problem to a nonlinear optimization problem and employ methods of the nonlinear optimization.

- The user does not have to handle adjunct variables, switching structures, ...
- The efficiency of the method highly depends on the discretization and the employed nonlinear optimization problem.
- Progress in nonlinear optimization (especially SQP) yield highly efficient direct methods.
- Most commonly used.

All of the following assumes an optimal control problem given as:

$$\begin{aligned} \min_{\boldsymbol{u}} J[\boldsymbol{u}] &= \phi\big(\boldsymbol{x}(t_0), \boldsymbol{x}(t_f), t_f\big) \\ \text{subject to} \qquad \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big), \quad t \in [t_0, t_f] \\ x_i(0) &= x_{i,0}, \quad i \in I_0 \subseteq \{1, \cdots, n_x\} \\ x_j(t_f) &= x_{j,f}, \quad j \in I_f \subseteq \{1, \cdots, n_x\} \\ \boldsymbol{g}\big(\boldsymbol{x}(t)\big) &\geq \boldsymbol{0} \\ \boldsymbol{g}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) &\geq \boldsymbol{0} \end{aligned}$$

The biggest difference is that this problem does not have a Lagrangian term, but as the Lagrangian can be transformed to a Mayer term, this is no loss of generality.

9.3.1 Direct Collocation Methods

Direct collocation methods approximate the control state variables $\boldsymbol{u}$, $\boldsymbol{x}$ using piecewise polynomial functions.

First of all, the interval $[t_0, t_f]$ is split into n_s segments $[t_i, t_{i+1}]$ where

$$t_i = t_0 + \tau_i \cdot (t_f - t_0) \quad \text{with} \quad 0 = \tau_1 < \tau_2 < \cdots < \tau_{n_s+1} = 1$$

with all τ_i given. The resulting segmented space is called the *mesh*. In every of these segments, $\boldsymbol{u}$ and $\boldsymbol{x}$ and approximated using a polynomial. As $\boldsymbol{u}$ is part of the integrand for $\boldsymbol{x}$, it is reasonable to choose a higher polynomial degree for $\boldsymbol{x}$ than $\boldsymbol{u}$.

Also let $t_{i+1/2} \coloneqq t_i + h_i/2$ with $h_i \coloneqq t_{i+1} - t_i$.

First Discretization, Constant Control

A first idea is to approximate the control using constant functions

$$\tilde{u}_i(t) = u_i(t_{j+1/2}), \quad t_j \le t < t_{j+1}, \quad j = 1, \cdots, n_s, \quad i = 1, \cdots, n_u$$

and to approximate the state using linear functions:

$$\tilde{x}_i(t) = x_i(t_j) + \frac{t - t_j}{h_j}\big(x_i(t_{j+1}) - x_i(t_j)\big), \quad t_j \le t < t_{j+1}, \quad j = 1, \cdots, n_s, \quad i = 1, \cdots, n_x$$

The optimization variables are given as the vector

$$\begin{aligned}\boldsymbol{p} &= \begin{bmatrix}\boldsymbol{x}(t_1) & \boldsymbol{u}(t_{1+1/2}) & \boldsymbol{x}(t_2) & \boldsymbol{u}(t_{2+1/2}) & \cdots & \boldsymbol{x}(t_{n_s}) & \boldsymbol{u}(t_{n_s+1/2}) & \boldsymbol{x}(t_{n_s+1}) & t_f\end{bmatrix}^T \\ &= \begin{bmatrix}\boldsymbol{p}_1^x & \boldsymbol{p}_1^u & \boldsymbol{p}_2^x & \boldsymbol{p}_2^u & \cdots & \boldsymbol{p}_{n_s}^x & \boldsymbol{p}_{n_s}^u & \boldsymbol{p}_{n_s+1}^x & t_f\end{bmatrix}^T\end{aligned}$$

where the final time t_f might be left out if it is given. The second row specifies a shorthand notation for the parameter of the approximation of $\boldsymbol{x}/\boldsymbol{u}$. The approximations shall now be determined subject to:

- Minimize the objective functional.
- Fulfill the differential equations and inequality constraints at the center of a segment.
- Comply with the initial and final conditions.

This yields the following finite-dimensional nonlinear optimization problem with $n_p = n_s \cdot (n_x + n_u) + n_x + 1$ optimization variables (or one less if the final time is given):

$$\begin{aligned}\min_{\boldsymbol{p} \in \mathbb{R}^{n_p}} \varphi(\boldsymbol{p}), \quad \varphi(\boldsymbol{p}) &= \phi\big(\boldsymbol{p}_1^x, \boldsymbol{p}_{n_s+1}^x, t_f\big) \\ \text{subject to} \quad \boldsymbol{f}\big(\tilde{\boldsymbol{x}}(t_{j+1/2}), \tilde{\boldsymbol{u}}(t_{j+1/2})\big) - \dot{\tilde{\boldsymbol{x}}}(t_{j+1/2}) &= \boldsymbol{0}, \quad j = 1, \cdots, n_s \\ \tilde{x}_i(t_0) &= x_{i,0}, \quad i \in I_0 \subseteq \{1, \cdots, n_x\} \\ \tilde{x}_j(t_f) &= x_{j,f}, \quad j \in I_f \subseteq \{1, \cdots, n_x\} \\ \boldsymbol{g}\big(\tilde{\boldsymbol{x}}(t_{j+1/2}), \tilde{\boldsymbol{u}}(t_{j+1/2})\big) &\ge \boldsymbol{0}, \quad j = 1, \cdots, n_s\end{aligned}$$

