Non-Linear Programming (NLP): Multivariable, Constrained

Benoît Chachuat <benoit@mcmaster.ca>
McMaster University
Department of Chemical Engineering

ChE 4G03: Optimization in Chemical Engineering

Outline

This lesson

Constrained Optimization

Numerical Solution Methods

Sequential Unconstrained Prog.
Sequential Linear Prog. (SLP)
Sequential Quadratic Prog. (SQP)
Generalized Reduced Gradient (GRG)

Analytical Solution Methods

Methods for Multivariable Constrained Optimization

Basic Concepts for Optimization – Part I
Basic Concepts for Optimization – Part II
Methods for Multivariable Unconstrained Optim.
Basic Concepts for Optimization – Part III

Penalty Methods

Idea: Transform a constrained NLP into an unconstrained NLP

Consider the NLP problem

\[
\begin{align*}
\text{minimize: } & \ f(x) \\
\text{subject to: } & \ g_j(x) \leq 0, \quad j = 1, \ldots, m_i \\
& \ h_j(x) = 0, \quad j = 1, \ldots, m_e
\end{align*}
\]

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

\[
\begin{align*}
\text{minimize: } \ F(x) & \triangleq f(x) + \mu \left[ \sum_{j=1}^{m_e} p_j^e(x) + \sum_{j=1}^{m_i} p_j^i(x) \right]
\end{align*}
\]

with \( \mu > 0 \) the penalty multiplier; \( F \), the auxiliary function

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

**Inequality Constraints:** \(g_j(x) \leq 0\)

\[
\begin{align*}
    p_j^f(x) &= 0, \text{ if } g_j(x) \leq 0 \\
    p_j^f(x) &= > 0, \text{ otherwise}
\end{align*}
\]

**Equality Constraints:** \(h_j(x) = 0\)

\[
\begin{align*}
    p_j^f(x) &= 0, \text{ if } h_j(x) = 0 \\
    p_j^f(x) &= > 0, \text{ otherwise}
\end{align*}
\]

**Common Choices:**

\[
p_j^f(x) \triangleq \max\{0, g_j(x)\}^\gamma, \gamma \geq 1
\]

**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 \(\mu\), the corresponding penalty function is said to be exact.
- If no such finite value of \(\mu\) exists, it is said to be inexact (yields an optimum as \(\mu \to \infty\) only).

**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:** **Sequential Unconstrained Penalty Algorithm**

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

**Constructing and Solving Penalty Models**

**Class Exercise:** Consider the optimization problem

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

- Solve this problem by inspection
- Construct a penalty model using a square penalty function, then solve the unconstrained NLP as a function of the penalty multiplier \(\mu\)
Sequential Unconstrained Penalty Algorithm

**Step 0: Initialization**
- Form penalty model; choose initial guess $x^0$, 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 $x^k$, solve penalty optimization problem

$$
\min_{x} F(x) \overset{\Delta}{=} f(x) + \mu \left[ \sum_{j=1}^{m_n} p_n^j(x) + \sum_{j=1}^{m_i} p_i^j(x) \right],
$$

with $\mu = \mu^k$, to produce $x^{k+1}$

**Step 2: Stopping**
- If $\mu^k \left[ \sum_{j=1}^{m_n} p_n^j(x^{k+1}) + \sum_{j=1}^{m_i} p_i^j(x^{k+1}) \right] < \epsilon$, stop — report $x^{k+1}$ (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

$$
\min_{x \in \mathbb{R}^n} f(x)
$$

subject to: $g_j(x) \leq 0$, $j = 1, \ldots, m_i$

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

$$
\min_{x \in \mathbb{R}^n} F(x) \overset{\Delta}{=} f(x) + \mu \sum_{j=1}^{m_i} b_j(x), \quad b_j(x) \overset{\Delta}{=} \frac{g_j(x)}{\mu} + \infty,
$$

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

Barrier Functions for Inequality Constrained NLPs

**Ideal Barrier Function**: $g_j(x) \leq 0$

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

**Common Barrier Functions**:

$$
\begin{align*}
    b_j(x) &\overset{\Delta}{=} -\frac{1}{g_j(x)}, \\
    b_j(x) &\overset{\Delta}{=} -\ln(-g_j(x))
\end{align*}
$$

