Lecture 2: Numerical Optimization for Control

(grad/SQP/QP; ALM vs. interior-point vs. penalty)

Arnaud Deza

August 29, 2025

ISYE 8803: Special Topics on Optimal Control and Learning

Overview and Big Picture of Lecture 2

Learning goals (what you'll be able to do)

Goals for today

- Pick and configure an optimizer for small control problems (unconstrained & constrained).
- Derive KKT conditions and form the SQP QP subproblem for a nonlinear program.
- Explain the differences between penalty, augmented Lagrangian, and interior-point methods.

Why?

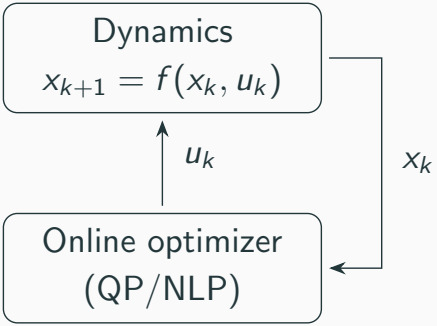
In future classes, this will help us map classic control tasks (LQR/MPC/trajectory optimization) to QPs/NLPs and choose a solver strategy.

Roadmap for today (2 hours)

1. Big picture and some notation *(5 min)*
2. Unconstrained optimization: Root-finding, Newton and globalization *(30 min)*
3. Equality constraints: KKT, Newton vs. Gauss–Newton *(20 min)*
4. Inequalities & KKT: complementarity *(10 min)*
5. Methods: penalty $\rightarrow$ ALM $\rightarrow$ interior-point (PDIP) *(20 min)*
6. Brief look at SQP for solving hard control problems *(20 min)*

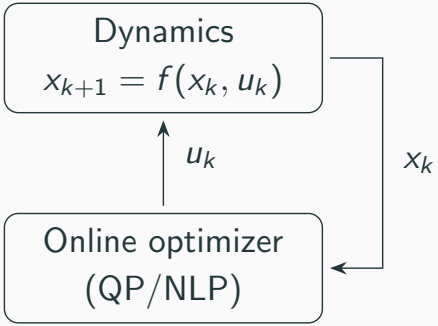
Big picture: why optimization for control?

- Controller synthesis often reduces to solving a sequence of optimization problems.



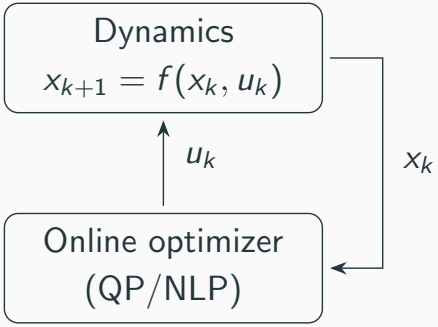
Big picture: why optimization for control?

- Controller synthesis often reduces to solving a sequence of optimization problems.
- **MPC** solves a QP/NLP online at each time step; warm-start and sparsity are critical.



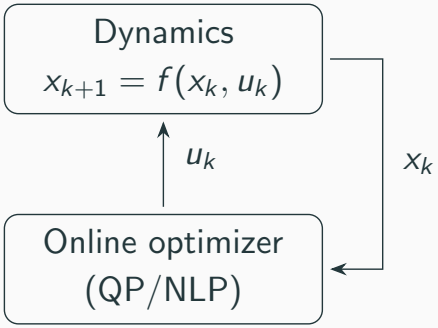
Big picture: why optimization for control?

- Controller synthesis often reduces to solving a sequence of optimization problems.
- **MPC** solves a QP/NLP online at each time step; warm-start and sparsity are critical.
- **Trajectory optimization** (nonlinear robots) uses NLP + collocation; needs robust globalization.



Big picture: why optimization for control?

- Controller synthesis often reduces to solving a sequence of optimization problems.
- **MPC** solves a QP/NLP online at each time step; warm-start and sparsity are critical.
- **Trajectory optimization** (nonlinear robots) uses NLP + collocation; needs robust globalization.
- **Learning-based control** backpropagates through optimizers (differentiable programming).