With $\tilde{\boldsymbol{u}}(t_{j+1/2}) = \boldsymbol{p}_j^u$, $\tilde{\boldsymbol{x}}(t_{j+1/2}) = \frac{1}{2}\big(\boldsymbol{p}_j^x + \boldsymbol{p}_{j+1}^x\big)$ and $\dot{\tilde{\boldsymbol{x}}}(t_{j+1/2}) = \frac{1}{h_j}\big(\boldsymbol{p}_{j+1}^x - \boldsymbol{p}_j^x\big)$.

But low approximations like this have to use lots of mesh points to get good approximations. Hence, approximations of higher degree might be useful.

Second Discretization, Linear Control

Approximate the control using linear functions:

$$\tilde{u}_i(t) = u_i(t_j) + \frac{t - t_j}{h_j}\big(u_i(t_{j+1}) - u_i(t_j)\big), \quad t_j \le t < t_{j+1}, \quad j = 1, \cdots, n_s, \quad i = 1, \cdots, n_u$$

And approximate the state using linear functions:

$$\tilde{x}_i(t) = \sum_{k=0}^{3} c_{i,j,k} \left(\frac{t - t_j}{h_j}\right)^k, \quad t_j \le t < t_{j+1}, \quad j = 1, \cdots, n_s, \quad i = 1, \cdots, n_u$$

Here, $c_{i,j,k}$ is the k-th coefficient for the i-th component in the j-th segment $[t_j, t_{j+1}]$, yielding four unknown parameters per component and segment. One of the parameters is determined by requiring continuity an the left side of the segment. The other three parameters are determined by requiring the fulfilling of the ODEs at three time steps in the j-th segment:

- Gauss Points: $t_{j+1/2} - \sqrt{3/5}h_j, \quad t_{j+1/2}, \quad t_{j+1/2} + \sqrt{3/5}h_j$
  - Theoretically provide the best approximation.
  - Do not provide differentiable transitions between the segments.
  - Need $3n_s$ evaluations of the ODEs.
- Lobatto Points: $t_j, \quad t_{j+1/2}, \quad t_{j+1}$
  - Provide continuously differentiable transitions between the segments.
  - Need $2n_s + 1$ evaluations of the ODEs.

When using Lobatto points, the NLP parameters per segment can be reduced from four to two, thus reducing the collocation constraints from three to one per segment. But the remaining constraints are more nonlinear...Hence, the dimension of the NLP is similar to the constraint approximation, bit given a much better approximation of the solution.

By plugging the constraints/locations of the Lobatto points into the state approximation, this yields the following formulas for the coefficients:

$$\begin{bmatrix} c_{i,j,0} \\ c_{i,j,1} \\ c_{i,j,2} \\ c_{i,j,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & h_j & 0 & 0 \\ -3 & -2h_j & 3 & -h_j \\ 2 & h_j & -2 & h_j \end{bmatrix} \cdot \begin{bmatrix} p^x_{i,j} \\ p^{\dot{x}}_{i,j} \\ p^x_{i,j+1} \\ p^{\dot{x}}_{i,j+1} \end{bmatrix} = \begin{bmatrix} p^x_{i,j} \\ h_j p^{\dot{x}}_{i,j} \\ -3p^x_{i,j} - 2h_j p^{\dot{x}}_{i,j} + 3p^x_{i,j+1} - h_j p^{\dot{x}}_{i,j+1} \\ 2p^x_{i,j} + h_j p^{\dot{x}}_{i,j} - 2p^x_{i,j+1} + h_j p^{\dot{x}}_{i,j+1} \end{bmatrix}$$

Where $p^{\dot{x}}_{i,j} = \dot{x}_i(t_j) = f_i\big(\boldsymbol{x}(t_j), \boldsymbol{u}(t_j)\big) = f_i(\boldsymbol{p}^x_j, \boldsymbol{p}^u_j)$. Hence, all coefficients can be computed using only the parameter vector

$$\begin{aligned} \boldsymbol{p} &= \begin{bmatrix} \boldsymbol{x}(t_1) & \boldsymbol{u}(t_{1+1/2}) & \boldsymbol{x}(t_2) & \boldsymbol{u}(t_{2+1/2}) & \cdots & \boldsymbol{x}(t_{n_s}) & \boldsymbol{u}(t_{n_s+1/2}) & \boldsymbol{x}(t_{n_s+1}) & \boldsymbol{u}(t_{n_s+1+1/2}) & t_f \end{bmatrix}^T \\ &= \begin{bmatrix} \boldsymbol{p}^x_1 & \boldsymbol{p}^u_1 & \boldsymbol{p}^x_2 & \boldsymbol{p}^u_2 & \cdots & \boldsymbol{p}^x_{n_s} & \boldsymbol{p}^u_{n_s} & \boldsymbol{p}^x_{n_s+1} & \boldsymbol{p}^u_{n_s+1} & t_f \end{bmatrix}^T \end{aligned}$$

