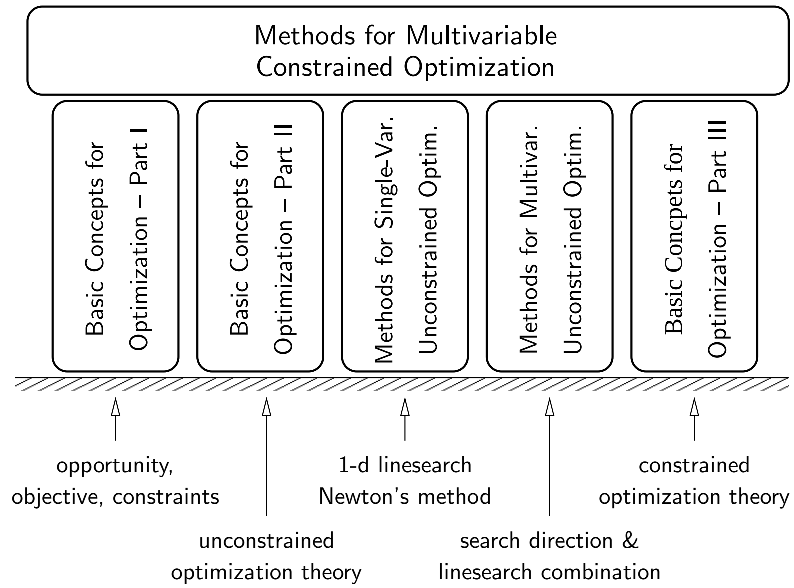
Non-Linear Programming (NLP): Multivariable, Constrained

Benoît Chachuat <benoit@mcmaster.ca>

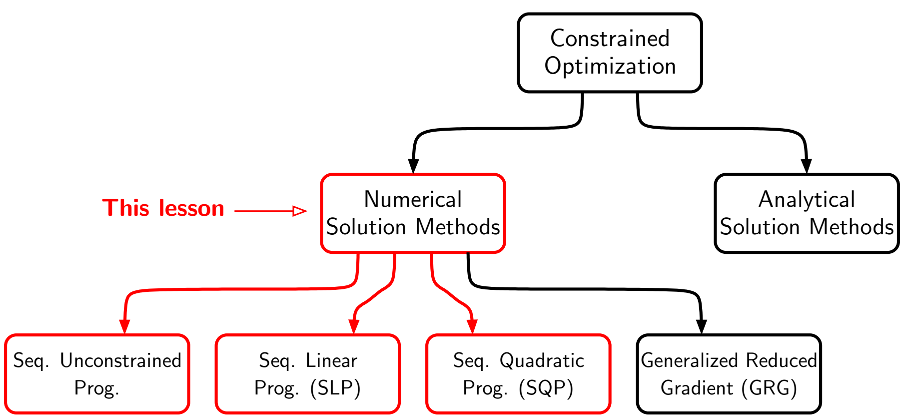
McMaster University
Department of Chemical Engineering

ChE 4G03: Optimization in Chemical Engineering

Solving Multivariable, Constrained NLPs



Outline



For additional details, see Rardin (1998), Chapter 14.5-14.7
(also check: `http://www.mpri.lsu.edu/textbook/Chapter6.htm`)

Penalty Methods

Idea: Transform a constrained NLP into an unconstrained NLP

- Consider the NLP problem

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n}: &\quad f(\mathbf{x}) \\ \text{subject to:} &\quad g_j(\mathbf{x}) \leq 0, \quad j = 1,\ldots,m_\mathrm{i} \\ &\quad h_j(\mathbf{x}) = 0, \quad j = 1,\ldots,m_\mathrm{e} \end{aligned}$$

- **Penalty methods** drop constraints and substitute new terms in the objective function penalizing infeasibility:

$$\min_{\mathbf{x}\in\mathbb{R}^n}: \quad F(\mathbf{x}) \stackrel{\Delta}{=} f(\mathbf{x}) + \mu\left[\sum_{j=1}^{m_\mathrm{e}} p_j^\mathrm{e}(\mathbf{x}) + \sum_{j=1}^{m_\mathrm{i}} p_j^\mathrm{i}(\mathbf{x})\right]$$

with $\mu > 0$ the **penalty multiplier**; F, the **auxiliary function**

Penalty Functions for Constrained NLPs

Inequality Constraints: $g_j(\mathbf{x}) \leq 0$

$$\begin{cases} p_j^{\mathrm{i}}(\mathbf{x}) = 0, & \text{if } g_j(\mathbf{x}) \leq 0 \\ p_j^{\mathrm{i}}(\mathbf{x}) > 0, & \text{otherwise} \end{cases}$$

Common Choices:

$$\boxed{p_j^{\mathrm{i}}(\mathbf{x}) \triangleq \max\{0, g_j(\mathbf{x})\}^\gamma,\ \gamma \geq 1}$$

Equality Constraints: $h_j(\mathbf{x}) = 0$

$$\begin{cases} p_j^{\mathrm{e}}(\mathbf{x}) = 0, & \text{if } h_j(\mathbf{x}) = 0 \\ p_j^{\mathrm{e}}(\mathbf{x}) > 0, & \text{otherwise} \end{cases}$$