Some Notation:

Given $f(x) : \mathbb{R}^n \to \mathbb{R}$ $\quad\quad$ $\frac{\partial f}{\partial x} \in \mathbb{R}^{1 \times n}$ is a row vector

This is because $\frac{\partial f}{\partial x}$ is the linear operator mapping Δx into Δf:

$$f(x + \Delta x) \approx f(x) + \frac{\partial f}{\partial x} \Delta x$$

Similarly $g(y) : \mathbb{R}^m \to \mathbb{R}^n$ $\quad\quad$ $\frac{\partial g}{\partial y} \in \mathbb{R}^{n \times m}$ because

$$g(y + \Delta y) \approx g(y) + \frac{\partial g}{\partial y} \Delta y$$

Some notation:

For convenience, we will define:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x}\right)^T \in \mathbb{R}^n \quad \text{(column vector)}$$

$$\nabla^2 f(x) = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right) = \frac{\partial^2 f}{\partial x^2} \in \mathbb{R}^{n \times n}$$

$$f(x + \Delta x) \approx f(x) + \frac{\partial f}{\partial x}\Delta x + \tfrac{1}{2}\Delta x^T \nabla^2 f(x)\Delta x$$

Root-Finding

Root Finding

Given $f(x)$, find x^* such that $f(x^*) = 0$ (example: finding equilibrium of a continuous-time dynamics).

Closely related: fixed point $f(x^*) = x^*$ (equilibrium of discrete-time dynamics)

Fixed-Point Iteration

- Simplest solution method
- If fixed point is stable, just "iterate the dynamics" until it converges
- Only works if x^* is a stable equilibrium point and if initial guess is in the basin of attraction
- can converge slowly (depends on f i.e depending on eigenvalues)

Gradient Descent in Fixed Point Iteration applied to gradient of function f.

Newton's Method

TLDR: Instead os solving for $f(x) = 0$, solve a linear for a linear approximation of $f(x)$.
Fit a linear approximation to $f(x)$: $f(x + \Delta x) \approx f(x) + \frac{\partial f}{\partial x}\Delta x$

Set approximation to zero and solve for Δx:

$$f(x) + \frac{\partial f}{\partial x}\Delta x = 0 \quad \Rightarrow \quad \Delta x = -\left(\frac{\partial f}{\partial x}\right)^{-1} f(x)$$

Apply correction: $x \leftarrow x + \Delta x$ Do this in a loop and repeat until convergence.

Example: Backward Euler

Last time: Implicit dynamics model (nonlinear function of current state and future state)

$$f(x_{n+1}, x_n, u_n) = 0$$

Implicit Euler: this time we have x_{n+1} on the right; i.e evaluate f at future time.

$$x_{n+1} = x_n + hf(x_{n+1})$$

(Evaluate f at future time)

$$\Rightarrow f(x_{n+1}, x_n, u_n) = x_{n+1} - x_n - hf(x_{n+1}) = 0$$

Solve root finding problem for x_{n+1}

- Very fast convergence with Newton (quadratic) and can get machine precision.
- Most expensive part is solving a linear system $O(n^3)$
- Can improve complexity by taking advantage of problem structure/sparsity.

Move to Julia Code

Quick Demo of Julia Notebook: part1_root_finding.ipynb

Minimization

$$\min_x f(x), \quad f(x) : \mathbb{R}^n \to \mathbb{R}$$

If f is smooth, $\frac{\partial f}{\partial x}(x^*) = 0$ at a local minimum.

Hence, now we have a root-finding problem $\nabla f(x) = 0 \Rightarrow$ Apply Newton!

$$\nabla f(x + \Delta x) \approx \nabla f(x) + \frac{\partial}{\partial x}(\nabla f(x))\Delta x = 0 \qquad \Rightarrow \Delta x = -(\nabla^2 f(x))^{-1}\nabla f(x)$$

$$x \leftarrow x + \Delta x$$

Repeat this step until convergence; Intuition to have about Newton:

- Fitting a quadratic approximation to $f(x)$; Exactly minimize approximation

Move to Julia Code

Quick Demo of Julia Notebook: part1_minimization.ipynb

Take-away Messages on Newton

Newton is a local root-finding method. Will converge to the closest fixed point to the initial guess (min, max, saddle).

