Nonlinear Programming:

Constrained NLP and IPOPT Solver

Assoc. Prof. Karl Ezra Pilario, Ph.D.

Process Systems Engineering Laboratory
Department of Chemical Engineering
University of the Philippines Diliman

Outline

- Introduction to NLP
- Necessary and Sufficient Conditions for Optimality
- Convex Programming
- Methods for Solving NLP
    - One-Dimensional, Unconstrained NLP
    - Multivariable, Unconstrained NLP
    - Zero-order, First-order, **Second-order Methods**
    - Constrained NLP

A Taxonomy of NLP Solvers

Zero-order Methods

Derivative-free, black-box

- Grid Search / Exhaustive Search
- Random Search
- Nelder-Mead Simplex

- Metaheuristic Search
    - Genetic Algorithms
    - Particle Swarm
    - Simulated Annealing
    - Differential Evolution
    - CMAES

- Bayesian Optimization / Surrogate-based

First-order Methods

Uses 1^{st} derivative information

- Steepest Descent or Gradient Descent
- Stochastic Gradient Descent (SGD)
    - RMSProp, Adam, etc.
- Conjugate Gradient methods (e.g. Fletcher-Reeves)
- Coordinate Descent

Second-order Methods

Uses 1^{st} and 2^{nd} derivative information

- *Newton's method*
- *Quasi-newton method (BFGS)*

- Levenberg-Marquardt
- Trust Region Newton Methods
- Active Set and Interior-Point Methods
- Sequential Quadratic Programming (SQP)
- IPOPT

Multivariable, Unconstrained NLP

- Newton's method
- Quasi-newton Method (BFGS)
- **Levenberg-Marquardt**
- Trust Region Newton Methods
- Active Set and Interior-Point Methods
- Sequential Quadratic Programming (SQP)
- IPOPT

Levenberg-Marquardt Method

$$\min f(\boldsymbol{x}_k)$$

This is the Newton update:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k [\boldsymbol{H}(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

Norm over all data points, $i = 1, 2, \dots, N$

$$\min f(\boldsymbol{x}_k) = \frac{1}{2} \|r(\boldsymbol{x})\|^2$$

Least-squares objective

$$\nabla f(\boldsymbol{x}_k) = [\boldsymbol{J}_r(\boldsymbol{x}_k)]^T r(\boldsymbol{x})$$

where $[\boldsymbol{J}_r(\boldsymbol{x}_k)]_{ij} = \left[\frac{\partial r_i}{\partial x_j}\right]$ (Jacobian)

$$\boldsymbol{H}(\boldsymbol{x}_k) = \boldsymbol{J}_r{}^T \boldsymbol{J}_r + \sum_i r_i(\boldsymbol{x}_k) \nabla^2 r_i(\boldsymbol{x}_k) \quad \text{(Hessian)}$$

(Gauss-Newton algorithm)

Use $\boldsymbol{H}(\boldsymbol{x}_k) \approx \boldsymbol{J}_r{}^T \boldsymbol{J}_r$

(Levenberg-Marquardt algorithm)

Use $\boldsymbol{H}(\boldsymbol{x}_k) \approx \boldsymbol{J}_r{}^T \boldsymbol{J}_r + \beta \boldsymbol{I}$ (Levenberg)

or $\boldsymbol{H}(\boldsymbol{x}_k) \approx \boldsymbol{J}_r{}^T \boldsymbol{J}_r + \beta \operatorname{diag}(\boldsymbol{J}_r{}^T \boldsymbol{J}_r)$ (Marquardt)

Ensures that $\boldsymbol{H}(\boldsymbol{x}_k)$ is positive-definite and well-conditioned.

β = damping parameter

*The LM algorithm is useful in least-squares curve fitting and parameter estimation.

