Numerical Optimal Control

Lecturer: Prof. Moritz Diehl. Notes by: Yunlong Song

The great watershed in optimization is not between linearity and nonlinearity, but convexity and non-convexity. – R. Tyrrell Rockafellar

Introduction

There are two pillars of optimal control:

- **Dynamic Systems**
- **Optimization**

Dynamic Systems Classes

Continuous vs Discrete Time Systems:

- *Continuous-time System:*

$$\dot{x}(t) = f(x(t), u(t))$$
$$y(t) = h(x(t), u(t))$$

where $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ is the control input, and $y(t) \in \mathbb{R}^p$ is the output.

- *Discrete-time System:*

$$x_{k+1} = f(x_k, u_k)$$
$$y_k = h(x_k, u_k)$$

where $x_k \in \mathbb{R}^n$ is the state, $u_k \in \mathbb{R}^m$ is the control input, and $y_k \in \mathbb{R}^p$ is the output.

Continuous vs Discrete State Space:

- *Continuous State Space*
- *Discrete State Space*

Linear vs Nonlinear Systems:

- *Linear System:*

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, and $D \in \mathbb{R}^{p\times m}$.

- *Nonlinear System:*

$$\dot{x}(t) = f(x(t), u(t))$$
$$y(t) = h(x(t), u(t))$$

Time-Invariant vs Time-Varient Systems:

- *Time-Invariant System:*

$$\text{Nonlinear:} \quad \dot{x}(t) = f(x(t), u(t))$$
$$\text{Linear:} \quad \dot{x}(t) = Ax(t) + Bu(t)$$

- *Time-Varient System:*

$$\text{Nonlinear:} \quad \dot{x}(t) = f(x(t), u(t), t)$$
$$\text{Linear:} \quad \dot{x}(t) = A(t)x(t) + B(t)u(t)$$

Stable vs Unstable Systems:

- *Stable System:* A dynamic system whose state trajectory remains bounded for bounded initial values and control inputs.
- *Unstable System:* Otherwise.

Note: For LTI, stability is determined by the eigenvalues of the system matrix A.

Deterministic vs Stochastic Systems:

- *Deterministic System:* The state trajectory is uniquely determined by the initial state and control input.
- *Stochastic System:* The state trajectory is determined by the initial state, control input, and random disturbances.

Continuous Time Systems

A continuous-time system is described by a set of ordinary differential equations (ODEs) on a time interval $t \in [t_0, T]$:

$$\dot{x}(t) = f(x(t), u(t), t), t \in [t_0, T]$$

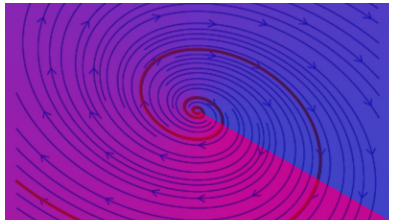

Figure 1: Continuous Time System

Initial Value Problem: An Initial Value Problem (IVP) is a type of differential equation problem that includes finding a solution to the equation that satisfies specified initial conditions. Want Unique Solution.

Zero-Order Hold: In the age of digital control, the inputs u are often generated by a computer and implemented at the physical system as piecewise constant between two sampling instants. This is called *zero-order hold*.

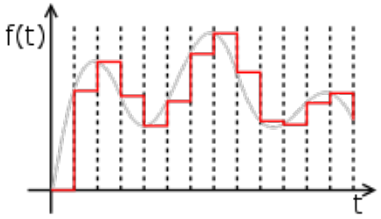

Figure 2: Zero-Order Hold

Numerical Integration:

- *Explicit Methods:* Euler, Runge-Kutta
- *Implicit Methods:* Backward Euler, Trapezoidal Rule

Discrete Time Systems

A discrete-time system is described by a set of difference equations on a time interval $k \in [0, N]$:

$$x_{k+1} = f(x_k, u_k), k \in [0, N]$$

Linear Time-Invariant System:

$$x_{k+1} = Ax_k + Bu_k$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$. An LTI system is stable if all eigenvalues of A are inside the unit circle of the complex plane.

The forward simulation of a discrete-time system is straightforward:

$$f_{\text{sim}}(x_0; u_0, \ldots, u_{N-1}) = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} x_0 \\ Ax_0 + Bu_0 \\ \vdots \\ A^N x_0 + \sum_{i=0}^{N-1} A^{N-i-1} B u_i \end{bmatrix}$$

Affine Systems and Linearizations along Trajectories: An important generalization of linear systems are affine time-varying systems of the form:

$$x_{k+1} = A_k x_k + B_k u_k + c_k, k \in [0, N-1]$$

Optimization

Finite vs Infinite Dimensional Optimization:

- *Finite Dimensional Optimization:* The optimization variables are finite-dimensional vectors.
- *Infinite Dimensional Optimization:* The optimization variables are functions.
- *Semi-Infinite Dimensional Optimization:* The optimization variables are finite-dimensional vectors and functions.