with one parameter more than the constant control approximation. That is, $n_p = (n_s + 1)(n_x + n_u) + 1$ parameters (or one less if the final time is fixed). This yields the following nonlinear optimization problem:

$$\begin{aligned} \min_{\boldsymbol{p} \in \mathbb{R}^{n_p}} \varphi(\boldsymbol{p}), \quad \varphi(\boldsymbol{p}) &= \phi\big(\boldsymbol{p}^x_1, \boldsymbol{p}^x_{n_s+1}, t_f\big) \\ \text{subject to} \quad \boldsymbol{f}\big(\tilde{\boldsymbol{x}}(t_{j+1/2}),\, \tilde{\boldsymbol{u}}(t_{j+1/2})\big) - \dot{\tilde{\boldsymbol{x}}}(t_{j+1/2}) &= \boldsymbol{0}, \quad j = 1, \cdots, n_s \\ \tilde{x}_i(t_0) &= x_{i,0}, \quad i \in I_0 \subseteq \{1, \cdots, n_x\} \\ \tilde{x}_j(t_f) &= x_{j,f}, \quad j \in I_f \subseteq \{1, \cdots, n_x\} \\ \boldsymbol{g}\big(\tilde{\boldsymbol{x}}(t_{j+1/2}), \tilde{\boldsymbol{u}}(t_{j+1/2})\big) &\geq \boldsymbol{0}, \quad j = 1, \cdots, n_s \end{aligned}$$

With $\tilde{\boldsymbol{u}}(t_{j+1/2}) = \boldsymbol{p}^u_j$, $\tilde{\boldsymbol{x}}(t_{j+1/2}) = \frac{1}{2}\big(\boldsymbol{p}^x_j + \boldsymbol{p}^x_{j+1}\big)$ and $\dot{\tilde{\boldsymbol{x}}}(t_{j+1/2}) = \frac{1}{h_j}\big(\boldsymbol{p}^x_{j+1} - \boldsymbol{p}^x_j\big)$.

Discretization of Inequality Constraints

In the NLP, the inequality constraints are evaluated at the points t_j, i.e. the mesh points. But why not at other points, e.g. $\boldsymbol{g}(t_{j+1/2})$? As of the theoretical results, one control is determined by

$$g^{(m_g)}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) = \frac{\mathrm{d}^{m_g}}{\mathrm{d}t^{m_g}} g\big(\boldsymbol{x}(t)\big) \equiv 0$$

in an active segment $t \in [t_1, t_2]$. Hence, for consistency, the number of degrees of freedom on a potential arc have to equal the number of degrees of freedom of the discretized control constraints. This is only ensured when the inequality constraints are evaluated at the mesh points t_j.

Sparsity

In direct collocation methods, the resulting NLPs have sparse gradients and Jacobians. For example, the memory complexity of the full Jacobian is $\mathcal{O}(n_s^2)$, while the non-zero elements rise linearly. By exploiting this sparse structure, the efficiency for NLP solvers can be highly enhanced!

Convergence Properties and Solution Validation

By studying the Lagrangian of the NLP, it is possible to show that a gradient method that is using the Lagrangian for determining the step size, the maximum principle is taken into account in a discretized matter. That way, using direct collocation methods, it is possible to compute approximations for the adjunct variables and the multipliers $\boldsymbol{\eta}^0$. Hence, the theoretical optimality conditions can be validated afterwards!

For autonomous problems, the Hamiltonian has to be sectionally constant which can be validated using the calculated approximations $\tilde{\boldsymbol{a}}(t)$, $\tilde{\boldsymbol{u}}(t)$, $\tilde{\boldsymbol{\lambda}}^0(t)$, $\tilde{\boldsymbol{\eta}}^0(t)$. Another thing that can be validates is whether the initial and final conditions for the adjunct variables are fulfilled.

Derivation

Choosing the Mesh Points

Direct collocation methods typically start with a rough grid and a rough approximation for $\boldsymbol{x}^*$ and $\boldsymbol{u}^*$ that is then successively adjusted until the solution is smooth enough. But how to do this mesh adjustments?

- Ideally: A segment-wise approximation error $\left\|\tilde{\boldsymbol{x}}(t) - \boldsymbol{x}^*(t)\right\|$, $\left\|\tilde{\boldsymbol{u}}(t) - \boldsymbol{u}^*(t)\right\|$.
- Alternatively: Use the error of in the ODE and inequality constraints: $\left\|\boldsymbol{f}\left(\tilde{\boldsymbol{x}}(t), \tilde{\boldsymbol{u}}(t)\right) - \dot{\tilde{\boldsymbol{x}}}(t)\right\|$, $\left\|\boldsymbol{g}_{\_}\left(\tilde{\boldsymbol{x}}(t), \tilde{\boldsymbol{u}}(t)\right)\right\|$ with $g_{\_} \coloneqq |g|$ as the element-wise absolute value.