Steps:

- Make an initial guess, $\boldsymbol{x}_0$
- Solve the search direction, $\boldsymbol{s}_k$:

$$(\boldsymbol{J}_r{}^T \boldsymbol{J}_r + \beta \boldsymbol{I}) \boldsymbol{s}_k = -[\boldsymbol{J}_r(\boldsymbol{x}_k)]^T r(\boldsymbol{x})$$

- Calculate step size α by minimizing:

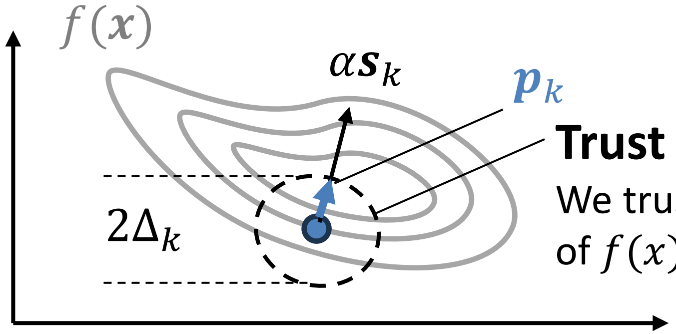
$$g_k(\alpha) = f(\boldsymbol{x}_k + \alpha \boldsymbol{s}_k)$$

- Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha \boldsymbol{s}_k$

Multivariable, Unconstrained NLP

- Newton's method
- Quasi-newton Method (BFGS)
- Levenberg-Marquardt
- **Trust Region Newton Methods**
- Active Set and Interior-Point Methods
- Sequential Quadratic Programming (SQP)

Trust Region Newton Methods



Trust region

We trust a quadratic approximation of $f(x)$ within a ball of radius Δ.

$$\min f(\boldsymbol{x}_k) \implies \min_{\boldsymbol{p}_k} f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k + \frac{1}{2}\boldsymbol{p}_k^T \boldsymbol{H}_k \boldsymbol{p}_k$$

Trust region subproblem

$$\text{such that} \quad \|\boldsymbol{p}_k\| \le \Delta_k$$

Approx. Newton solution: $\boldsymbol{p}_k = -[\boldsymbol{H}(\boldsymbol{x}_k)]^{-1}\nabla f(\boldsymbol{x}_k)$

Approx. Steepest descent solution: $\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k)$

Step 1: Make an initial guess, $\boldsymbol{x}_0$.

Step 2: Solve the trust-region subproblem to get $\boldsymbol{p}_k$. → e.g. Dogleg method

Step 3: Check if the approximation is good:

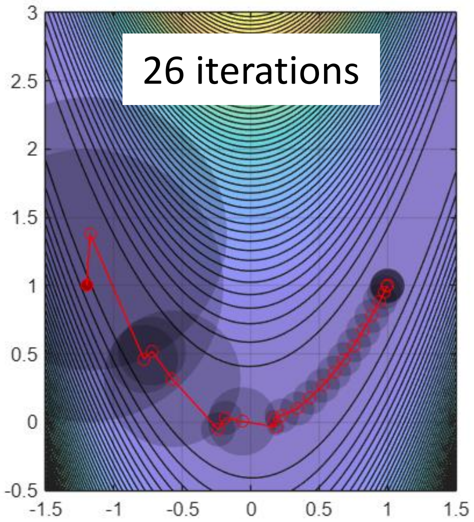
- If yes, accept $\boldsymbol{p}_k$ and maybe *enlarge* Δ_k.
- If no, reject $\boldsymbol{p}_k$ and *shrink* Δ_k.

Step 4: Update: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k$

Example:

Minimize $f(\boldsymbol{x})$ from $\boldsymbol{x}_0 = (-1.2, 1.0)$:

$$f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



Outline

- Introduction to NLP
- Necessary and Sufficient Conditions for Optimality
- Convex Programming
- Methods for Solving NLP
    - One-Dimensional, Unconstrained NLP
    - Multivariable, Unconstrained NLP
    - Zero-order, First-order, Second-order Methods
    - **Constrained NLP**