Common Choices:

$$\boxed{p_j^{\mathrm{e}}(\mathbf{x}) \triangleq |h_j(\mathbf{x})|^\gamma,\ \gamma \geq 1}$$

Exact vs. Inexact Penalty Functions

- If the unconstrained optimum of a penalty model F is feasible in the original NLP, it is also optimal in that NLP
- If the unconstrained optimum of a penalty model F is optimal in the original NLP for some finite value of μ, the corresponding penalty function is said to be exact
- If no such finite value of μ exists, it is said to be inexact (yields an optimum as $\mu \to \infty$ only)

Penalty Functions for Constrained NLPs

Inequality Constraints: $g_j(\mathbf{x}) \leq 0$

$$\begin{cases} p_j^{\mathrm{i}}(\mathbf{x}) = 0, & \text{if } g_j(\mathbf{x}) \leq 0 \\ p_j^{\mathrm{i}}(\mathbf{x}) > 0, & \text{otherwise} \end{cases}$$

Common Choices:

$$\boxed{p_j^{\mathrm{i}}(\mathbf{x}) \triangleq \max\{0, g_j(\mathbf{x})\}^\gamma,\ \gamma \geq 1}$$

Equality Constraints: $h_j(\mathbf{x}) = 0$

$$\begin{cases} p_j^{\mathrm{e}}(\mathbf{x}) = 0, & \text{if } h_j(\mathbf{x}) = 0 \\ p_j^{\mathrm{e}}(\mathbf{x}) > 0, & \text{otherwise} \end{cases}$$

Common Choices:

$$\boxed{p_j^{\mathrm{e}}(\mathbf{x}) \triangleq |h_j(\mathbf{x})|^\gamma,\ \gamma \geq 1}$$

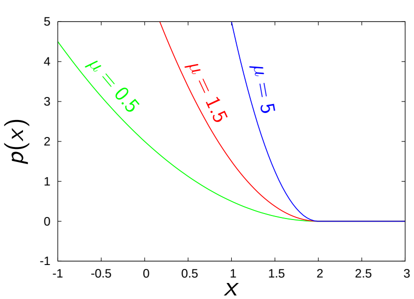
- Nonsquared penalty functions, $\gamma = 1$:
  - Such penalty functions are exact, under mild assumptions, for μ sufficiently large
  - But, the auxiliary function F is nonsmooth even though g_j and h_j may be differentiable
- Squared penalty functions, $\gamma = 2$:
  - The auxiliary function F is differentiable provided that g_j and h_j are differentiable
  - But, such penalty functions are typically inexact

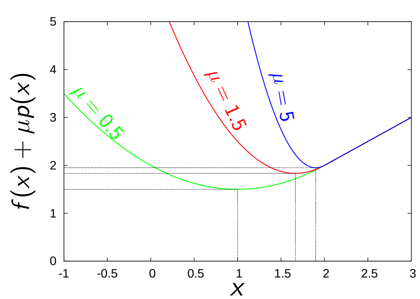
Constructing and Solving Penalty Models

Class Exercise: Consider the optimization problem

$$\min_x\ f(x) \triangleq x$$
$$\text{s.t. } g(x) \triangleq 2 - x \leq 0$$

1. Solve this problem by inspection
2. Construct a penalty model using a square penalty function, then solve the unconstrained NLP as a function of the penalty multiplier μ





Pros and Cons of Penalty Models

Pros:

- Straightforward approach
- Possible use of fast and robust algorithms for unconstrained NLP (e.g., BFGS quasi-Newton search)

Cons:

- Large penalty multipliers lead to ill-conditioned penalty models
  - Subject to slow convergence (small steps)
  - Possible early termination (numerical errors)

In practice: $\boxed{\text{Sequential Unconstrained Penalty Algorithm}}$

- Considers a sequence of increasing penalty parameters, $\mu^0 < \mu^1 < \ldots$
- Solves each new optimization problem (μ^{k+1}) from the optimal solution obtained for the previous problem $(\mathbf{x}^k)$
- Produces a sequence of infeasible points, whose limit is an optimal solution to the original NLP (*exterior* penalty function approach)

Sequential Unconstrained Penalty Algorithm

- **Step 0: Initialization**
  - Form penalty model; choose initial guess $\mathbf{x}^0$, penalty multiplier $\mu^0 > 0$, escalation factor $\beta > 1$, and stopping tolerance $\epsilon > 0$; set $k \leftarrow 0$
- **Step 1: Unconstrained Optimization**
  - Direction: Starting from $\mathbf{x}^k$, solve penalty optimization problem

$$\min_{\mathbf{x}} \; F(\mathbf{x}) \stackrel{\Delta}{=} f(\mathbf{x}) + \mu \left[\sum_{j=1}^{m_e} p_j^{e}(\mathbf{x}) + \sum_{j=1}^{m_i} p_j^{i}(\mathbf{x}) \right],$$

  with $\mu = \mu^k$, to produce $\mathbf{x}^{k+1}$
- **Step 2: Stopping**
  - If $\mu^k \left[\sum_{j=1}^{m_e} p_j^{e}(\mathbf{x}^{k+1}) + \sum_{j=1}^{m_i} p_j^{i}(\mathbf{x}^{k+1}) \right] < \epsilon$, stop — report $\mathbf{x}^{k+1}$ (approximate KKT point)
- **Step 3: Update**
  - Enlarge the penalty parameter as $\mu^{k+1} \leftarrow \beta \mu^k$
  - Increment $k \leftarrow k+1$ and return to step 1