Sufficient Conditions

- $\nabla f = 0$: "first-order necessary condition" for a minimum. Not a sufficient condition.
- Let's look at scalar case: $\Delta x = -\frac{1}{\nabla^2 f} \nabla f$

where: negative corresponds to "descent", ∇f corresponds to the gradient and $\nabla^2 f$ acts as the "leading rate" / "step size".

$\nabla^2 f > 0 \quad \Rightarrow \quad$ descent (minimization) $\qquad \nabla^2 f < 0 \quad \Rightarrow \quad$ ascent (maximization)

- In $\mathbb{R}^n$, if $\nabla^2 f \succeq 0$ (positive definite) $\Rightarrow$ descent
- If $\nabla^2 f > 0$ everywhere $\Rightarrow f(x)$ is strongly convex $\rightarrow$ Can always solve with Newton

Regularization: Ensuring Local Minimization

Practical solution to make sure we always minimize:

If H ($H \leftarrow \nabla^2 f$) not positive definite, we just make it so with regularization.

While $H \not\succeq 0$:

$$H \leftarrow H + \beta I \quad (\beta > 0 \text{ scalar hyperparameter})$$

Then do newton step as usual. I.e:

$$x \quad \leftarrow \quad x + \Delta x \quad = \quad x - H^{-1} \nabla f$$

- also called "damped Newton" (shrinks steps)
- Guarantees descent
- Regularization makes sure we minimize, but what about over-shooting?

Line Search: Mitigating overshooting in Newton

- Often Δx step from Newton overshoots the minimum.
- To fix this, check $f(x + \alpha\Delta x)$ and "back track" until we get a "good" reduction.
- Many strategies: all differ in definition of good.
- A simple + effective one is **Armijo Rule**:

Start with $\alpha = 1$ as our step length and have tolerance b as a hyper-parameter.

$$\text{while } f(x + \alpha\Delta x) > f(x) + b\alpha\nabla f(x)^T\Delta x : \qquad \alpha \leftarrow c\alpha \quad (\text{scalar } < 1 \text{ i.e } c = \frac{1}{2})$$

The intuition behind this is that $\alpha\nabla f(x)^T\Delta x$ corresponds to the expected change in f based on a 1st order taylor series expansion. I.e we are checking the actual decrease in f agrees with a 1st order taylor approximation within a tolerance b.

Constrained Optimization

Equality-constrained minimization: geometry and conditions

Problem.; $\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad C(x) = 0, C : \mathbb{R}^n \to \mathbb{R}^m$.

Geometric picture. At an optimum on the manifold $C(x) = 0$, the negative gradient must lie in the tangent space:

$$\nabla f(x^\star) \ \perp \ \mathcal{T}_{x^\star} = \{p : \ J_C(x^\star)p = 0\}.$$

Equivalently, the gradient is a linear combination of constraint normals:

$$\nabla f(x^\star) + J_C(x^\star)^T \lambda^\star = 0, \qquad C(x^\star) = 0 \quad (\lambda^\star \in \mathbb{R}^m).$$

Lagrangian.; $L(x, \lambda) = f(x) + \lambda^T C(x)$.

A nicer visual explanation/derivation of KKT conditions

Quick little whiteboard derivation

KKT system for equalities (first-order necessary conditions)

KKT (FOC).

$$\nabla_x L(x,\lambda) = \nabla f(x) + J_C(x)^T \lambda = 0, \qquad \nabla_\lambda L(x,\lambda) = C(x) = 0.$$

Solve by Newton on KKT: linearize both optimality and feasibility:

$$\begin{bmatrix} \nabla^2 f(x) + \sum_{i=1}^m \lambda_i \nabla^2 C_i(x) & J_C(x)^T \\ J_C(x) & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(x) + J_C(x)^T \lambda \\ C(x) \end{bmatrix}.$$

Notes. This is a symmetric *saddle-point* system; typical solves use block elimination (Schur complement) or sparse factorizations.