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 $\mu \downarrow 0$, 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

$$
\min_{x} f(x) \overset{\Delta}{=} x
$$

s.t. $g(x) \overset{\Delta}{=} 2 - x \leq 0$

- Construct a barrier model using the inverse barrier function, then solve the unconstrained NLP as a function of the barrier multiplier $\mu$
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

Sequential Unconstrained Barrier Algorithm

Step 0: Initialization
- Form barrier model; choose initial guess $x^0$, barrier multiplier $\mu^0 > 0$, reduction factor $0 < \beta < 1$, and stopping tolerance $\epsilon > 0$; set $k ← 0$

Step 1: Unconstrained Optimization
- Direction: Starting from $x^k$, solve barrier optimization problem
  $$\min_{x} F(x) \triangleq f(x) + \mu \sum_{j=1}^{m_i} b_j(x),$$
  with $\mu = \mu^k$, to produce $x^{k+1}$

Step 2: Stopping
- If $\mu^k \sum_{j=1}^{m_i} b_j(x^{k+1}) < \epsilon$, stop — report $x^{k+1}$ (approximate KKT point)

Step 3: Update
- Decrease the penalty parameter as $\mu^{k+1} ← \beta \mu^k$
- Increment $k ← k + 1$ 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” $\pm \delta$
- The common approach is to bound the direction components with a “box” (or hypercube)

Problem: The LP-based search direction could be infeasible!
Constructing and Solving Direction-Finding LP

**Class Exercise:** Consider the optimization problem

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

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

![Diagram of the optimization problem and its feasible region]

### SLP Algorithm — Minimize Problem

- **Step 0: Initialization**
  - Choose initial guess \(x^0\), initial step bound \(\delta^0\), 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 \(\nabla f(x^k)\), \(\nabla g_i(x^k)\) and \(\nabla h_j(x^k)\)
  - Solve direction-finding LP,
    \[
    \min_{\Delta x, y, z} L \triangleq \nabla f(x^k)^T \Delta x + \mu \left[ \sum_{j=1}^{m_i} y_j + \sum_{j=1}^{m_c} (z^+_j + z^-_j) \right]
    \text{s.t. } g_i(x^k) + \nabla g_i(x^k)^T \Delta x \leq y_j, \quad j = 1, \ldots, m_i \\
h_j(x^k) + \nabla h_j(x^k)^T \Delta x = z^+_j - z^-_j, \quad j = 1, \ldots, m_c \\
- \delta^k \leq \Delta x \leq \delta^k, \quad y, z^\pm \geq 0
    \]
    to produce \(\Delta x^{k+1}\) and \(L^{k+1}\)

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:

\[
\text{minimize: } f(\bar{x}) + \nabla f(\bar{x})^T \Delta x + \mu \left[ \sum_{j=1}^{m_i} y_j + \sum_{j=1}^{m_c} (z^+_j + z^-_j) \right]
\]

subject to:
\[
\begin{align*}
g_i(\bar{x}) + \nabla g_i(\bar{x})^T \Delta x &\leq y_j, \quad j = 1, \ldots, m_i \\
h_j(\bar{x}) + \nabla h_j(\bar{x})^T \Delta x &= z^+_j - z^-_j, \quad j = 1, \ldots, m_c \\
- \delta_i \leq \Delta x_i &\leq \delta_i, \quad i = 1, \ldots, n
\end{align*}
\]

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

### SLP Algorithm — Minimize Problem (cont’d)

- **Step 2: Stopping**
  - If \(\Delta x^{k+1} < \epsilon\), stop — report \(x^k\) (approximate KKT point)

- **Step 3: Step Sizes**
  - Compute \(\Delta F^{k+1} = F(x^k + \Delta x^{k+1}) - F(x^k)\), with

  **Merit function:** \(F(x) \triangleq f(x) + \mu \left[ \sum_{j=1}^{m_i} \max\{0, g_i(x)\} + \sum_{j=1}^{m_c} |h_i(x)| \right]\)

    - If \(\Delta F^{k+1} > 0\) (no improvement), shrink: \(\delta^k \leftarrow 0.75\delta^k\); return to step 1
    - If \(\Delta F^{k+1} > \rho_1 L^{k+1}\) (small improvement), shrink: \(\delta^k \leftarrow \beta \delta^k\)
    - If \(\Delta F^{k+1} < \rho_2 L^{k+1}\) (good improvement), expand: \(\delta^k \leftarrow 1.25\delta^k\)