Barrier Methods

Idea: Transform a constrained NLP into an unconstrained NLP

- Consider the NLP problem with inequality constraints only

$$\begin{aligned} \underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimize:}} \quad & f(\mathbf{x}) \\ \text{subject to:} \quad & g_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, m_i \end{aligned}$$

- **Barrier methods** drop constraints and substitute new terms in the objective function discouraging approach to the boundary of the feasible region:

$$\underset{\mathbf{x} \in \mathbb{R}^n}{\text{minimize:}} \quad F(\mathbf{x}) \stackrel{\Delta}{=} f(\mathbf{x}) + \mu \sum_{j=1}^{m_i} b_j(\mathbf{x}), \qquad b_j(\mathbf{x}) \xrightarrow{g_j(\mathbf{x}) \nearrow 0} +\infty,$$

  with $\mu > 0$ the **barrier multiplier**; F, the **auxiliary function**

Barrier Functions for Inequality Constrained NLPs

Ideal Barrier Function: $g_j(\mathbf{x}) \leq 0$

$$\begin{cases} b_j(\mathbf{x}) = 0, & \text{if } g_j(\mathbf{x}) < 0 \\ b_j(\mathbf{x}) = +\infty, & \text{otherwise} \end{cases}$$

Common Barrier Functions:

$$\boxed{b_j(\mathbf{x}) \stackrel{\Delta}{=} -\frac{1}{g_j(\mathbf{x})}, \qquad b_j(\mathbf{x}) \stackrel{\Delta}{=} -\ln\left(-g_j(\mathbf{x})\right)}$$

Properties of Barrier Functions

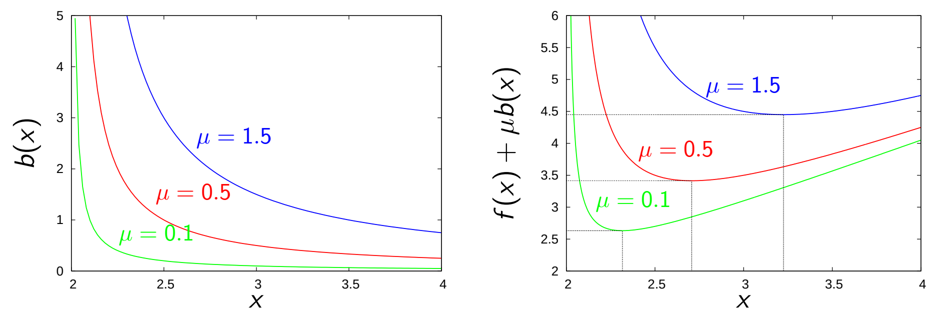
- The optimum of a barrier model can never equal the optimum of the original NLP model if $\mu > 0$ and that optimum lies on the boundary of the feasible domain
- However, as $\mu \searrow 0$, the unconstrained optimum comes closer and closer to the constrained solution (as with penalty methods)

Constructing and Solving Barrier Models

Class Exercise: Consider the same optimization problem as previously

$$\min_x \; f(x) \stackrel{\Delta}{=} x$$
$$\text{s.t. } g(x) \stackrel{\Delta}{=} 2 - x \leq 0$$

1. Construct a barrier model using the inverse barrier function, then solve the unconstrained NLP as a function of the barrier multiplier μ



Pros and Cons of Barrier Models

Pros:

- Straightforward approach
- Possible use of fast and robust algorithms for unconstrained NLP (e.g., BFGS quasi-Newton search)

Cons:

- Small barrier multipliers lead to ill-conditioned barrier models
  - Subject to slow convergence (small steps)
  - Possible early termination (numerical errors)

In practice: Sequential Unconstrained Barrier Algorithm

- Considers a sequence of decreasing, positive barrier parameters, $\mu^0 > \mu^1 > \ldots > 0$
- Solves each new optimization problem (μ^{k+1}) from the optimal solution obtained for the previous problem ($\mathbf{x}^k$)
- Produces a sequence of feasible points, whose limit is an optimal solution to the original NLP (*interior point* approach)

Sequential Unconstrained Barrier Algorithm

- **Step 0: Initialization**
  - Form barrier model; choose initial guess $\mathbf{x}^0$, barrier multiplier $\mu^0 > 0$, reduction factor $0 < \beta < 1$, and stopping tolerance $\epsilon > 0$; set $k \leftarrow 0$
- **Step 1: Unconstrained Optimization**
  - Direction: Starting from $\mathbf{x}^k$, solve barrier optimization problem

$$\min_{\mathbf{x}} \; F(\mathbf{x}) \stackrel{\Delta}{=} f(\mathbf{x}) + \mu \sum_{j=1}^{m_e} b_j(\mathbf{x}),$$

  with $\mu = \mu^k$, to produce $\mathbf{x}^{k+1}$
- **Step 2: Stopping**
  - If $\mu^k \sum_{j=1}^{m_e} b_j(\mathbf{x}^{k+1}) < \epsilon$, stop — report $\mathbf{x}^{k+1}$ (approximate KKT point)
- **Step 3: Update**
  - Decrease the penalty parameter as $\mu^{k+1} \leftarrow \beta\mu^k$
  - Increment $k \leftarrow k+1$ and return to step 1