And refine the mesh until $\|\cdot\| \leq \varepsilon_{\text{tol}}$.

Segment-Wise Error Estimation Assuming $\tilde{\boldsymbol{u}}(t) = \boldsymbol{u}^*(t)$, theories of collocation methods can be used to estimate the error

$$\left\|\tilde{\boldsymbol{x}}(t) - \boldsymbol{x}^*(t)\right\|$$

where $\boldsymbol{x}^*$ is the "real" solution of the BVP. But this method does not work well for direct collocation methods.

Segment-Wise Error Estimation of Time-Continuous Constraints Simple strategy that works good in practice: Check the fulfilling of the constraints at test points, e.g. for the second discretization:

$$\left\|\boldsymbol{f}\left(\tilde{\boldsymbol{x}}(t), \tilde{\boldsymbol{u}}(t)\right) - \dot{\tilde{\boldsymbol{x}}}(t)\right\| \quad \left\|\boldsymbol{g}_{\_}\left(\tilde{\boldsymbol{x}}(t), \tilde{\boldsymbol{u}}(t)\right)\right\| \quad \text{for} \quad t = t_{j+k/4} = t_j + \frac{k}{4} h_j, \quad k = 1, 2, 3$$

Then successively split the segments with too big errors into smaller segments.

Segment-Wise Estimation of Optimality Error The previous error estimation methods only take the constraints into account, not that this is an optimal control problem. But it is also possible to approximate the "optimality error" using the calculated values for the adjunct variables (e.g. using quadrature rules):

$$L = \phi\big(\tilde{\boldsymbol{x}}(0), \tilde{\boldsymbol{x}}(t_f), t_f\big) + \sum_{j=1}^{n_s} \underbrace{\int_{t_j}^{t_{j+1}} \Big[\tilde{\boldsymbol{\lambda}}^T(t) \cdot \Big(\boldsymbol{f}\big(\tilde{\boldsymbol{x}}(t), \tilde{\boldsymbol{u}}(t)\big) - \dot{\tilde{\boldsymbol{x}}}(t)\Big) + \tilde{\boldsymbol{\eta}}^T(t) \cdot \boldsymbol{g}\big(\tilde{\boldsymbol{x}}(t), \tilde{\boldsymbol{u}}(t)\big)\Big] \,\mathrm{d}t}_{\text{Optimality Error in the } j\text{-th Segment}}$$

Implementation

The general approach to implement direct collocation methods is shown inalgorithm 7.

Algorithm 7: Direct Collocation Algorithm.

1 **Initialization:** Choose a start grid $\big(\tau_j^{(0)}\big)_{j=1}^{n_s^{(0)}+1}$ and initial values $\boldsymbol{p}_j^{x,(0)} = \boldsymbol{x}(t_j)$, $\boldsymbol{p}_j^{u,(0)} = \boldsymbol{u}(t_{j+1/2})$

2 **while** *not converged (error in $\dot{\boldsymbol{x}}$, maximum iterations or maximum mesh size)* **do**

3 Adjust the mesh: $\big(\tau_j^{(k)}\big)_{j=1}^{n_s^{(k)}+1}$

4 Solve the resulting NLP:

$$\begin{aligned} \min_{\boldsymbol{p} \in \mathbb{R}^{n_p}} \ & \varphi(\boldsymbol{p}) \\ \text{subject to} \quad & \boldsymbol{a}(\boldsymbol{p}) = \boldsymbol{0} \\ & \boldsymbol{b}(\boldsymbol{p}) \geq \boldsymbol{0} \end{aligned}$$

5 $k \leftarrow k + 1$

Notes

- For a rough mesh, the tolerances ε_{opt} and ε_{ft} do not have to be chosen too tight as the NLP solver might not ding a solution then.
- While successively refining the grid, the tolerances can be set to tighter values.
- Ideally, the most SQP iterations are needed for the first two meshes. For the high-dimensional NLPs for fine grids, only a few iterations are needed.
- System parameters can be optimized simultaneously by adding them to the variable vector $\boldsymbol{p}$.

Application: Optimal Robot Control

9.3.2 Direct Shooting Methods

Similar to the direct collocation methods, shooting methods rely on a segmentation of the the interval $[t_0, t_f]$ into n_s segments $[t_j, t_{j+1}]$ with $n_s + 1$ mesh point t_j. Then the control $\boldsymbol{u}$ is approximated per section, e.g. using constant or linear functions.