Continuous vs Integer Optimization:

Continuous Optimization: The optimization variables are continuous. For example, linear programming, quadratic programming, and nonlinear programming.
Nonlinear programming has the form:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \le 0, i = 1,\dots,m \\ & h_j(x) = 0, j = 1,\dots,p \end{aligned}$$

Integer Optimization: The optimization variables are integers. For example, mixed-integer linear programming (MILP) and mixed-integer nonlinear programming (MINLP).
Mixed-integer nonlinear programming has the form:

$$\begin{aligned} \min_{x\in\mathbb{R}^n, z\in\mathbb{Z}^m} \quad & f(x,z) \\ \text{s.t.} \quad & g_i(x,z) \le 0, i = 1,\dots,m \\ & h_j(x,z) = 0, j = 1,\dots,p \end{aligned}$$

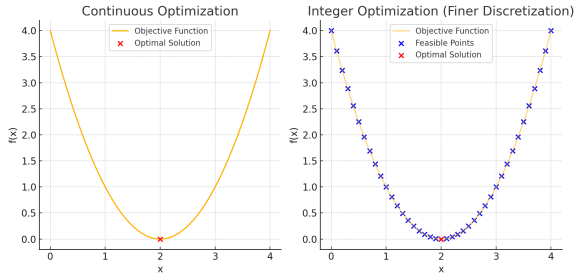

Figure 3: Continuous vs Integer Optimization

Convex vs Nonconvex Optimization:

- *Convex Optimization:* The objective function and constraints are convex. For example, linear programming (LP), quadratic programming (QP), and semidefinite programming.
- *Nonconvex Optimization:* Otherwise

Root-Finding: Newton-Type Methods

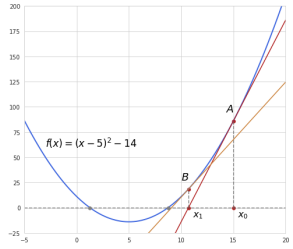

Figure 4: Newton's Method for root finding

Newton's Method: Let's consider a continuous function $f : \mathbb{R}^n \to \mathbb{R}^n$, where our aim is to solve the nonlinear equation $f(x) = 0$. Newton's idea was to start with an initial guess x_0 and iteratively improve it by solving the linear system:

$$f(x_k) + \frac{\partial f}{\partial x}(x_k)(x_{k+1} - x_k) = 0$$

which leads to the update rule:

$$x_{k+1} = x_k - \left(\frac{\partial f}{\partial x}(x_k)\right)^{-1} f(x_k)$$

Local Convergence Rates

- *Q-linearly*
- *Q-superlinearly*
- *Q-quadratically*

Affine Invariance
An iterative method to solve a root-finding problem is called affine invariant if the method is invariant under affine transformations of the function f.
Newton's method is affine invariant, which means that if

$$f(x) = 0 \quad \text{and} \quad y = Ax + b$$

then the solution of $f(y) = 0$ is $y = Ax + b$.

Nonlinear Optimization

The general form of a nonlinear optimization problem is:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \le 0, i = 1,\dots,m \\ & h_j(x) = 0, j = 1,\dots,p \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g_i : \mathbb{R}^n \to \mathbb{R}$, and $h_j : \mathbb{R}^n \to \mathbb{R}$ are continuously differentiable functions.
Feasible Set: The set of all feasible solutions is called the feasible set:

$$\mathcal{F} = \{x \in \mathbb{R}^n \mid g_i(x) \le 0, h_j(x) = 0\}$$

Global Minimum: A point $x^* \in \mathbb{R}^n$ is a global minimum if and only if $x^* \in \mathcal{F}$ and $f(x^*) \le f(x)$ for all $x \in \mathcal{F}$. The value $f(x^*)$ is called the global minimum value.

Local Minimum: A point $x^* \in \mathbb{R}^n$ is a local minimum if there exists a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) \le f(x)$ for all $x \in \mathcal{N} \cap \mathcal{F}$. The value $f(x^*)$ is called the local minimum value.

Linear Programming (LP):

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & c^T x \\ \text{s.t.} \quad & Ax - b = 0 \\ & Cx - d \le 0 \end{aligned}$$

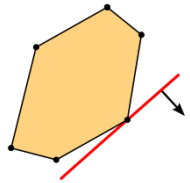

Figure 5: Linear Optimization

Quadratic Programming (QP):

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \frac{1}{2}x^T Hx + c^T x \\ \text{s.t.} \quad & Ax - b = 0 \\ & Cx - d \le 0 \end{aligned}$$

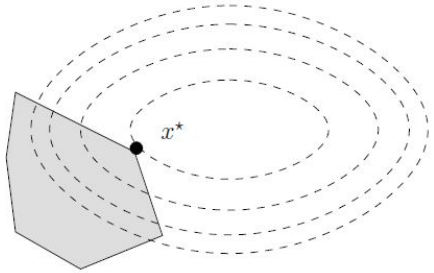

Figure 6: Quadratic Optimization

Convexity:

- *Convex Function:* A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if for all $x, y \in \mathbb{R}^n$ and $\lambda \in [0, 1]$:

$$f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda)f(y)$$

- *Convex Set:* A set $\mathcal{S} \subseteq \mathbb{R}^n$ is convex if for all $x, y \in \mathcal{S}$ and $\lambda \in [0, 1]$:

$$\lambda x + (1-\lambda)y \in \mathcal{S}$$

Convex Optimization: *Definition:* An optimization problem with convex feasible set and convex objective function is called a convex optimization problem.
Note that the feasible set of an optimization problem is convex if the function g_i is affine and the function h_j is convex.

First-Order Optimality Conditions: *KKT Conditions:* The Karush-Kuhn-Tucker (KKT) conditions are necessary conditions for a point x^* to be a local minimum of a nonlinear

optimization problem. The KKT conditions are:

$$\begin{aligned}\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x^*) + \sum_{j=1}^{p} \mu_j \nabla h_j(x^*) &= 0 \\ g_i(x^*) &\leq 0, i = 1, \dots, m \\ h_j(x^*) &= 0, j = 1, \dots, p \\ \lambda_i &\geq 0, i = 1, \dots, m \\ \lambda_i g_i(x^*) &= 0, i = 1, \dots, m\end{aligned}$$

KKT is a First-Order Necessary Condition (FONC) for optimality. It is not sufficient in general.