Sequential Linear Programming Methods

Idea: Develop a method for constrained NLP based on a sequence of LP approximations

- Follows the improving-search paradigm:
  - Generate a search direction by formulating, then solving, an LP problem at each iteration
  - LP problems can be solved both reliably and efficiently
- An LP solution is always obtained at a corner/extreme point of the feasible region:
  - A successful approach must consider extra bounds on the direction components: a *"trust region"* $\pm\boldsymbol{\delta}$
  - The common approach is to bound the direction components with a "*box*" (or hypercube)

LP-based Search Direction: Principles

- Consider the NLP problem

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n}\text{imize:} \quad & f(\mathbf{x}) \\ \text{subject to:} \quad & g_j(\mathbf{x}) \leq 0, \quad j = 1,\ldots,m_i \\ & h_j(\mathbf{x}) = 0, \quad j = 1,\ldots,m_e \end{aligned}$$

- An **LP-based search direction** $\boldsymbol{\Delta}\mathbf{x}$ at a given point $\bar{\mathbf{x}}$ is determined by linearizing the original NLP problem at $\bar{\mathbf{x}}$:

$$\begin{aligned} \min_{\boldsymbol{\Delta}\mathbf{x}}\text{imize:} \quad & f(\bar{\mathbf{x}}) + \boldsymbol{\nabla} f(\bar{\mathbf{x}})^{\mathsf{T}} \boldsymbol{\Delta}\mathbf{x} \\ \text{subject to:} \quad & g_j(\bar{\mathbf{x}}) + \boldsymbol{\nabla} g_j(\bar{\mathbf{x}})^{\mathsf{T}} \boldsymbol{\Delta}\mathbf{x} \leq 0, \quad j = 1,\ldots,m_i \\ & h_j(\bar{\mathbf{x}}) + \boldsymbol{\nabla} h_j(\bar{\mathbf{x}})^{\mathsf{T}} \boldsymbol{\Delta}\mathbf{x} = 0, \quad j = 1,\ldots,m_e \\ & -\delta_i \leq \Delta x_i \leq \delta_i, \quad i = 1,\ldots,n \end{aligned}$$

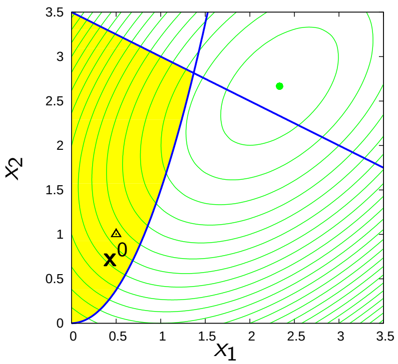
Problem: The LP-based search direction could be infeasible!

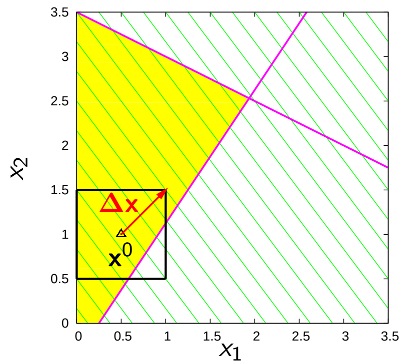
Constructing and Solving Direction-Finding LP

Class Exercise: Consider the optimization problem

$$\min_{\mathbf{x}} \ f(\mathbf{x}) \triangleq 2x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2$$
$$\text{s.t. } g_1(\mathbf{x}) \triangleq 3x_1^2 - 2x_2 \leq 0$$
$$g_2(\mathbf{x}) \triangleq x_1 + 2x_2 - 7 \leq 0$$

Formulate, then solve, the direction-finding LP at $\mathbf{x}^0 = (\frac{1}{2}, 1)^\mathsf{T}$, for $\delta = \frac{1}{2}$





LP-based Search Direction: Penalty Approach

- Feasibility of the LP-based search direction problem can be enforced via softening the constraints by penalization in the LP objective:

$$\underset{\mathbf{\Delta x},\mathbf{y},\mathbf{z}^\pm}{\text{minimize:}} \quad f(\bar{\mathbf{x}}) + \nabla f(\bar{\mathbf{x}})^\mathsf{T} \mathbf{\Delta x} + \mu \left[\sum_{j=1}^{m_\mathrm{i}} y_j + \sum_{j=1}^{m_\mathrm{e}} (z_j^+ + z_j^-) \right]$$