Move to Julia Code

Quick Demo of Julia Notebook: part2_eq_constraints.ipynb

Numerical practice: Newton on KKT

When it works best.

- Near a regular solution with $J_C(x^\star)$ full row rank and positive-definite reduced Hessian.
- With a globalization (line search on a merit function) and mild regularization for robustness.

Common safeguards.

- *Regularize* the $(1,1)$ block to ensure a good search direction (e.g., add βI).
- *Merit/penalty* line search to balance feasibility vs. optimality during updates.
- *Scaling* constraints to improve conditioning of the KKT system.

Gauss–Newton vs. full Newton on KKT

Full Newton Hessian of the Lagrangian: $\nabla^2_{xx} L(x, \lambda) = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 C_i(x)$

Gauss–Newton approximation: drop the *constraint-curvature* term $\sum_i \lambda_i, \nabla^2 C_i(x)$:

$$H_{\mathrm{GN}}(x) \approx \nabla^2 f(x).$$

Trade-offs (high level).

- *Full Newton:* fewer iterations near the solution, but each step is costlier and can be less robust far from it.
- *Gauss–Newton:* cheaper per step and often more stable; may need more iterations but wins in wall-clock on many problems.

Practice tip. Start with GN (with line search); switch to full Newton (or add low-rank updates) as feasibility improves.

Inequality-constrained minimization and KKT

Problem. $\min f(x) \quad \text{s.t.} \quad c(x) \geq 0, \qquad c : \mathbb{R}^n \to \mathbb{R}^p.$

KKT conditions (first-order).

$$\begin{aligned}
&\text{Stationarity:} && \nabla f(x) - J_c(x)^T \lambda = 0, \\
&\text{Primal feasibility:} && c(x) \geq 0, \\
&\text{Dual feasibility:} && \lambda \geq 0, \\
&\text{Complementarity:} && \lambda^T c(x) = 0 \quad (\text{i.e., } \lambda_i c_i(x) = 0 \ \forall i).
\end{aligned}$$

Interpretation.

- *Active* constraints: $c_i(x) = 0 \Rightarrow \lambda_i \geq 0$ can be nonzero (acts like an equality).
- *Inactive* constraints: $c_i(x) > 0 \Rightarrow \lambda_i = 0$ (no influence on optimality).

Complementarity in plain English (and why Newton is tricky)

What $\lambda_i c_i(x) = 0$ means.

- Tight constraint ($c_i = 0$) $\Rightarrow$ can press back ($\lambda_i \geq 0$).
- Loose constraint ($c_i > 0$) $\Rightarrow$ no force ($\lambda_i = 0$).

Why naive Newton fails.

- Complementarity = nonsmooth + inequalities ($\lambda \geq 0$, $c(x) \geq 0$).
- Equality-style Newton can violate nonnegativity or bounce across boundary.

Two main strategies (preview).

- *Active-set:* guess actives $\Rightarrow$ solve equality-constrained subproblem, update set.
- *Barrier/PDIP/ALM:* smooth or relax complementarity, damped Newton, drive relaxation $\to 0$.

Minimization w/ Inequality Constraints

Three families you should know (high level)

Goal: Handle inequalities $c(x) \geq 0$ (and equalities) robustly and efficiently.

Families.

1. **Penalty**: embed violations in the objective; crank a parameter $\rho \uparrow$.
2. **Augmented Lagrangian (ALM)**: maintain multipliers & a *moderate* penalty; solve easier subproblems.
3. **Interior-Point (PDIP)**: enforce $c(x) > 0$ via a barrier; follow the *central path* with primal–dual Newton.

Rule of thumb. Penalty is simplest; ALM is a strong default for medium accuracy; PDIP is the gold standard for convex QPs and very robust with Newton.

Inequality-Constrained Minimization

Problem Setup:

$$\min f(x) \quad \text{s.t. } c(x) \geq 0$$

KKT conditions:

$$\nabla f - \left(\frac{\partial c(x)}{\partial x}\right)^T \lambda = 0 \quad \text{(stationarity)}$$

$$c(x) \geq 0 \quad \text{(primal feasibility)} \qquad \lambda \geq 0 \quad \text{(dual feasibility)}$$

$$\lambda \circ c(x) = \lambda^T c(x) = 0 \quad \text{(complementarity)}$$

Unlike equality case, we can't directly solve KKT conditions with Newton! Why?