Langrange Function: The Lagrange function is defined as:

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{p} \mu_j h_j(x)$$

Second-Order Optimality Conditions: *Second-Order Sufficient Conditions:* The Second-Order Sufficient Conditions (SOSC) are sufficient conditions for a point x^* to be a local minimum of a nonlinear optimization problem. The SOSC are:

$$\begin{aligned}\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x^*) + \sum_{j=1}^{p} \mu_j \nabla h_j(x^*) &= 0 \\ \nabla^2 \mathcal{L}(x^*, \lambda, \mu) &\succeq 0 \\ g_i(x^*) &\leq 0, i = 1, \dots, m \\ h_j(x^*) &= 0, j = 1, \dots, p \\ \lambda_i &\geq 0, i = 1, \dots, m \\ \lambda_i g_i(x^*) &= 0, i = 1, \dots, m\end{aligned}$$

Newton-type Optimization Algorithms

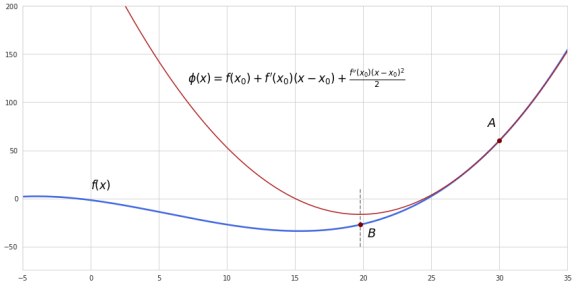

Figure 7: Newton-type Optimization

Equality Constrained Optimization: The general form of an equality constrained optimization problem is:

$$\begin{aligned}\min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & h(x) = 0\end{aligned}$$

The idea of the Newton-type optimization algorithm is to iteratively solve the nonlinear KKT Conditions:

$$\begin{aligned}\nabla \mathcal{L}(x, \lambda) = \nabla f(x) + \lambda \nabla h(x) &= 0 \\ h(x) &= 0\end{aligned}$$

In order to simplfy the notation, we define:

$$z := \begin{bmatrix} x \\ \lambda \end{bmatrix} \quad \text{and} \quad F(z) := \begin{bmatrix} \nabla f(x) + \nabla h(x)^T \lambda \\ h(x) \end{bmatrix} = \begin{bmatrix} \nabla \mathcal{L}(x, \lambda) \\ h(x) \end{bmatrix}$$

Starting from an initial guess z_0, Newton's method generates a sequence of iterates $\{z_k\}_{k=0}^{\infty}$ by linearizing the nonlinear equation at the current iterate:

$$\begin{aligned}F(z_k) + \nabla_{z_k} F(z_k)(z_{k+1} - z_k) &= 0 \\ \rightarrow z_{k+1} = z_k - \nabla_{z_k} F(z_k)^{-1} F(z_k)\end{aligned}$$

1) Exact Newton Step:
For equality constrained optimization, the update rule becomes:

$$\begin{aligned}& F(z_k) + \nabla_{z_k} F(z_k)(z_{k+1} - z_k) = 0 \\ \Rightarrow & \begin{bmatrix} \nabla_x \mathcal{L}(x_k, \lambda_k) \\ h(x_k) \end{bmatrix} + \underbrace{\begin{bmatrix} \nabla_x^2 \mathcal{L}(x_k, \lambda_k) & \nabla h(x_k) \\ \nabla h(x_k)^T & 0 \end{bmatrix}}_{\text{KKT-Matrix}} \begin{bmatrix} x - x_k \\ \lambda \end{bmatrix} = 0 \\ \Rightarrow & \begin{bmatrix} \nabla f(x_k) \\ h(x_k) \end{bmatrix} + \begin{bmatrix} \nabla_x^2 \mathcal{L}(x_k, \lambda_k) & \nabla h(x_k) \\ \nabla h(x_k)^T & 0 \end{bmatrix} \begin{bmatrix} x - x_k \\ \lambda \end{bmatrix} = 0\end{aligned}$$

This formulation shows that the data of the linear system only depend on λ_k via the Hessian matrix.

Newton-type Iteration: We can approximate the Hessian matrix using different methods. $\mathbf{B}_k \approx \nabla_x^2 \mathcal{L}(x_k, \lambda_k)$, we get:

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ 0 \end{bmatrix} - \begin{bmatrix} \mathbf{B}_k & \nabla h(x_k) \\ \nabla h(x_k)^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \nabla f(x_k) \\ h(x_k) \end{bmatrix}$$

Quadratic Model Interpretation: The Newton-type iteration can be interpreted as minimizing a quadratic model of the Lagrange function:

$$\begin{aligned}\min_{x \in \mathbb{R}^n} \quad & \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T \mathbf{B}_k (x - x_k) \\ \text{s.t.} \quad & h(x_k) + \nabla h(x_k)^T (x - x_k) = 0\end{aligned}$$

Hence, we can interpret the Newton-type optimization as a *Sequential Quadratic Programming* (SQP) method.

The Exact Newton Step: The first and obvious choice for the Hessian matrix $\mathbf{B}_k$ is the exact Hessian of the Lagrange function:

$$\mathbf{B}_k = \nabla_x^2 \mathcal{L}(x_k, \lambda_k)$$

Local Convergence Rate: The exact Newton method has a quadratic convergence rate in a neighbourhood of the optimal solution z^*. As a rule of thumb, once a Newton method is in its area of quadratic convergence, it needs at maximum **6 iterations** to reach the highest possible accuracy.