- **Step 4: Update**
  - Update \(x^{k+1} = x^k + \Delta x^{k+1}\); 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:
minimize: \( f(x) \)
subject to: \( h_j(x) = 0, \quad j = 1, \ldots, m_e \)

At a regular optimal point \( x^* \), there exist Lagrange multipliers \( \lambda^* \) such that
\[
0 = \nabla L(x^*, \lambda^*) = \begin{pmatrix}
\nabla f(x^*) - \sum_{j=1}^{m_e} \lambda_j^* \nabla h_j(x^*) \\
h(x^*)
\end{pmatrix}
\]
where \( L(x, \lambda) \triangleq f(x) - \sum_{j=1}^{m_e} \lambda_j h_j(x) \)

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 \( y \in \mathbb{R}^n \) such that \( F(y) = 0 \):
  \[
y^{k+1} = y^k - \nabla F(y^k)^{-1} F(y^k); \quad y^0 \text{ given}
\]

- With \( F \triangleq \nabla L \) and \( y \triangleq (x, \lambda) \),

  \[
  \begin{pmatrix}
  \nabla^2_{xx} L(x^k, \lambda^k) & -\nabla h(x^k) \\
  \nabla h(x^k) & 0
  \end{pmatrix}
  \begin{pmatrix}
  \Delta x^{k+1} \\
  \lambda^{k+1}
  \end{pmatrix}
  =
  -\begin{pmatrix}
  \nabla f(x^k) \\
  h(x^k)
  \end{pmatrix}
\]

  where \( \Delta x^{k+1} \triangleq x^{k+1} - x^k \)

  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:

minimize: \( c^T x + \frac{1}{2} x^T Q x \)
subject to: \( A_i x \leq b_i \)
\( A_e x = b_e \)

with \( Q \in \mathbb{R}^{n \times n}, \ c \in \mathbb{R}^n, \ A_i \in \mathbb{R}^{m_i \times n}, \ b_i \in \mathbb{R}^{m_i}, \ A_e \in \mathbb{R}^{m_e \times n}, \ b_e \in \mathbb{R}^{m_e} \)

- QPs are [strictly] convex programs provided that the matrix \( Q \) 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 \((\Delta x^{k+1}, \lambda^{k+1})\) to the direction-finding system

\[
\begin{pmatrix}
\nabla_x^2 L(x^k, \lambda^k) - \nabla h(x^k)^T \\
\n\nabla h(x^k)
\end{pmatrix}
\begin{pmatrix}
\Delta x^{k+1} \\
\lambda^{k+1}
\end{pmatrix}
= -\begin{pmatrix}
\nabla f(x^k) \\
0
\end{pmatrix}
\]

exactly match stationary points to the Lagrangian of QP

- minimize: \(\nabla f(x^k)^T \Delta x + \frac{1}{2} \Delta x^T \nabla_x^2 L(x^k, \lambda^k) \Delta x\)
- subject to: \(h_j(x^k) + \nabla h_j(x^k)^T \Delta x = 0, \ j = 1, \ldots, m_e\)

with \(\lambda^{k+1}\) corresponding to the QP Lagrange multipliers

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

Constructing and Solving Direction-Finding Problem

Class Exercise: Consider the optimization problem

\[
\min_x f(x) \triangleq 2x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2
\]

s.t. \(g_1(x) \triangleq 3x_1^2 - 2x_2 \leq 0\)

\(g_2(x) \triangleq x_1 + 2x_2 - 7 \leq 0\)

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

\[
\begin{pmatrix}
4x_1^0 - 2x_2^0 - 4 \\
-2x_1^0 + 4x_2^0 - 6
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta x
\end{pmatrix}
+ \frac{1}{2} \begin{pmatrix}
4 & -6 \\
-2 & 4
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta x
\end{pmatrix}
\begin{pmatrix}
3(x_1^0)^2 - 2x_2^0 + (6x_1^0) \\
2 - 2
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta x
\end{pmatrix}
\leq 0
\]

\[
\begin{pmatrix}
x_1^0 + 2x_2^0 - 7 + \begin{pmatrix}
1 \\
2
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta x
\end{pmatrix}
\leq 0
\end{pmatrix}
\]

- The QP depends on the KKT multiplier \(\nu_1\) associated to \(g_1\)