$$\text{subject to:} \quad g_j(\bar{\mathbf{x}}) + \nabla g_j(\bar{\mathbf{x}})^\mathsf{T} \mathbf{\Delta x} \leq y_j, \quad y_j \geq 0, \quad j = 1, \ldots, m_\mathrm{i}$$
$$h_j(\bar{\mathbf{x}}) + \nabla h_j(\bar{\mathbf{x}})^\mathsf{T} \mathbf{\Delta x} = z_j^+ - z_j^-, \quad z_j^+, z_j^- \geq 0, \quad j = 1, \ldots, m_\mathrm{e}$$
$$-\delta_i \leq \Delta x_i \leq \delta_i, \quad i = 1, \ldots, n$$

with $\mu > 0$ a suitable (large enough) penalty multiplier

SLP Algorithm — Minimize Problem

- **Step 0: Initialization**
  - Choose initial guess $\mathbf{x}^0$, initial step bound $\boldsymbol{\delta}^0$, penalty multiplier $\mu > 0$, scalars $0 < \rho_1 < \rho_2 < 1$ (e.g., $\rho_1 = 0.25$, $\rho_2 = 0.75$), step-bound adjustment parameter $0 < \beta < 1$ (e.g., $\beta = 0.5$), and stopping tolerance $\epsilon > 0$; set $k \leftarrow 0$
- **Step 1: LP-based Search Direction**
  - Compute gradients $\nabla f(\mathbf{x}^k)$, $\nabla g_i(\mathbf{x}^k)$ and $\nabla h_i(\mathbf{x}^k)$
  - Solve direction-finding LP,

$$\min_{\mathbf{\Delta x},\mathbf{y},\mathbf{z}^\pm} \ L \triangleq \nabla f(\mathbf{x}^k)^\mathsf{T} \mathbf{\Delta x} + \mu \left[\sum_{j=1}^{m_\mathrm{i}} y_j + \sum_{j=1}^{m_\mathrm{e}} (z_j^+ + z_j^-) \right]$$
$$\text{s.t. } g_j(\mathbf{x}^k) + \nabla g_j(\mathbf{x}^k)^\mathsf{T} \mathbf{\Delta x} \leq y_j, \quad j = 1, \ldots, m_\mathrm{i}$$
$$h_j(\mathbf{x}^k) + \nabla h_j(\mathbf{x}^k)^\mathsf{T} \mathbf{\Delta x} = z_j^+ - z_j^-, \quad j = 1, \ldots, m_\mathrm{e}$$
$$-\boldsymbol{\delta}^k \leq \mathbf{\Delta x} \leq \boldsymbol{\delta}^k, \quad \mathbf{y}, \mathbf{z}^\pm \geq \mathbf{0}$$

  to produce $\mathbf{\Delta x}^{k+1}$ and L^{k+1}

SLP Algorithm — Minimize Problem (cont'd)

- **Step 2: Stopping**
  - If $\mathbf{\Delta x}^{k+1} < \epsilon$, stop — report $\mathbf{x}^k$ (approximate KKT point)
- **Step 3: Step Sizes**
  - Compute $\Delta F^{k+1} = F(\mathbf{x}^k + \mathbf{\Delta x}^{k+1}) - F(\mathbf{x}^k)$, with

$$\text{Merit function:} \quad F(\mathbf{x}) \triangleq f(\mathbf{x}) + \mu \left[\sum_{j=1}^{m_\mathrm{i}} \max\{0, g_i(\mathbf{x})\} + \sum_{j=1}^{m_\mathrm{e}} |h_i(\mathbf{x})| \right]$$

  - If $\Delta F^{k+1} > 0$ (no improvement), shrink: $\boldsymbol{\delta}^k \leftarrow \beta \boldsymbol{\delta}^k$; return to step 1
  - If $\Delta F^{k+1} > \rho_1 L^{k+1}$ (small improvement), shrink: $\boldsymbol{\delta}^k \leftarrow \beta \boldsymbol{\delta}^k$
  - If $\Delta F^{k+1} < \rho_2 L^{k+1}$ (good improvement), expand: $\boldsymbol{\delta}^k \leftarrow \frac{1}{\beta} \boldsymbol{\delta}^k$
- **Step 4: Update**
  - Update $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{\Delta x}^{k+1}$; increment $k \leftarrow k + 1$; return to step 1

Pros and Cons of SLP

Pros:

- Functions well for mostly linear programs
- Converges quickly if the solution lies on the constraints
- Can rely on robust and efficient LP codes
- No need for computing/estimating second-order derivatives

Cons:

- Poor convergence for highly nonlinear programs
- Slow convergence to optimal points not at constraints (interior)
- Not available in general purpose modeling systems (GAMS, AMPL)

But,

- Used often in some industries (petrochemical)
- Available in commercial products tailored for specific applications in specific industries

Second-Order Methods

Goal: Incorporate second-order information to achieve faster convergence

- First, consider NLPs with equality constraints only:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n}\text{imize:} \quad & f(\mathbf{x}) \\ \text{subject to:} \quad & h_j(\mathbf{x}) = 0, \quad j = 1,\ldots,m_e \end{aligned}$$