Lots of solution methods to use: Active Set Method

Active Set Method

- High level idea: Guess which inequalities are redundant at optimality and throw them away.
- Switch inequality constraints on/off in outer-loop and solve equality-constrained problem.
- Works well if you can guess active set well (common in MPC where good warm-starts are common).
- Has really bad worst-time complexity.
- Usually custom heuristics are used for specific problem classes/structure.

Lots of solution methods to use: Penalty Methods

Penalty Method: Replace constraints with cost terms that penalize violation!

$$\min_x \; f(x) \; + \; \tfrac{\rho}{2} \left\| c^-(x) \right\|_2^2, \qquad c^-(x) := \min(0, c(x)) \text{ (elementwise).}$$

Algorithm sketch.

1. Start with a small $\rho > 0$; minimize the penalized unconstrained objective.
2. Increase ρ (e.g., $\times 10$) and warm start from previous x.
3. Stop when $c^-(x)$ is small enough.

Pros. Dead simple; reuse unconstrained machinery (Grad/Newton + line search).
Cons. Ill-conditioning as $\rho \to \infty$ to satisfy constraints.; struggles to reach high accuracy; multipliers are implicit.
Popular fix: estimate λ from penalty at each iteration to be able to converge with finite ρ. Called "Augmented Lagrangian" (also closely related to ADMM).

Augmented Lagrangian (ALM): fix penalty's weaknesses

Core idea. Introduce multipliers λ so we can keep ρ moderate and still achieve accuracy.

Lagrangian for equality case: $\mathcal{L}_\rho(x, \lambda) = f(x) + \lambda^T C(x) + \frac{\rho}{2}\|C(x)\|_2^2$.

Outer loop.

1. $x^{k+1} \approx \arg\min_x \mathcal{L}_\rho(x, \lambda^k)$ (unconstrained solve).
2. $\lambda^{k+1} = \lambda^k + \rho\, C(x^{k+1})$.

Inequalities (sketch). Apply to the *hinge* $c^-(x)$ and keep $\lambda \geq 0$:

$$\mathcal{L}_\rho(x, \lambda) = f(x) - \lambda^T c(x) + \tfrac{\rho}{2}\|c^-(x)\|_2^2, \quad \lambda^{k+1} = \max\big(0, \lambda^k - \rho\, c(x^{k+1})\big).$$

Why it works. Subproblems are better conditioned than pure penalty; λ estimates improve the model; finite ρ can reach high accuracy.

ALM in practice (optimization loop view)

Inner solver. Use (damped) Newton or quasi-Newton on $\mathcal{L}_\rho(\cdot, \lambda^k)$ with Armijo/Wolfe line search.

Tuning.

- Keep ρ fixed or adapt slowly (increase if feasibility stalls).
- Scale constraints; monitor $|C(x)|$ and stationarity.

When to pick ALM.

- Nonconvex NLPs where feasibility progress matters and you want robust globalization.
- Medium accuracy is fine, or as a precursor to a polished PDIP phase on a convex QP.

Interior-Point / Barrier Methods

Replace inequalities with barrier function in objective:

$$\min f(x), \quad x \geq 0 \quad \rightarrow \quad \min f(x) - \rho \log(x)$$

- Gold standard for convex problems.
- Fast convergence with Newton and strong theoretical properties.
- Used in IPOPT.

(Diagram: $-\log(x)$ barrier.)

Primal-Dual Interior Point Method

$$\min f(x) \quad \text{s.t. } x \geq 0$$

$$\rightarrow \min f(x) - \rho \log(x)$$

$$\frac{\partial f}{\partial x} - \frac{\rho}{x} = 0$$

- This "primal" FON condition blows up as $x \rightarrow 0$.
- We can fix this with the "primal-dual trick."