2) Constrained Gauss-Newton Method:
Let's consider the nonlinear least squares problem:

$$\begin{aligned}\min_{x \in \mathbb{R}^n} \quad & \frac{1}{2}\|f(x)\|^2 \\ \text{s.t.} \quad & h(x) = 0\end{aligned}$$

The idea of Gauss-Newton is to linearize at a given iterate x_k both the objective function and the constraints:

$$\begin{aligned}f(x) &\approx f(x_k) + \nabla f(x_k)(x - x_k) \\ h(x) &\approx h(x_k) + \nabla h(x_k)(x - x_k)\end{aligned}$$

Hence, we have an aproximation of the original problem:

$$\begin{aligned}\min_{x \in \mathbb{R}^n} \quad & \frac{1}{2}\|f(x_k) + \nabla f(x_k)^T (x - x_k)\|^2 \\ \text{s.t.} \quad & h(x_k) + \nabla h(x_k)(x - x_k) = 0\end{aligned}$$

This is a convex QP which can be easily be seen by noting that the objective is equal to:

$$\begin{aligned}&\frac{1}{2} f(x_k)^T f(x_k) + (x - x_k)^T \underbrace{\nabla f(x_k) f(x_k)}_{=\nabla f(x_k)} + \\ &\underbrace{\frac{1}{2}(x - x_k)^T \nabla f(x_k)^T \nabla f(x_k)(x - x_k)}_{=:\mathbf{B}_k}\end{aligned}$$

Thus, the Gauss-Newton is identical to the SQP problem with a special choice of the Hessian approximation, namely:

$$\mathbf{B}_k = \nabla f(x_k)^T \nabla f(x_k) = \sum_{i=1}^{m} \nabla f_i(x_k)^T \nabla f_i(x_k)$$

Local Convergence Rate: The Gauss-Newton method converges linearly, $z_{k+1} - z^* \leq k\|z_k - z^*\|$ with a contraction rate $k = O(\|f(z^*)\|)$.

3) Quasi-Newton BFGS:
Quasi-Newton Hessian update methods use the previous Hessian approximation $\mathbf{B}_k$, the step $s_k = x_{k+1} - x_k$, and the gradient difference $y_k = \nabla_x \mathcal{L}(x_{k+1}, \lambda_{k+1}) - \nabla_x \mathcal{L}(x_k, \lambda_k)$ to obtain a new Hessian approximation $\mathbf{B}_{k+1}$. The BFGS update formula is:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{\mathbf{B}_k s_k s_k^T \mathbf{B}_k}{s_k^T \mathbf{B}_k s_k}$$

Local Convergence Rate: The BFGS method has a superlinear convergence rate in a neighbourhood of the optimal solution z^*.

Inequality Constrained Optimization:
The general form of an inequality constrained optimization problem is:

$$\begin{aligned}\min_{x \in \mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & h(x) = 0 \\ & g(x) \leq 0\end{aligned}$$

Two big families of methods exist to solve inequality constrained optimization problems:

- *Interior Point Methods*

- *Sequential Quadratic Programming*
- *Active Set Methods*

1) Interior Point Methods:
The basic idea of interior point methods is to replace the non-smooth L-shaped set resulting from the complementarity conditions with a smooth approximation, typically a hyperbola. Thus, a smoothing constant $\tau \geq 0$ is introduced and the KKT conditions are replaced by the smooth equation system:

$$\begin{aligned} \nabla \mathcal{L}(x, \lambda, \mu) &= \nabla f(x) + \lambda \nabla h(x) + \mu \nabla g(x) = 0 \\ h(x) &= 0 \\ g(x) &\leq 0 \\ \mu &\geq 0 \\ \mu_i g_i(x) + \tau_i &= 0, i = 1, \ldots, n_g \end{aligned}$$

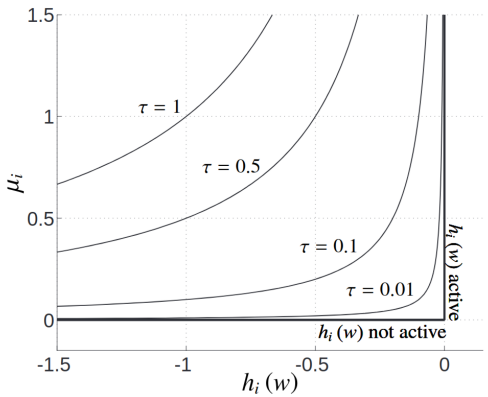

Figure 8: Interior Point Method. The L-shaped set is replaced by a smooth approximation. [$h_i(w)$ corresponds to $g_i(x)$.]

We usually start with a large value of τ and solve the resulting nonlinear equation system by a Newton-type method, and then, iteratively decrease τ always using the previously obtained solution as initialization for the next one.

Barrier Function: One way to interpret the above smoothened KKT-conditions is to use the last condition to eliminate:

$$\mu_i = -\frac{\tau_i}{g_i(x)}$$

and insert this into the Lagrange function. This results in a barrier function $\phi(x, \tau)$ which is a smooth approximation of the original problem.
Note that

$$\nabla(\log(-g(x))) = \frac{\nabla g(x)}{g(x)}$$

Thus, the above smooth form of the KKT conditions is nothing else than the optimality conditions of *equality constrained optimization* of the barrier function.

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & f(x) - \tau \sum_{i=1}^{n_g} \log(-g_i(x)) \\ \text{s.t.} \quad & h(x) = 0 \end{aligned}$$

2) Sequential Quadratic Programming:
The idea of Sequential Quadratic Programming (SQP) is to approximate the original optimization problem by a sequence of QPs. The QP is obtained by linearizing the objective function and the constraints at the current iterate x_k:

$$\begin{aligned} f(x) &\approx f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T \mathbf{B}_k (x - x_k) \\ h(x) &\approx h(x_k) + \nabla h(x_k)^T (x - x_k) \\ g(x) &\approx g(x_k) + \nabla g(x_k)^T (x - x_k) \end{aligned}$$

which results in the QP:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T \mathbf{B}_k (x - x_k) \\ \text{s.t.} \quad & h(x_k) + \nabla h(x_k)^T (x - x_k) = 0 \\ & g(x_k) + \nabla g(x_k)^T (x - x_k) \leq 0 \end{aligned}$$

3) Active Set Methods:
If we would know the active set, then we could solve directly an equality constrained optimization problem. The main task is thus to finding the corrective active set. A major advantage of active set strategies is that they can very efficiently be warm-started under circumstances where a series of similar optimization problems have to be solved, e.g., within an SQP method, or in the context of model predictive control.