- At a regular optimal point $\mathbf{x}^*$, there exist Lagrange multipliers $\boldsymbol{\lambda}^*$ such that

$$\mathbf{0} = \nabla\mathcal{L}(\mathbf{x}^*,\boldsymbol{\lambda}^*) = \begin{pmatrix} \nabla f(\mathbf{x}^*) - \sum_{j=1}^{m_e} \lambda_j^* \nabla h_j(\mathbf{x}^*) \\ \mathbf{h}(\mathbf{x}^*) \end{pmatrix}$$

where $\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}) \triangleq f(\mathbf{x}) - \sum_{j=1}^{m_e} \lambda_j h_j(\mathbf{x})$

Second-Order Methods (cont'd)

Idea: Solve the nonlinear system of $(n+m)$ equations using a Newton-like iterative method

- Newton's method to find $\mathbf{y} \in \mathbb{R}^n$ such that $\mathbf{F}(\mathbf{y}) = \mathbf{0}$:

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \nabla\mathbf{F}(\mathbf{y}^k)^{-1}\mathbf{F}(\mathbf{y}^k); \qquad \mathbf{y}^0 \text{ given}$$

- With $\mathbf{F} \triangleq \nabla\mathcal{L}$ and $\mathbf{y} \triangleq (\mathbf{x},\boldsymbol{\lambda})$,

$$\begin{pmatrix} \nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}^k,\boldsymbol{\lambda}^k) & -\nabla\mathbf{h}(\mathbf{x}^k)^\mathsf{T} \\ \nabla\mathbf{h}(\mathbf{x}^k) & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta}\mathbf{x}^{k+1} \\ \boldsymbol{\lambda}^{k+1} \end{pmatrix} = - \begin{pmatrix} \nabla f(\mathbf{x}^k) \\ \mathbf{h}(\mathbf{x}^k) \end{pmatrix}$$

where $\boldsymbol{\Delta}\mathbf{x}^{k+1} \triangleq \mathbf{x}^{k+1} - \mathbf{x}^k$

- **But**, no distinction between local minima and local maxima!

Quadratic Programming

Quadratic Programs

A constrained nonlinear program is a quadratic program, or QP, if its objective function is quadratic and *all* its constraints are linear:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n}\text{imize:} \quad & \mathbf{c}^\mathsf{T}\mathbf{x} + \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{Q}\mathbf{x} \\ \text{subject to:} \quad & \mathbf{A}_\mathrm{i}\mathbf{x} \leq \mathbf{b}_\mathrm{i} \\ & \mathbf{A}_\mathrm{e}\mathbf{x} = \mathbf{b}_\mathrm{e} \end{aligned}$$

with $\mathbf{Q} \in \mathbb{R}^{n\times n}$, $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{A}_\mathrm{i} \in \mathbb{R}^{m_\mathrm{i}\times n}$, $\mathbf{b}_\mathrm{i} \in \mathbb{R}^{m_\mathrm{i}}$, $\mathbf{A}_\mathrm{e} \in \mathbb{R}^{m_\mathrm{e}\times n}$, $\mathbf{b}_\mathrm{e} \in \mathbb{R}^{m_\mathrm{e}}$

- QPs are [strictly] convex programs provided that the matrix $\mathbf{Q}$ in the objective function is positive semi-definite [positive definite]
- Like LPs, powerful and reliable techniques/codes are available to solve convex QPs, including very large-scale QPs

Search Direction: QP-based Approach

- Solutions $(\boldsymbol{\Delta}\mathbf{x}^{k+1}, \boldsymbol{\lambda}^{k+1})$ to the direction-finding system

$$\begin{pmatrix} \nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}^k,\boldsymbol{\lambda}^k) & -\nabla\mathbf{h}(\mathbf{x}^k)^\mathsf{T} \\ \nabla\mathbf{h}(\mathbf{x}^k) & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\Delta}\mathbf{x}^{k+1} \\ \boldsymbol{\lambda}^{k+1} \end{pmatrix} = -\begin{pmatrix} \nabla f(\mathbf{x}^k) \\ \mathbf{h}(\mathbf{x}^k) \end{pmatrix}$$

exactly match stationary points to the Lagrangian of QP

$$\begin{aligned} \min_{\boldsymbol{\Delta}\mathbf{x}}\text{imize:} \quad & \nabla f(\mathbf{x}^k)^\mathsf{T}\boldsymbol{\Delta}\mathbf{x} + \frac{1}{2}\boldsymbol{\Delta}\mathbf{x}^\mathsf{T}\nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}^k,\boldsymbol{\lambda}^k)\boldsymbol{\Delta}\mathbf{x} \\ \text{subject to:} \quad & h_j(\mathbf{x}^k) + \nabla h_j(\mathbf{x}^k)^\mathsf{T}\boldsymbol{\Delta}\mathbf{x} = 0, \quad j = 1,\ldots,m_\mathrm{e} \end{aligned}$$

with $\boldsymbol{\lambda}^{k+1}$ corresponding to the QP Lagrange multipliers

> Solution of this linear system provides: (i) the search direction $\boldsymbol{\Delta}\mathbf{x}^{k+1}$ at $\mathbf{x}^k$; (ii) estimates $\boldsymbol{\lambda}^{k+1}$ of the Lagrange multipliers