The Primal-Dual Trick for IPM

Introduce new variable $\lambda = \frac{\rho}{x} \quad \Rightarrow \quad x\lambda = \rho.$

$$\begin{cases} \nabla f - \lambda = 0 \\ x\lambda = \rho \end{cases}$$

- This can actually be viewed as a relaxed complementarity slackness from KKT!
- Converges to exact KKT solution as $\rho \to 0$.
- We lower ρ gradually as solver converges (from $\rho \sim 1$ to $\rho \sim 10^{-6}$).
- Note: we still need to enforce $x \geq 0$ and $\lambda \geq 0$ (with line search).

We will use another approach from 2022 from a researcher at TRI that developped an even cooler trick.

Log-Domain Interior-Point Method

More general constraint case: $\min f(x) \quad \text{s.t. } c(x) \geq 0$

Simplify by introducing a "slack variable":

$$\min_{x,s} f(x) \quad \text{s.t. } c(x) - s = 0,\ s \geq 0$$

$$\rightarrow \min_{x,s} f(x) - \rho \log(s) \quad \text{s.t. } c(x) - s = 0$$

Write out Lagrangian: $L(x, s, \lambda) = f(x) - \rho \log(s) - \lambda^T (c(x) - s)$

Apply F.O.N.C to Lagrangian from last slide:

$$\nabla_x L = \nabla f - \left(\frac{\partial c}{\partial x}\right)^T \lambda = 0$$

$$\nabla_s L = \frac{\rho}{s} + \lambda = 0 \quad \Rightarrow \quad s\lambda = \rho$$

$$\nabla_\lambda L = s - c(x) = 0$$

This second equation has a really nice interpretation: relaxed complementarity slackness

Log-Domain Interior-Point Method

To ensure $s \geq 0$ and $\lambda \geq 0$, introduce change of variables:

$$s = \sqrt{\rho} e^{\sigma}, \quad \lambda = \sqrt{\rho} e^{-\sigma}$$

Now (relaxed) complementarity is always satisfied by construction.

Plug back into F.O.N.C

$$\nabla f - \left(\frac{\partial c}{\partial x}\right)^T \lambda = 0 \qquad c(x) - \sqrt{\rho} e^{\sigma} = 0$$

We can solve these with (Gauss) Newton:

$$\begin{bmatrix} H & \sqrt{\rho} c^T e^{-\sigma} \\ c & -\sqrt{\rho} e^{\sigma} \end{bmatrix} \begin{bmatrix} \delta x \\ \delta \sigma \end{bmatrix} = \begin{bmatrix} -\nabla f + c^T \lambda \\ -c(x) + \sqrt{\rho} e^{\sigma} \end{bmatrix}$$

Example: Quadratic Program

Super common problem to be solved in control applications: quadratic programs

$$\min_x \frac{1}{2} x^T Q x + q^T x, \quad Q \succeq 0$$

s.t.

$$Ax = b, \quad Cx \leq d$$

- Super useful in control (SQP)
- Can be solved very fast ($\sim kHz$).

Move to Julia Code

Quick Demo of Julia Notebook: part3_ipm.ipynb

Penalty vs. ALM vs. PDIP: what changes?

- **Feasibility handling:**
  - Penalty: encourages $c(x) \geq 0$ via cost; feasibility only in the limit $\rho \uparrow$.
  - ALM: balances optimality and feasibility via λ updates at finite ρ.
  - PDIP: enforces strict interior $c(x) > 0$; drives $s_i \lambda_i = \rho \to 0$.
- **Conditioning:**
  - Penalty gets ill-conditioned as ρ grows.
  - ALM keeps conditioning reasonable.
  - PDIP maintains well-scaled Newton systems near the path (with proper scaling).
- **Accuracy:** Penalty (low–med), ALM (high with finite ρ), PDIP (high; excellent for convex).
- **Per-iteration work:** Penalty/ALM solve unconstrained-like subproblems; PDIP solves structured KKT systems with slacks/duals.

Sequential Quadratic Programming (SQP)

What is SQP?

Idea: Solve a nonlinear, constrained problem by repeatedly solving a *quadratic program (QP)* built from local models.