Calculating Derivatives

Automatic Differentiation:
Automatic differentiation exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, derivatives of arbitrary order can be computed automatically, accurately to working precision, and using at most a small constant factor more arithmetic operations than the original program.

Forward Mode:

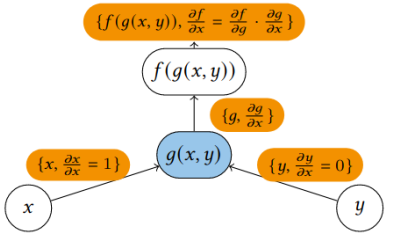

Figure 9: AD Forward Mode

Reverse Mode:

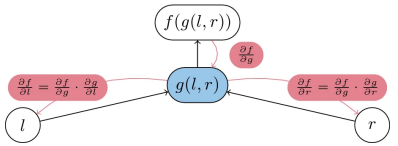

Figure 10: AD Reverse Mode

Discrete Optimal Control

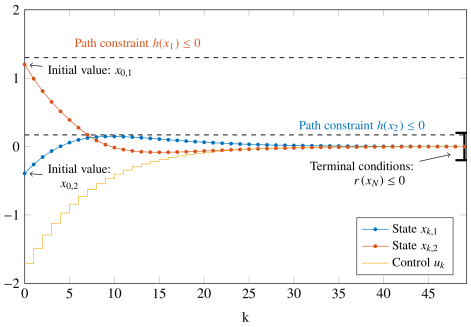

Figure 11: Discrete Optimal Control

Optimal Control Problem (OCP):
Given the system model and constraints, a generic discrete-time optimal control problem can be formulated as the following constrained optimization problem:

$$\begin{aligned} \min_{x_0, \ldots, x_N, u_0, \ldots, u_{N-1}} \quad & \sum_{k=0}^{N-1} l(x_k, u_k) + E(x_N) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k), k = 0, \ldots, N-1 \\ & g(x_k, u_k) \leq 0, k = 0, \ldots, N-1 \\ & r(x_0, x_N) = 0 \end{aligned}$$

where $x_k \in \mathbb{R}^{n_x}$ is the state, $u_k \in \mathbb{R}^{n_u}$ is the control input, $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ is the stage cost, $E : \mathbb{R}^{n_x} \to \mathbb{R}$ is

the terminal cost, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is the system dynamics, $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_g}$ are the path constraints, and $r : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_r}$ are the boundary constraints.
The optimal control problem is solved by finding the optimal state and control trajectories $\{x_k^*, u_k^*\}_{k=0}^{N-1}$ that minimize the cost function while satisfying the system dynamics and constraints.

Broadly speaking, there are two main approaches to solve OCPs:

- *Simultaneous Approaches* solve the OCP by formulating the optimization problem as a single large optimization problem, e.g., direct multiple shooting and direct collocation.
- *Sequential Approaches* solve the OCP by iteratively solving a sequence of smaller optimization problems, e.g., indirect methods and direct single shooting.

Dynamic Programming

Dynamic programming is easist to apply to systems with discrete states and control spaces. DP suffers from the **curse of dimensionality**, which makes it computationally infeasible for high-dimensional systems. On the positive side, DP can easily deal with all kinds of hybrid systems or non-differentiable dynamics.

1) Dynamic Programming for Discrete Systems:
Let's define an optimal control probelm for discrete systems:

$$\begin{aligned} \min_{x_0,\dots,x_N,u_0,\dots,u_{N-1}} \quad & \sum_{k=0}^{N-1} l(x_k, u_k) + E(x_N) \\ \text{s.t.} \quad & f(x_k, u_k) - x_{k+1} = 0, k = 0, \dots, N-1 \\ & \bar{x}_0 - x_0 = 0 \end{aligned}$$

Cost-to-go Function: The cost-to-go function $J_k(x_k)$ is the minimum cost that can be obtained from state x_k to the terminal state x_N:

$$J_k(x_k) = \min_{u_k,\dots,u_{N-1}} \sum_{i=k}^{N-1} l(x_i, u_i) + E(x_N)$$

Recusion Step A recursive equation that relates the cost-to-go function at time k to the cost-to-go function at time $k+1$:

$$J_k(x_k) = \min_u l(x_k, u) + J_{k+1}(f(x_k, u))$$

Optimal feedback control: The optimal control input u_k^* is obtained by:

$$u_k^* = \arg\min_u l(x_k, u) + J_{k+1}(f(x_k, u))$$

Example: Linear Quadratic Problem:
Let's consider the linear quadratic optimal control problem:

$$\begin{aligned} \min_{x,u} \quad & \sum_{k=0}^{N-1} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} Q_k & S_k \\ S_k^T & R_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + x_N^T P_N x_N \\ \text{s.t.} \quad & x_{k+1} - A_k x_k - B_k u_k = 0, k = 0, \dots, N-1 \\ & x_0 - \bar{x}_0 = 0 \end{aligned}$$

step 1: $J_N(x_N) = x_N^T P_N x_N$

step 2:

$$\begin{aligned} J_k(x_k) &= \min_{u_k} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} Q_k & S_k \\ S_k^T & R_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} + J_{k+1}(f(x_k, u_k)) \\ &= \min_{u_k} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T \begin{bmatrix} Q_k & S_k \\ S_k^T & R_k \end{bmatrix} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \\ &\quad + \begin{bmatrix} x_k \\ u_k \end{bmatrix}^T [A|B]^T P [A|B] \begin{bmatrix} x_k \\ u_k \end{bmatrix} \\ &= x^T P_{\text{new}} x \\ \text{with} \quad & P_{\text{new}} = Q_k + A^T P A - \cdots \end{aligned}$$

step 3: $u_k^* = -K_k x_k$

2) Infinite Horizon Problems:
Infinite horizon problems are defined:

$$\begin{aligned} \min_{x,u} \quad & \sum_{k=0}^{\infty} l(x_k, u_k) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k), k = 0, 1, \dots \\ & \bar{x}_0 - x_0 = 0 \end{aligned}$$