Search Direction: Problems with Inequality Constraints

- Consider the general NLP:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n}\text{imize:} \quad & f(\mathbf{x}) \\ \text{subject to:} \quad & g_j(\mathbf{x}) \leq 0, \quad j = 1,\ldots,m_\mathrm{i} \\ & h_j(\mathbf{x}) = 0, \quad j = 1,\ldots,m_\mathrm{e} \end{aligned}$$

- The search direction $\boldsymbol{\Delta}\mathbf{x}^{k+1}$ at $\mathbf{x}^k$ can be obtained from:

$$\begin{aligned} \min_{\boldsymbol{\Delta}\mathbf{x}}\text{imize:} \quad & \nabla f(\mathbf{x}^k)^\mathsf{T}\boldsymbol{\Delta}\mathbf{x} + \frac{1}{2}\boldsymbol{\Delta}\mathbf{x}^\mathsf{T}\nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}^k,\boldsymbol{\nu}^k,\boldsymbol{\lambda}^k)\boldsymbol{\Delta}\mathbf{x} \\ \text{subject to:} \quad & g_j(\mathbf{x}^k) + \nabla g_j(\mathbf{x}^k)^\mathsf{T}\boldsymbol{\Delta}\mathbf{x} \leq 0, \quad j = 1,\ldots,m_\mathrm{i} \\ & h_j(\mathbf{x}^k) + \nabla h_j(\mathbf{x}^k)^\mathsf{T}\boldsymbol{\Delta}\mathbf{x} = 0, \quad j = 1,\ldots,m_\mathrm{e} \end{aligned}$$

with $\mathcal{L}(\mathbf{x},\boldsymbol{\nu},\boldsymbol{\lambda}) \triangleq f(\mathbf{x}) - \boldsymbol{\nu}^\mathsf{T}\mathbf{g}(\mathbf{x}) - \boldsymbol{\lambda}^\mathsf{T}\mathbf{h}(\mathbf{x})$

- Estimates $\boldsymbol{\lambda}^{k+1}, \boldsymbol{\nu}^{k+1}$ of the Lagrange/KKT multipliers correspond to the QP Lagrange/KKT multipliers

Constructing and Solving Direction-Finding Problem

Class Exercise: Consider the optimization problem

$$\begin{aligned} \min_{\mathbf{x}} \; & f(\mathbf{x}) \triangleq 2x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2 \\ \text{s.t.} \; & g_1(\mathbf{x}) \triangleq 3x_1^2 - 2x_2 \leq 0 \\ & g_2(\mathbf{x}) \triangleq x_1 + 2x_2 - 7 \leq 0 \end{aligned}$$

Formulate, then solve, the direction-finding QP problem at $\mathbf{x}^0 = (\frac{1}{2}, 1)^\mathsf{T}$

$$\begin{aligned} \min_{\boldsymbol{\Delta}\mathbf{x}} \quad & \begin{pmatrix} 4x_1^0 - 2x_2^0 - 4 \\ -2x_1^0 + 4x_2^0 - 6 \end{pmatrix}^\mathsf{T}\boldsymbol{\Delta}\mathbf{x} + \frac{1}{2}\boldsymbol{\Delta}\mathbf{x}^\mathsf{T}\begin{pmatrix} 4 - 6\nu_1^0 & -2 \\ -2 & 4 \end{pmatrix}\boldsymbol{\Delta}\mathbf{x} \\ \text{s.t.} \quad & 3(x_1^0)^2 - 2x_2^0 + \begin{pmatrix} 6x_1^0 \\ -2 \end{pmatrix}^\mathsf{T}\boldsymbol{\Delta}\mathbf{x} \leq 0 \\ & x_1^0 + 2x_2^0 - 7 + \begin{pmatrix} 1 \\ 2 \end{pmatrix}^\mathsf{T}\boldsymbol{\Delta}\mathbf{x} \leq 0 \end{aligned}$$

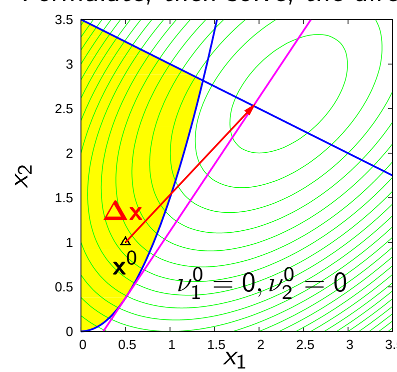
- The QP depends on the KKT multiplier ν_1 associated to g_1

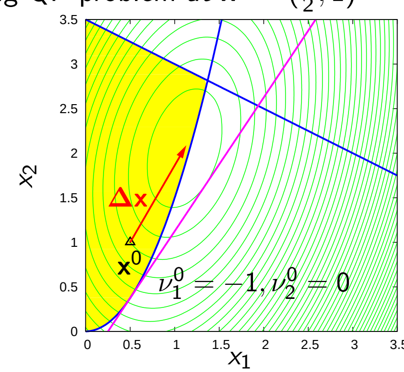
Constructing and Solving Direction-Finding Problem