Direct shooting methods then solve various initial value problems until one fulfills the boundary conditions. Informally speaking, multiple trajectories are "shot into the room" until one is feasible. More formally, the process is:

1. Divide the interval into n_s segments and approximate the control $\boldsymbol{u}$, e.g. using a piecewise linear function $\tilde{\boldsymbol{u}}$. Then parameterize $\boldsymbol{u}$ by $\boldsymbol{p} = \begin{bmatrix}\boldsymbol{u}(t_1) & \cdots & \boldsymbol{u}(t_f) & t_f\end{bmatrix}^T \in \mathbb{R}^{n_u \cdot (n_s+1)}$.
2. Simulate the movement by forward-integrating the system dynamics starting from the initial values and by using the approximated control $\tilde{\boldsymbol{u}}$. This gives an approximation of the state, $\tilde{\boldsymbol{x}}$.
3. Calculate the objective and the violations of the constraints, especially the final constraints.
4. Optimize $\boldsymbol{p}$ such that the objective is minimized while fulfilling the constraints. Repeat.

But the calculation of the NLP gradients and the Jacobians needs calculating the sensitivity matrix $\frac{\partial \boldsymbol{x}(t;\boldsymbol{p})}{\partial \boldsymbol{p}}$.

- Advantages:
  - The resulting dimension of the NLPs are lot less than for direct collocation methods.
  - In every iteration of the NLP solver, a solution of the ODEs is available.
- Disadvantages:
  - Solving the initial value problem may highly depend on the approximations $\tilde{\boldsymbol{u}}$, $\tilde{t}_f$. To increase the robustness, multiple shooting methods can be used.
  - Approximations for the adjunct variables are not as natural as in direct collocation methods, but possible.

9.4 Notes

- Some big disadvantages of indirect methods, that good approximations of the optimal state- and adjunct variables and the switching structure are needed, can be overcome by using a direct method first.
- The majority of optimal control problems are nowadays solved using direct methods (other approaches exist besides direct collocation and shooting methods).
- Important applications are for example, aerospace engineering, robots, vehicles, process engineering, economy, biology, ecology, ...
- Direct collocation methods are also called "simultaneous simulation and optimization".
- Direct shooting methods are also called "iterative simulation and optimization".
  - Two different algorithms are used for simulation and optimization.
  - Highly efficient and stable calculation of the sensitivity matrix using special integration techniques.

10 Optimal Feedback Control

Applying the computed control trajectory directly as an open loop (feedforward) control as illustrated in Figure 10.1 would cause growing divergence from the nominal state $\boldsymbol{x}_d^*(t)$. This can be caused by e.g.:

- Errors in the model (complex models cannot be 100% accurate).
- Noise: During the run of a dynamic process noise may affect the systems behavior.

Hence, the wanted end state might not be reached when just using feedforward control.

10.1 Classical Feedback Control (Position Control)

It seems obvious to just use a classical position control for the nominal stater state trajectory $\boldsymbol{x}^*(t)$ (a set point trajectory control) as illustrated in Figure 10.2. But this causes "ringing" around the nominal trajectory, e.g. using a PID control law. Additionally, using a position control has more flaws:

- An individual control for each state component is merely possible.
- The quality of the position control depends on the desired states and the control parameters.
- Returning to the nominal trajectory using control laws is not the optimal trajectory from the disturbed state as the initial state to the final state!
- Also, using position control can cause violations of the constraints, e.g. a bang-bang control always operates on the border of the control constraints. Hence, returning to the nominal trajectory might cause overshooting this constraints.

10.2 Optimal Feedback Control

If the real trajectory at time step t_1 differs from the nominal state $\boldsymbol{x}^*(t_1; \boldsymbol{x}_0)$ that was calculated starting from $\boldsymbol{x}_0$ it would be optimal to now following a new trajectory with the initial state $\boldsymbol{x}_d(t_1)$. But how to calculate this new trajectory? These computations have to be done anytime. But data from the old trajectory can be used for faster calculations!

These re-computations would not be necessary if the optimal control trajectory would not be calculated as a function of time $\boldsymbol{u}^*(t)$ (as a feedforward control), but as a function of the initial state $\boldsymbol{u}^*(\boldsymbol{x}_0)$, i.e. as a feedback control. Sadly, it is nearly impossible to calculate this function except for some special cases...

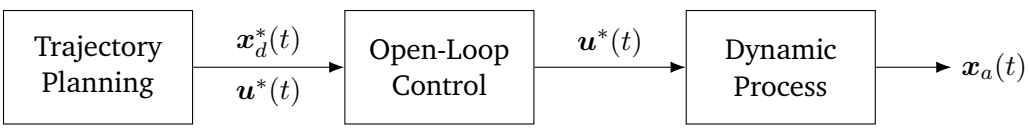

Figure 10.1: Feedforward Control

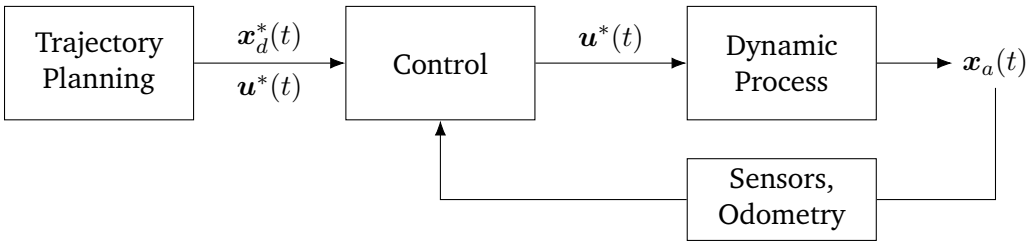

Figure 10.2: Feedback Control

10.3 Linear Quadratic Regulator (LQR)

Linear systems with quadratic objective are the only system for which the optimal feedback control $\boldsymbol{u}^*(\boldsymbol{x}_0)$ can be computed in closed form! A linear system with quadratic objective is given as