Interestingly, the cost-to-go function $J(x_k)$ is independent of time k. This directly leads to the Bellman equation:

$$J(x) = \min_u \underbrace{l(x, u) + J(f(x, u))}_{=\bar{J}(x,u)}$$

The optimal controls are obtained by the Function:

$$u^* = \arg\min_u \bar{J}(x, u)$$

> **Bellman's Principle of Optimality**
>
> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

3) The gradient of the Value Function:

The gradients of the value function at the optimal trajectory have to satisfy the following equation:

$$\nabla J(x_k) = \nabla_x l(x_k, u_k) + \frac{\partial f}{\partial x} f(x_k, u_k)^T \nabla J(x_{k+1})$$

This recursive condition on the gradients is equivalent to the first order necessary conditions for optimality (FONC) that we obtained previously for differentiable optimal control problems, if we identify the gradients with the multiplier λ_k, e.g., set

$$\lambda_k = \nabla J_k(x_k).$$

This is a very important interpretation of the multipliers λ_k; they are nothing else than the gradients of the value function at the optimal trajectory!

4) Differential Dynamic Programming:
Let's start from the Bellman equation:

$$J(x) = \min_u l(x, u) + J(f(x, u))$$

Let Q be the variation of this quantity around the i-th (x, u) pair:

$$\begin{aligned} Q(\delta_x, \delta_u) = & \, l(x + \delta_x, u + \delta_u) + J(f(x + \delta_x, u + \delta_u)) \\ & - l(x, u) - J(f(x, u)) \end{aligned}$$

The second-order Taylor expansion of Q is:

$$Q(\delta_x, \delta_u) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta_x \\ \delta_u \end{bmatrix}^T \begin{bmatrix} 0 & Q_x^T & Q_u^T \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta_x \\ \delta_u \end{bmatrix}$$

The expansion coefficients are:

$$\begin{aligned} Q_x &= l_x + f_x^T J_x^{'} \\ Q_u &= l_u + f_u^T J_x^{'} \\ Q_{xx} &= l_{xx} + f_x^T J_{xx}^{'} f_x + V_x^{'} f_{xx} \\ Q_{xu} &= l_{xu} + f_x^T J_{xx}^{'} f_u + V_x^{'} f_{uu} \\ Q_{uu} &= l_{ux} + f_u^T J_{xx}^{'} f_x + V_x^{'} f_{ux} \end{aligned}$$

Backward Pass: Minimalization of Q with respect to δ_u:

$$\delta_u^* = \arg\min_{\delta_u} Q(\delta_x, \delta_u) = -Q_{uu}^{-1}(Q_u + Q_{ux}\delta_x)$$

By initializing the backward pass with $V(x, N) = E(x_N)$, we can solve the dynamic programming problem by iteratively solving the backward pass. At $t = N$, we can compute all the Q terms. Also, $V(\delta_x) = \min_{\delta_u} Q(\delta_x, \delta_u) = Q(\delta_x, \delta_u^*)$.

Forward Pass: A forward pass computes a new trajectory τ^*:

$$\begin{aligned} x_0^* &= \hat{x}_0 \\ u_k^* &= -Q_{uu}^{-1}(Q_u + Q_{ux}(x_k^* - \hat{x}_k)) \\ x_{k+1}^* &= f(x_k^*, u_k^*) \end{aligned}$$

5) Iterative LQR:
The iterative LQR algorithm is a variant of the differential dynamic programming algorithm. iLQR is nearly identical to the generic DDP presented earlier, with one exception: we use a linear approximation to the dynamics model rather than an quadratic approximation. This approximation results in the cancellation of a few terms in the 2nd order expansion coefficients for our Q-function derivatives. Notice that the f_{xx}, f_{uu}, f_{ux} terms cancel to 0 because of this linear approximation.

Continuous Time Optimal Control

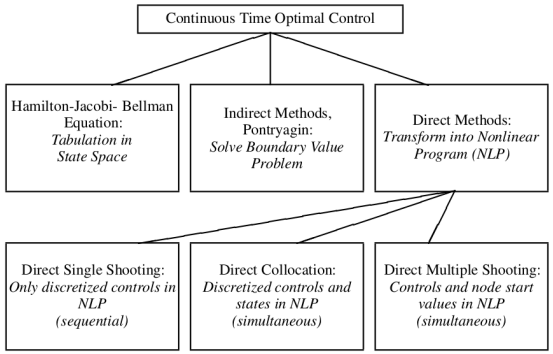

Figure 12: The optimal control family tree.