Class Exercise: Consider the optimization problem

$$\begin{aligned} \min_{\mathbf{x}} \; & f(\mathbf{x}) \triangleq 2x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2 \\ \text{s.t.} \; & g_1(\mathbf{x}) \triangleq 3x_1^2 - 2x_2 \leq 0 \\ & g_2(\mathbf{x}) \triangleq x_1 + 2x_2 - 7 \leq 0 \end{aligned}$$

Formulate, then solve, the direction-finding QP problem at $\mathbf{x}^0 = (\frac{1}{2}, 1)^\mathsf{T}$





Sequential Quadratic Programming Method

- Follows the improving-search paradigm
- Update search direction $\Delta\mathbf{x}^{k+1}$ repeatedly via the solution of a QP subproblem
- Linesearch can be performed along a given direction by using a suitable merit function that measures progress — Typical choice:

$$F(\mathbf{x},\mu) \triangleq f(\mathbf{x}) + \mu\left[\sum_{j=1}^{m_\mathrm{i}} \max\{0, g_i(\mathbf{x})\} + \sum_{j=1}^{m_\mathrm{e}} |h_i(\mathbf{x})|\right]$$

  with a suitable penalty multiplier $\mu > 0$
- Possibility to construct an approximation $\mathbf{D}^k$ of the second-order derivatives $\nabla^2_{\mathbf{xx}}\mathcal{L}(\mathbf{x}^k, \boldsymbol{\nu}^k, \boldsymbol{\lambda}^k)$ — E.g., based on a BFGS recursive scheme
  - Positive definiteness of $\mathbf{D}^k$ provides robustness
  - Reduces computational effort

SQP Algorithm — Minimize Problem

- **Step 0: Initialization**
  - Choose initial guess $\mathbf{x}^0$, initial multipliers $\boldsymbol{\lambda}^0$ and $\boldsymbol{\nu}^0 \geq \mathbf{0}$, positive definite matrix $\mathbf{D}^0$, penalty multiplier $\mu > 0$, and stopping tolerance $\epsilon > 0$; set $k \leftarrow 0$
- **Step 1: QP-based Search Direction**
  - Compute gradients $\nabla f(\mathbf{x}^k)$, $\nabla g_i(\mathbf{x}^k)$ and $\nabla h_i(\mathbf{x}^k)$
  - Solve direction-finding QP,

$$\begin{aligned}\min_{\Delta\mathbf{x}}\quad & \nabla f(\mathbf{x}^k)^\mathsf{T}\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^\mathsf{T}\mathbf{D}^k\Delta\mathbf{x}\\ \text{s.t.}\quad & g_j(\mathbf{x}^k) + \nabla g_j(\mathbf{x}^k)^\mathsf{T}\Delta\mathbf{x} \leq 0, \quad j = 1,\ldots,m_\mathrm{i}\\ & h_j(\mathbf{x}^k) + \nabla h_j(\mathbf{x}^k)^\mathsf{T}\Delta\mathbf{x} = 0, \quad j = 1,\ldots,m_\mathrm{e}\end{aligned}$$

    to produce $\Delta\mathbf{x}^{k+1}$, $\boldsymbol{\lambda}^{k+1}$ and $\boldsymbol{\nu}^{k+1}$
- **Step 2: Stopping**
  - If $\Delta\mathbf{x}^{k+1} < \epsilon$, stop — report $\mathbf{x}^k$ (approximate KKT point)

SQP Algorithm — Minimize Problem (cont'd)

- **Step 3: Linesearch**
  - Solve 1-d linesearch problem (at least approximately), $\min_{\alpha\geq 0}\ell(\alpha) \triangleq F(\mathbf{x}^k + \alpha\Delta\mathbf{x}^{k+1}, \mu)$, to compute the step α^{k+1}
- **Step 4: Update**
  - Iterate: $\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \alpha^{k+1}\Delta\mathbf{x}^{k+1}$
  - BFGS: $\mathbf{D}^{k+1} \leftarrow \mathbf{D}^k + \frac{\mathbf{g}\mathbf{g}^\mathsf{T}}{\mathbf{g}^\mathsf{T}\mathbf{d}} - \frac{\mathbf{D}^k\mathbf{d}\mathbf{d}^\mathsf{T}\mathbf{D}^k}{\mathbf{d}^\mathsf{T}\mathbf{D}^k\mathbf{d}}$, with $\mathbf{d} = \mathbf{x}^{k+1} - \mathbf{x}^k$, $\mathbf{g} = \nabla\mathcal{L}(\mathbf{x}^{k+1}, \boldsymbol{\nu}^{k+1}, \boldsymbol{\lambda}^{k+1}) - \nabla\mathcal{L}(\mathbf{x}^k, \boldsymbol{\nu}^{k+1}, \boldsymbol{\lambda}^{k+1})$
  - Increment $k \leftarrow k+1$ and return to step 1

- SQP usually much faster and more reliable than first-order methods
  - Analytical derivatives highly recommended for reliability
  - Method of choice for optimization of complex, first-principle models
- Available in general purpose modeling systems (GAMS, AMPL)
  - Use within a modeling manager recommended
  - Often need to adjust parameters for good performance (more tuning!)
- Used routinely in engineering optimization products