$$\begin{aligned} \min_{\boldsymbol{u}} J[\boldsymbol{u}], \quad J[\boldsymbol{u}] &= \int_{t_0}^{t_f} \left(\boldsymbol{x}^T(t)\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}^T(t)\boldsymbol{\Gamma}\boldsymbol{u}(t) \right) \mathrm{d}t \\ \dot{\boldsymbol{x}}(t) &= \boldsymbol{A}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) \\ \text{subject to} \quad \boldsymbol{x}(t) &= \boldsymbol{x}_0 \\ \boldsymbol{x}(t_f) \quad &\text{free} \end{aligned}$$

with a symmetric and positive definite matrix $\boldsymbol{Q}$ and a diagonal Matrix $\boldsymbol{\Gamma} = \mathrm{diag}(\gamma_1, \cdots, \gamma_{n_u})$. The optimal control law is then given as

$$\boldsymbol{u}(\boldsymbol{x}) = -\boldsymbol{\Gamma}^{-1}\boldsymbol{B}^T(t)\boldsymbol{P}(t)\boldsymbol{x}$$

where the matrix $\boldsymbol{P}(t)$ is found by solving the matrix-Riccati ODE

$$\dot{\boldsymbol{P}}(t) + \boldsymbol{Q} + \boldsymbol{A}^T(t)\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{A}(t) - \boldsymbol{P}(t)\boldsymbol{\Gamma}^{-1}\boldsymbol{B}(t)\boldsymbol{B}^T(t)\boldsymbol{P}(t) = \boldsymbol{O}$$

with the boundary condition $\boldsymbol{P}(t_f) = \boldsymbol{O}$.

If the system is time invariant, i.e. $\dot{\boldsymbol{A}} = \boldsymbol{O}$, $\dot{\boldsymbol{B}} = \boldsymbol{O}$ or $t_f = \infty$, it follows $\dot{\boldsymbol{P}} = \boldsymbol{O}$ and hence $\boldsymbol{P}$ is given by solving the algebraic matrix-Riccati equation:

$$\boldsymbol{Q} + \boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{P}\boldsymbol{A} - \boldsymbol{P}\boldsymbol{\Gamma}^{-1}\boldsymbol{B}\boldsymbol{B}^T\boldsymbol{P} = \boldsymbol{O}$$

As $\boldsymbol{P}$ is symmetric, these are $n_x \cdot (n_x + 1)/2$ equations to determine every element of $\boldsymbol{P}$.

These matrix-Riccati algebraic/differential equations can be solved efficiently using special numerical methods. LQR systems can approximate a lots of systems and are therefore more useful than it appears at first (e.g. active damping in cars or linearized inverted pendulum). In theory, LQR methods can be applied to any system using an infinite-dimensional nonlinear embedding (“measurement”) which can be finitely approximated, still yielding good results. Another approach is iterative LQR (iLQR) that linearizes the system in every time step.

10.3.1 Derivation

10.4 Neighboring Extremals

TO approximately the new trajectory $\boldsymbol{x}^*\big(t; \boldsymbol{x}_a(t_1)\big)$, all information that have already been expended for calculating the trajectory $\boldsymbol{x}^*(t; \boldsymbol{x}_0)$ can be used. These updates have to be be done continuously.

10.4.1 Indirect Methods

Indirect methods calculates a solution trajectory $\boldsymbol{x}^*(t)$, $\boldsymbol{u}^*(t)$, $\boldsymbol{\lambda}^*(t)$, $\boldsymbol{\eta}^*(t)$ as the numerical solution of the multi-point boundary value problem rising from the necessary optimality conditions. By Taylor-expanding the disturbed trajectory $\boldsymbol{x}^*(t; \boldsymbol{x}_0 + \boldsymbol{\varepsilon}_0)$ around the nominal trajectory yields a linear-quadratic optimal control problem (called "accessory minimum problem") for

$$\delta\boldsymbol{x}(t), \quad \delta\boldsymbol{u}(t), \quad \delta\boldsymbol{\lambda}(t)$$

Different variants of this are possible, e.g. the repeated correction method:

1. Calculate $\boldsymbol{x}^*(t)$, $\boldsymbol{u}^*(t)$, $\boldsymbol{\lambda}^*(t)$, $\boldsymbol{\eta}^*(t)$ using the multi-point BVP width multiple-shooting methods.
2. Feedback schema for noisy nominal trajectory $\boldsymbol{u}^*(t_0; \boldsymbol{x}_0 + \boldsymbol{\varepsilon}_0) = \boldsymbol{u}^*(t_0; \boldsymbol{x}_0) + \delta\boldsymbol{u}(t_0)$ containing the nominal control and a correction term.
3. Calculate the control correction using: $\Delta\boldsymbol{u}(t_0) = \boldsymbol{K}_1(t_0) \cdot \partial\boldsymbol{x}(t_0) + \boldsymbol{K}_2(t) \cdot \partial\boldsymbol{\lambda}(t_0)$
4. Apply this repeatedly for different time steps $t_0 = t_1$.