The Hamilton-Jacobi-Bellman (HJB) equation is a partial differential equation (PDE) that describes the optimal cost-to-go function for a continuous-time optimal control problem. The HJB equation is defined as:

$$\begin{aligned} \min_{x(\cdot),u(\cdot)} \quad & \int_0^T l(x(t),u(t))dt + E(x(T)) \\ \text{s.t.} \quad & \dot{x}(t) = f(x(t),u(t)), t \in [0,T], \quad \text{(ODE Model)} \\ & x(0) = \bar{x}_0 \quad \text{(Initial Condition)} \end{aligned}$$

Value Function: The value function $V(x,t)$ is the minimum cost-to-go function for the continuous-time optimal control problem:

$$V(x,t) = \min_{u(\cdot)} \left\{ \int_t^T l(x(\tau),u(\tau))d\tau + E(x(T)) \right\}$$

The HJB equation provides a way to compute the value function. It is a partial differential equation (PDE) that relates the value function to the system dynamics and the cost function:

Hamilton-Jacobi-Bellman Equation

$$\begin{aligned} \frac{\partial V(x,t)}{\partial t} &= -\min_u \left\{ l(x,u,t) + \frac{\partial V(x,t)}{\partial x} f(x,u,t) \right\} \\ V(x,T) &= E(x) \end{aligned}$$

Optimal Control: The optimal control input $u^*(x,t)$ is obtained by minimizing the right-hand side of the HJB equation:

$$\begin{aligned} u^*(x,t) &= \arg\min_u \left\{ l(x,u,t) + \frac{\partial V(x,t)}{\partial x} f(x,u,t) \right\} \\ &= \arg\min_u \left\{ \underbrace{l(x,u,t) + \lambda^T f(x,u,t)}_{:=H(x,\lambda,u)} \right\} \end{aligned}$$

where $H(x,\lambda,u)$ is the Hamiltonian function, and $\lambda = \frac{\partial V(x,t)}{\partial x}$ is the costate or adjoint variable.

Pontryagin's Minimum Principle

Pontryagin Minimum Principle

The PMP provides necessary conditions for optimality of a continuous-time optimal control problem. The PMP states that the optimal control input $u^*(t)$ is a minimizer of the Hamiltonian function $H(x(t),\lambda(t),u(t))$.

$$u^*(t) = \arg\min_u H(x(t),\lambda(t),u(t))$$

Apply the PMP:

- *Step 1:* Compute the Hamiltonian function $H(x,\lambda,u)$.
- *Step 2:* Solve for the state equation $\dot{x}(t) = f(x(t),u(t))$.
- *Step 3:* Solve for the costate equation $\dot{\lambda}(t) = -\frac{\partial H}{\partial x}$ with the terminal condition $\lambda(T) = \frac{\partial E}{\partial x}$.
- *Step 4:* Compute the optimal control input $u^*(t)$ by minimizing the Hamiltonian function $H(x(t),\lambda(t),u(t))$.

TPBVP and Numerical Solutions

Two-Point Boundary Value Problems (TPBVP):
A two-point boundary value problem (TPBVP) typically arises when using the Pontryagin Minimum Principle (PMP) and the Hamilton-Jacobi-Bellman (HJB) equation to find the optimal control policy. Here's how TPBVPs are connected to optimal control:

$$\begin{aligned} \text{Initial Condition:} \quad & x(t_0) = \bar{x}_0 \\ \text{Terminal Costate:} \quad & \lambda(T) = \frac{\partial E(x(T))}{\partial x} \\ \text{ODE Equation:} \quad & \dot{x}(t) = f(x(t),u(t)) \\ \text{Costate Equation:} \quad & \dot{\lambda}(t) = -\frac{\partial H(x,u,\lambda,t)}{\partial x} \end{aligned}$$

Single Shooting:

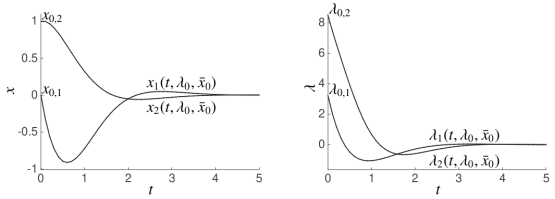

Figure 13: Single Shooting

Multiple Shooting:

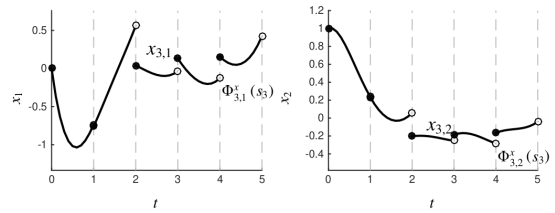

Figure 14: Multiple Shooting

Collocation:

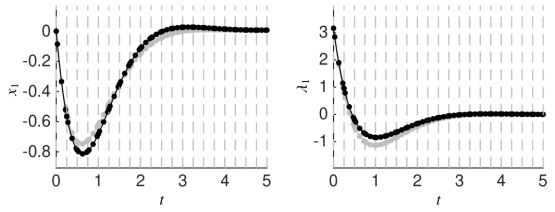

Figure 15: Collocation

Table 1: Summary of Numerical Methods for Solving TPBVPs in Optimal Control

Method	Description	Advantages	Disadvantages	Best Suited For
Single Shooting	• Solve TPBVP by guessing initial conditions and integrating forward in time. • Adjust initial guess iteratively to meet terminal boundary conditions.	• Simple to implement. • Low computational cost for simple problems.	• Sensitive to initial guess. • Can struggle with instability or stiffness in the problem.	• Problems with stable dynamics. • Simple or well-behaved problems.
Multiple Shooting	• Divide the interval into several subintervals. • Solve IVPs on each subinterval. • Ensure continuity at subinterval boundaries by imposing matching conditions.	• More robust to instability and stiffness compared to single shooting. • Better handling of complex dynamics.	• More complex to implement. • Higher computational cost due to solving larger system of equations.	• Problems with unstable or stiff dynamics. • Complex problems requiring robust solutions.
Collocation	• Approximate solution using basis functions (e.g., polynomials). • Enforce differential equations and boundary conditions at specific collocation points.	• High accuracy. • Effective for handling complex boundary conditions and nonlinearities.	• Complex implementation. • Computationally intensive, especially for higher-order methods.	• Problems requiring high accuracy. • Complex problems with significant nonlinearities and boundary conditions. • Large-scale problems where precision is crucial.