Multivariable, Constrained NLP

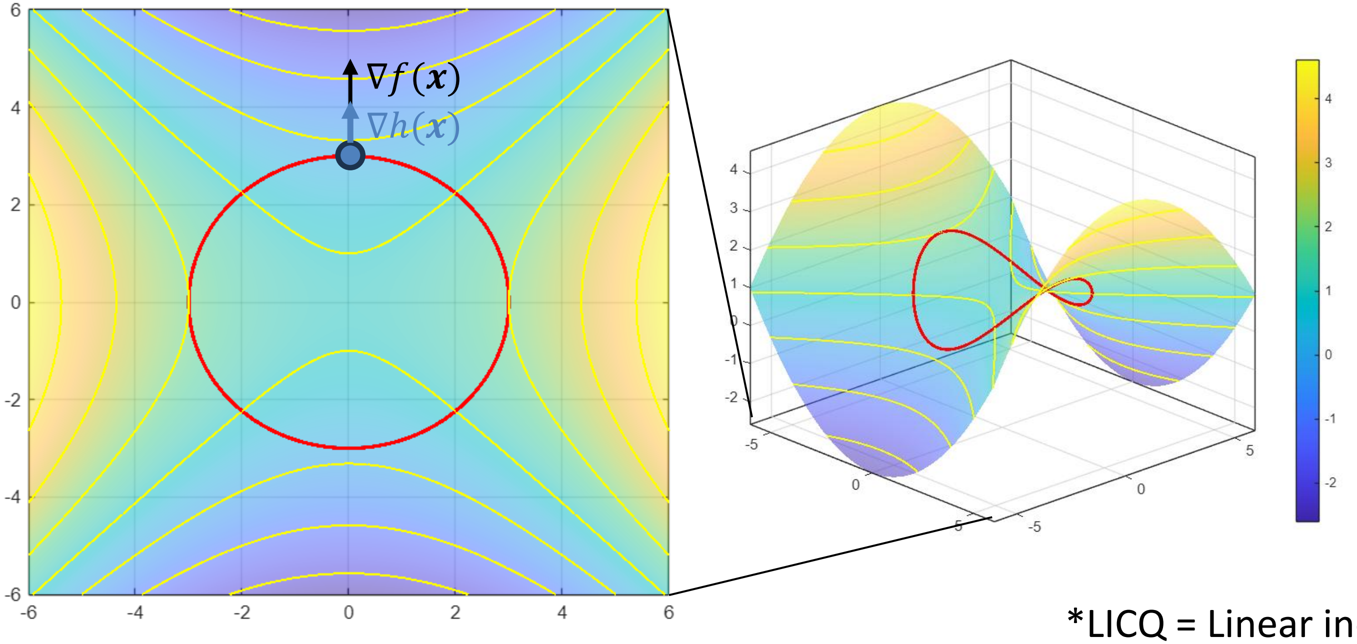
Theory of Lagrange Multipliers

Minimize: $f(\boldsymbol{x})$

Subject to: $h_i(\boldsymbol{x}) = 0 \qquad i \in \mathcal{E}$

$g_j(\boldsymbol{x}) \leq 0 \qquad j \in \mathcal{I}$

and $x_k^L \leq x_k \leq x_k^U \qquad k = 1, 2, \ldots, n$



If $\boldsymbol{x}^*$ is a local minimizer and the *gradients of the active constraints* satisfy linear independence (LICQ), then there exists a set of scalars (Lagrange multipliers), λ_i ($i \in \mathcal{E}$) and σ_j ($j \in \mathcal{I}$) such that:

Karush-Kuhn-Tucker (KKT) Conditions

$$\nabla f(\boldsymbol{x}^*) + \sum_{i \in \mathcal{E}} \lambda_i \nabla h_i(\boldsymbol{x}^*) + \sum_{j \in \mathcal{I}} \sigma_j \nabla g_j(\boldsymbol{x}^*) = 0$$

or $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\sigma}^*) = 0$ *(stationarity)*

$$h_i(\boldsymbol{x}^*) = 0$$
$$g_j(\boldsymbol{x}^*) \leq 0$$
(primal feasibility)

$$\lambda_i \geq 0, \qquad i \in \mathcal{E}$$
$$\sigma_j \geq 0, \qquad j \in \mathcal{I}$$
(dual feasibility)

$$\sigma_j g_j(\boldsymbol{x}^*) = 0, \qquad j \in \mathcal{I}$$
(complementarity)

*LICQ = Linear independence constraint qualification

Multivariable, Constrained NLP

- Newton's method
- Quasi-newton Method (BFGS)
- Levenberg-Marquardt
- Trust Region Newton Methods
- **Active Set and Interior-Point Methods**
- Sequential Quadratic Programming (SQP)
- IPOPT