- Advantages:
  - Fast, numerical calculation.
  - First-order optimal trajectories.
  - Can be used to correct noise in the model parameters of the ODE.
- Disadvantages:
  - Only locally applicable along a nominal trajectory (i.e. "little" noise that does not change the switching structure)
  - Depends on the multi-point BVP of the necessary conditions. Hence, application is time consuming as the ODEs have to be formulated.

10.4.2 Direct Methods

Direct methods solve the problem by discretizing the problem, resulting in a NLP that contains the initial value $\boldsymbol{x}(t_0)$ in its equality constraints. Studying the dependency on this parameters leads to sensitivity and stability analysis of nonlinear optimization problems.

For direct collocation methods it can be shown that the linear-quadratic optimal control problem is analogous to a quadratic problem (QP) for calculating the corrections $\boldsymbol{p}^*(\boldsymbol{x}_0 + \boldsymbol{\varepsilon}_0) = \boldsymbol{p}^*(\boldsymbol{x}_0) + \delta\boldsymbol{p}$.

10.4.3 Nonlinear Model Predictive Control (NMPC)

For *nonlinear model predictive control*, a series of optimal control problems is solved for a varying time horizon P_j. These methods are commonly used for chemical processes with slow reaction time and aerospace engineering, but not so often in fast environments like robot movements.

10.5 Numerical Synthesis of the Nonlinear Feedback Control

Direct collocation methods can compute optimal control problems with different initial conditions $\boldsymbol{x}_0$ pretty fast and robust. Using an appropriate "step size control", multiple neighboring trajectories can be computed kind of automatic.

- First approach for synthesis $\boldsymbol{u}^*(\boldsymbol{x})$: Calculate $\boldsymbol{u}^*$ using the HJB equation where $\boldsymbol{x}^*(t)$, $\boldsymbol{\lambda}^*(t)$ are approximated using reference trajectories for $\boldsymbol{x}_a(t_1)$.
- Second approach: Approximate $\boldsymbol{u}^*(t)$, e.g. an approximation that has been trained on lots of open-loop optimal trajectories.

11 Further Topics on Optimal Control

11.1 Inverse Optimal Control

Inverse optimal control approaches the following problem:

- *Given* a dynamic process $\dot{\boldsymbol{x}} = \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big)$ and measurements of a run $\boldsymbol{x}_k = \boldsymbol{x}(t_k) + \boldsymbol{\varepsilon}_k$ that is in general noisy,
- *Assume* the run was controlled optimally,
- *Find* the objective function that was used.

One possible approach is the linear combination of multiple "basis" functionals $J_k[\boldsymbol{u}]$, e.g. one for minimum energy and one for minimum time:

$$J^*[\boldsymbol{u}] = \sum_{k=1}^{n_J} \omega_k J_k[\boldsymbol{u}]$$

This yields a two-level optimization problem:

- "Outer" problem: Finite-dimensional, nonlinear optimization problem: $\Phi(\boldsymbol{\omega}) = \sum_{k=1}^{n_m} \|\boldsymbol{x}^*(t_k, \boldsymbol{\omega}) - \boldsymbol{x}_k\|_2^2$
- "Inner" problem: Constraints of the outer problem; Optimal control problem with solution $\boldsymbol{x}^*$

$$\begin{aligned} \min_{\boldsymbol{u}} J^*[\boldsymbol{u}], \quad J^*[\boldsymbol{u}] &= \sum_{k=1}^{n_J} \omega_k J_k[\boldsymbol{u}] \\ \text{subject to} \quad \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) \end{aligned}$$

11.2 Differential/Dynamic Games

Differential games have ODEs for the complete state $\boldsymbol{x}$ containing multiple ODEs for every player and controls $\boldsymbol{v}_1(t), \cdots, \boldsymbol{v}_{n_p}(t)$ for every placer. Additionally they might have state or control constraints. The players then might minimize or maximize one or more objective cooperative or non-cooperative.

11.2.1 Non-Cooperative Two-Player Zero-Sum Differential Games

A common class of differential games are *non-cooperative two-player zero-sum differential games* where:

- *Non-cooperative* means that one player tries to maximize the objective and the other tries to minimize it.
- *Zero-sum* means that the loss of one player is the reward of the other and vice versa.

Necessary conditions for these games can be derived from the minimax principle similar to the maximum principle using the ODEs for both players, adjunct differential equations and a Hamiltonian.