- Linearize constraints; quadratic model of the Lagrangian/objective.
- Each iteration: solve a QP to get a step d, update $x \leftarrow x + \alpha d$.
- Strength: strong local convergence (often superlinear) with good Hessian info.

Target Problem (NLP)

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

- $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$ (equalities), $h : \mathbb{R}^n \to \mathbb{R}^p$ (inequalities).
- KKT recap (at candidate optimum $x^\star$):

$$\exists\, \lambda \in \mathbb{R}^m,\ \mu \in \mathbb{R}^p_{\geq 0} :\ \nabla f(x^\star) + \nabla g(x^\star)^T \lambda + \nabla h(x^\star)^T \mu = 0,$$

$$g(x^\star) = 0, \quad h(x^\star) \leq 0, \quad \mu \geq 0, \quad \mu \odot h(x^\star) = 0.$$

From NLP to a QP (Local Model)

At iterate x_k with multipliers (λ_k, μ_k):

Quadratic model of the Lagrangian

$$m_k(d) = \langle \nabla f(x_k),\, d \rangle + \tfrac{1}{2} d^T B_k d$$

with $B_k \approx \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k, \mu_k)$.

Linearized constraints

$$g(x_k) + \nabla g(x_k)\, d = 0, \qquad h(x_k) + \nabla h(x_k)\, d \leq 0.$$

The SQP Subproblem (QP)

$$\begin{aligned} \min_{d \in \mathbb{R}^n} \quad & \nabla f(x_k)^T d + \tfrac{1}{2} d^T B_k d \\ \text{s.t.} \quad & \nabla g(x_k)\, d + g(x_k) = 0, \\ & \nabla h(x_k)\, d + h(x_k) \leq 0. \end{aligned}$$

- Solve QP $\Rightarrow$ step d_k and updated multipliers $(\lambda_{k+1}, \mu_{k+1})$.
- Update $x_{k+1} = x_k + \alpha_k d_k$ (line search or trust-region).

Algorithm Sketch (SQP)

1. Start with x_0, multipliers (λ_0, μ_0), and $B_0 \succ 0$.
2. Build QP at x_k with B_k, linearized constraints.
3. Solve QP $\Rightarrow$ get d_k, $(\lambda_{k+1}, \mu_{k+1})$.
4. Globalize: line search on merit or use filter/TR to choose α_k.
5. Update $x_{k+1} = x_k + \alpha_k d_k$, update B_{k+1} (e.g., BFGS).

Toy Example (Local Models)

Problem:

$$\min_{x\in\mathbb{R}^2} \ \tfrac{1}{2}\|x\|^2 \quad \text{s.t.} \quad g(x) = x_1^2 + x_2 - 1 = 0, \quad h(x) = x_2 - 0.2 \le 0.$$

At x_k, build QP with

$$\nabla f(x_k) = x_k, \quad B_k = I, \quad \nabla g(x_k) = \begin{bmatrix} 2x_{k,1} & 1 \end{bmatrix}, \ \nabla h(x_k) = \begin{bmatrix} 0 & 1 \end{bmatrix}.$$

Solve for d_k, then $x_{k+1} = x_k + \alpha_k d_k$.

Globalization: Making SQP Robust

SQP is an important method, and there are many issues to be considered to obtain an **efficient** and **reliable** implementation:

- Efficient solution of the linear systems at each Newton Iteration (Matrix block structure can be exploited.
- Quasi-Newton approximations to the Hessian.
- Trust region, line search, etc. to improve robustnes (i.e TR: restrict $\|d\|$ to maintain model validity.)
- Treatment of constraints (equality and inequality) during the iterative process.
- Selection of good starting guess for λ.

Final Takeaways on SQP

When SQP vs. Interior-Point?

- **SQP**: strong local convergence; warm-start friendly; natural for NMPC.
- **IPM**: very robust for large, strictly feasible problems; good for dense inequality sets.
- In practice: both are valuable—choose to match problem structure and runtime needs.

Takeaways of SQP

- SQP = Newton-like method using a sequence of structured QPs.
- Globalization (merit/filter/TR) makes it reliable from poor starts.
- Excellent fit for control (NMPC/trajectory optimization) due to sparsity and warm starts.