Search Direction: Problems with Inequality Constraints

- Consider the general NLP:

\[
\begin{align*}
\text{minimize:} & \quad f(x) \\
\text{subject to:} & \quad g_i(x) \leq 0, \quad j = 1, \ldots, m_i \\
& \quad h_j(x) = 0, \quad j = 1, \ldots, m_e
\end{align*}
\]

- The search direction \(\Delta x^{k+1}\) at \(x^k\) can be obtained from:

\[
\begin{align*}
\text{minimize:} & \quad \nabla f(x^k)^T \Delta x + \frac{1}{2} \Delta x^T \nabla_x^2 L(x^k, \nu^k, \lambda^k) \Delta x \\
\text{subject to:} & \quad g_j(x^k) + \nabla g_j(x^k)^T \Delta x \leq 0, \quad j = 1, \ldots, m_i \\
& \quad h_j(x^k) + \nabla h_j(x^k)^T \Delta x = 0, \quad j = 1, \ldots, m_e
\end{align*}
\]

with \(L(x, \nu, \lambda) \triangleq f(x) - \nu^T g(x) - \lambda^T h(x)\)

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

Constructing and Solving Direction-Finding Problem

Class Exercise: Consider the optimization problem

\[
\min_x f(x) \triangleq 2x_1^2 + 2x_2^2 - 2x_1x_2 - 4x_1 - 6x_2
\]

s.t. \(g_1(x) \triangleq 3x_1^2 - 2x_2 \leq 0\)

\(g_2(x) \triangleq x_1 + 2x_2 - 7 \leq 0\)

Formulate, then solve, the direction-finding QP problem at \(x^0 = (\frac{1}{2}, 1)^T\)
Sequential Quadratic Programming Method

- Follows the improving-search paradigm
- Update search direction $\Delta x^{k+1}$ 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:
  
  \[
  F(x, \mu) \triangleq f(x) + \mu \left[ \sum_{j=1}^{m} \max\{0, g_i(x)\} + \sum_{j=1}^{m_e} |h_i(x)| \right]
  \]
  
  with a suitable penalty multiplier $\mu > 0$
- Possibility to construct an approximation $D^k$ of the second-order derivatives $\nabla^2 xx L(x^k, \nu^k, \lambda^k)$ — E.g., based on a BFGS recursive scheme
  
  - Positive definiteness of $D^k$ provides robustness
  - Reduces computational effort

SQP Algorithm — Minimize Problem

**Step 0: Initialization**
- Choose initial guess $x^0$, initial multipliers $\lambda^0$ and $\nu^0 > 0$, positive definite matrix $D^0$, penalty multiplier $\mu > 0$, and stopping tolerance $\epsilon > 0$; set $k \leftarrow 0$

**Step 1: QP-based Search Direction**
- Compute gradients $\nabla f(x^k)$, $\nabla g_i(x^k)$ and $\nabla h_i(x^k)$
- Solve direction-finding QP,
  
  \[
  \min_{\Delta x} \nabla f(x^k)^T \Delta x + \frac{1}{2} \Delta x^T D^k \Delta x
  \]
  
  s.t. $g_j(x^k) + \nabla g_j(x^k)^T \Delta x \leq 0$, $j = 1, \ldots, m_i$
  
  $h_j(x^k) + \nabla h_j(x^k)^T \Delta x = 0$, $j = 1, \ldots, m_e$

  to produce $\Delta x^{k+1}$, $\lambda^{k+1}$ and $\nu^{k+1}$

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

**Step 3: Linesearch**
- Solve 1-d linesearch problem (at least approximately),
  
  $\min_{\alpha \geq 0} \ell(\alpha) \triangleq F(x^k + \alpha \Delta x^{k+1}, \mu)$, to compute the step $\alpha^{k+1}$

**Step 4: Update**
- Iterate: $x^{k+1} \leftarrow x^k + \alpha^{k+1} \Delta x^{k+1}$
- BFGS: $D^{k+1} \leftarrow D^k + \frac{s^T d}{s^T d} - \frac{D^k s d^T}{d^T D^k d}$, with $d = x^{k+1} - x^k$, $g = \nabla L(x^{k+1}, \nu^{k+1}, \lambda^{k+1}) - \nabla L(x^k, \nu^{k+1}, \lambda^{k+1})$
- 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