This also yields a multi-point BVP that can be solved numerically using indirect methods and it is also possible to use direct methods.

These games can be used, e.g. to generate robust trajectories by letting one player "be the noise" or friction or similar that tries to destroy the optimal trajectory.

Example

11.3 Learning Methods and Optimization

Learning methods (from machine learning) often rely on formulations of solving specific optimization problems. Hence, machine learning and optimization is highly coupled. The focus is a little bit different as most optimization methods try to find solutions as accurate as possible while ML algorithms try to generalize as best as possible to handle unseen scenarios.

11.3.1 Foundations

The underlying task often is to find a model $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$ that has a specific input/output behavior that is enforced using a *loss function*, e.g. the least squares loss function. These models are then optimized using basic gradient methods, gradient descent, without step size determination but using a small step size or an adaptive one (e.g. adam, adagrad, adadelta). A common function approximator are neural networks which use multiple *layers*, *activation functions* and *weights* to approximate arbitrary functions (in fact, any function can be approximated using a two-layer neural net).

Some open questions are:

- Why and when does gradient descent work and how fast?
- Why does not training not always cause overfitting (even when the number of parameters are much higher than the number of samples)?
- How to interpret the trained models (explainable AI)?

11.3.2 Reinforcement Learning

Reinforcement learning is highly related to optimal control in terms if a reward that is maximized by tweaking controls etc. However, optimal control is often discrete and models are often treated as stochastic models allowing reasoning about uncertainty. It is also possible to not have any model (e.g. in model-free reinforcement learning)! The model is then learned implicitly.

Reward

In the RL setting, it is possible to study both finite and infinite time horizons using discounted rewards ensuring that the sum of rewards converges.

Value Function

In RL, a policy π is searched which is commonly dependent on the state, not time: $\pi = \pi(\boldsymbol{x})$.

The *value function* describes the discounted reward starting from the current state:

$$V^{\pi}(\boldsymbol{x}_0) = \sum_{k=0}^{\infty} \gamma^k r_k\big(\boldsymbol{x}_k, \pi(\boldsymbol{x}_k)\big)$$

The *state-action function* describes the discounted reward of the current state if a specific action is taken next and following the policy afterwards:

$$Q^{\pi}(\boldsymbol{x}_0, \boldsymbol{u}_0) = r_1(\boldsymbol{x}_0, \boldsymbol{u}_0) + \sum_{min}^{max}$$

First Approach

A first approach would be to learn the control directly by maximizing the reward J using gradient descent:

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} J = \frac{\partial J}{\partial \pi_{\boldsymbol{\theta}}} \frac{\partial \pi_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$$

But this requires a gradient of the objective… The approximation may have high variances. Additionally, new gradient is computed independently of old approximations. Hence, no learning happens.

Learning the Value Function

It is better to solve the optimization problem

$$\boldsymbol{u}^*(\boldsymbol{x}) = \arg\max_{\boldsymbol{u}} Q^{\pi}(\boldsymbol{x}, \boldsymbol{u})$$

by approximating the value function $V(\boldsymbol{x}_0)$ with a function $V_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$, e.g. with *Temporal Difference Learning* (TD).

This uses the Bellman equation

$$V(\boldsymbol{x}) = \max_{\boldsymbol{u}} r(\boldsymbol{x}, \boldsymbol{u}) + \gamma V(\boldsymbol{x}, \boldsymbol{u}) = \max_{\boldsymbol{u}} Q(\boldsymbol{x}, \boldsymbol{u})$$

which is a discretized version of the H-J-B equation. Temporal difference learning measures the TD-error $\delta_n = r(\boldsymbol{x}, \boldsymbol{u}) + V_{\boldsymbol{\theta}}(\boldsymbol{x}_n) - \gamma V_{\boldsymbol{\theta}}(\boldsymbol{x}_{n+1})$ and uses gradient descent to update the parameters $\boldsymbol{\theta}$. An instance of this class of algorithms is Q-Learning.

- Advantages:
  - Small variance in the approximations of the expected reward.
- Disadvantages:
  - In every state, an optimization problem has to be solved. The overhead can be reduced by discretizing the control and just trying out all possibilities.
  - No guarantee of convergence (but this holds for every algorithm proposed in this course as it may guarantee convergence but convergence to a poor local minimum).

Actor-Critic

Another approach is to approximate the control and the value function at the same time:

- The *actor* generates control $\boldsymbol{u}$ for a given state $\boldsymbol{x}$ and
- the *critic* estimates the quality of the current control and learns the value functions that is used to tweak the control parameters.

This method combined the advantages of the previous approaches as it does not require to solve an optimization problem in each step and no control discretization. It also allows a more precise measurement of the gradients.

Notes

- All proposed methods are based on an approximation of the value function or the control. In practice, neural networks are often used for these approximations.
- Stability proves have been made for systems with special structures (e.g. affine dynamics).
- Better convergence than actor-only methods.
- The approach can be extended to models with continuous time.