Direct Methods to Continuous OC

Direct methods to continuous optimal control problems discretize the control and state trajectories and solve the resulting finite-dimensional optimization problem.

Direct Single Shooting:

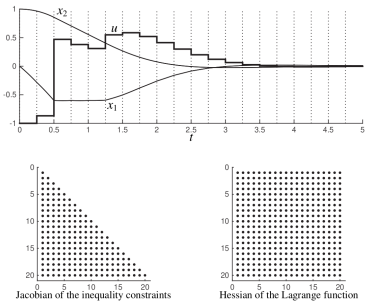

Figure 16: Direct Single Shooting

Direct Multiple Shooting:

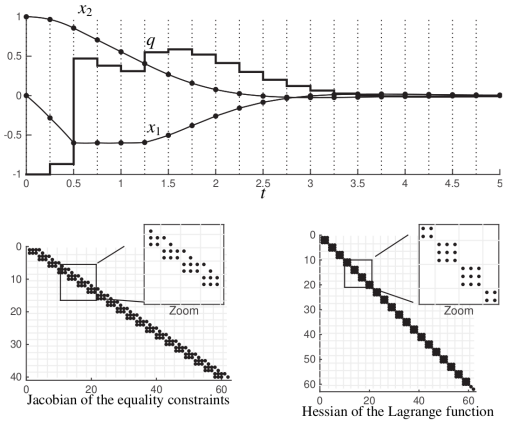

Figure 17: Direct Multiple Shooting

Direct Collocation:

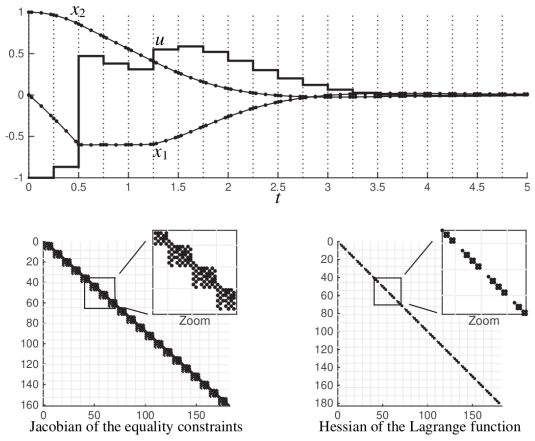

Figure 18: Direct Collocation

Table 2: Summary of Direct Methods for Solving Optimal Control Problems

Method	Description	Advantages	Disadvantages	Best Suited For
Direct Single Shooting	• Parametrize the control function over the entire interval. • Solve resulting nonlinear programming problem (NLP) by forward integrating the state equations.	• Simple to implement. • Suitable for problems with a small number of parameters.	• Sensitive to initial guess. • Can struggle with instability or stiffness.	• Problems with stable dynamics. • Problems with a small number of optimization parameters.
Direct Multiple Shooting	• Divide the interval into several subintervals. • Parametrize the control function and solve an NLP on each subinterval. • Ensure continuity at subinterval boundaries.	• More robust to instability and stiffness compared to single shooting. • Better handling of complex dynamics.	• More complex to implement. • Higher computational cost due to solving larger NLP.	• Problems with unstable or stiff dynamics. • Complex problems requiring robust solutions.
Direct Collocation	• Approximate state and control variables using basis functions (e.g., polynomials). • Discretize the optimal control problem into a large NLP. • Ensure differential equations and boundary conditions are satisfied at collocation points.	• High accuracy. • Effective for handling complex boundary conditions and nonlinearities.	• Complex implementation. • Computationally intensive, especially for large-scale problems.	• Problems requiring high accuracy. • Complex problems with significant nonlinearities and boundary conditions. • Large-scale problems where precision is crucial.

Nonlinear Model Predictive Control

Lets consider the following optimal control problem:

$$\begin{aligned} \min_{x,u,z} \quad & \sum_{k=0}^{N-1} l(x_k, u_k, z_k) + E(x_N) \\ \text{s.t.} \quad & x_{k+1} = f(x_k, u_k), k = 0, \dots, N-1 \\ & g(x_k, u_k, z_k) \leq 0, k = 0, \dots, N-1 \\ & h(x_k, u_k, z_k) = 0, k = 0, \dots, N-1 \\ & x_0 = \bar{x}_0 \\ & r(x_N) \leq 0 \end{aligned}$$

Here, $x_k \in \mathbb{R}^{n_x}$ is the state, $u_k \in \mathbb{R}^{n_u}$ is the control input, $z_k \in \mathbb{R}^{n_z}$ are the algebraic variables, $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_z} \to \mathbb{R}$ is the stage cost, $E : \mathbb{R}^{n_x} \to \mathbb{R}$ is the terminal cost, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is the system dynamics, $g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_z} \to \mathbb{R}^{n_g}$ are the path constraints, $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_z} \to \mathbb{R}^{n_h}$ are the equality constraints, and $r : \mathbb{R}^{n_x} \to \mathbb{R}^{n_r}$ are the terminal constraints.

The optimal control problem is solved by finding the optimal state, control, and algebraic trajectories $\{x_k^*, u_k^*, z_k^*\}_{k=0}^{N-1}$ that minimize the cost function while satisfying the system dynamics and constraints.

The NMPC problem is solved by iteratively solving a sequence of finite-horizon optimal control problems. At each time step, the current state estimate is used to initialize the optimization problem, and the resulting control input is applied to the system. The process is repeated at the next time step with an updated state estimate.

Real-Iime Optimization Strategies:

- *Offline Precomputation*
- *Delay Compensation by Prediction*
- *Warm-Starting*
- *Division into prepration and feedback phase*
- *Iterating while the problem changes*