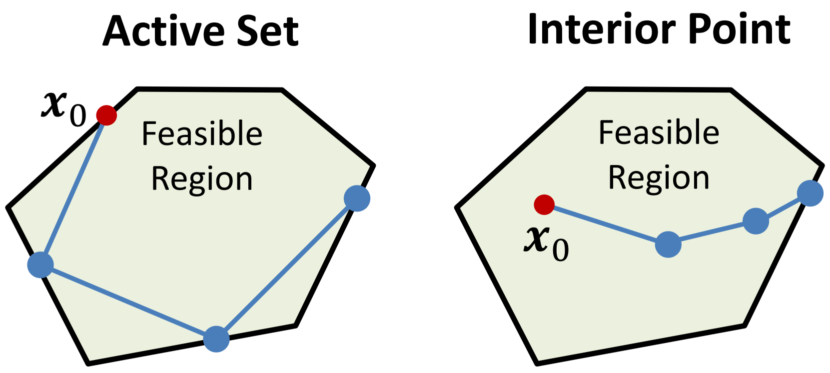
Active Set and Interior Point Methods

Minimize: $f(\boldsymbol{x})$

Subject to: $h_i(\boldsymbol{x}) = 0 \quad i = 1, 2, \dots, l$

$g_j(\boldsymbol{x}) \le 0 \quad j = 1, 2, \dots, m$

and $x_k^L \le x_k \le x_k^U \quad k = 1, 2, \dots, n$



Active Set Methods

- Popular since the 1970s
- **Idea:** Guess an initial active set of constraints, then add / delete using gradient and Lagrange multiplier information until optimality is detected.
- **Suitable for:** small number of constraints
- **Usage:** MATLAB SQP, Python scipy SLSQP, etc.

Interior Point Methods

- Popular since the 1990s
- **Idea:** Guess an initial $\boldsymbol{x}$ inside the feasible region, then stay strictly inside as you iterate until a boundary is approached.
- **Suitable for:** Large-scale, sparse problems
- **Usage:** IPOPT, MOSEK, other modern solvers

https://doi.org/10.1016/j.ejco.2025.100105

Multivariable, Constrained NLP

- Newton's method
- Quasi-newton Method (BFGS)
- Levenberg-Marquardt
- Trust Region Newton Methods
- Active Set and Interior-Point Methods
- **Sequential Quadratic Programming (SQP)**
- IPOPT

Sequential Quadratic Programming (SQP)

Minimize: $f(\boldsymbol{x})$

Subject to: $h_i(\boldsymbol{x}) = 0 \quad i = 1, 2, \ldots, l$

$g_j(\boldsymbol{x}) \leq 0 \quad j = 1, 2, \ldots, m$

and $x_k^L \leq x_k \leq x_k^U \quad k = 1, 2, \ldots, n$

- In Python's `scipy.optimize`, the **SLSQP** algorithm (Sequential Least-squares QP) uses line search (BFGS) + active-set solver.
- In MATLAB's `fmincon`, available algorithms include `sqp`, `active-set`, and `interior-point`.

SQP Subproblem

Quadratic

Linear

$$\min_{\boldsymbol{d}_x, \boldsymbol{d}_\lambda, \boldsymbol{d}_\sigma} f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_x + \frac{1}{2} \boldsymbol{d}_x^T \nabla_{xx}^2 \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\sigma}_k) \boldsymbol{d}_x$$

Subject to:

$$\boldsymbol{h}(\boldsymbol{x}_k) + \nabla \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{d}_\lambda = 0$$

$$\boldsymbol{g}(\boldsymbol{x}_k) + \nabla \boldsymbol{g}(\boldsymbol{x}_k)^T \boldsymbol{d}_\sigma \leq 0$$

Step 1: Make an initial guess, $\boldsymbol{x}_0, \boldsymbol{\lambda}_0, \boldsymbol{\sigma}_0$.
Step 2: Solve the SQP subproblem for $\boldsymbol{d}_{x,\lambda,\sigma}$.
Step 3: Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}_x$,

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \boldsymbol{d}_\lambda,$$

$$\boldsymbol{\sigma}_{k+1} = \boldsymbol{\sigma}_k + \boldsymbol{d}_\sigma.$$

where: $\mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\sigma}_k) = f(\boldsymbol{x}_k) + \boldsymbol{\lambda}_k^T \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{\sigma}_k^T \boldsymbol{g}(\boldsymbol{x}_k)$
is the corresponding Lagrangian of the original NLP.

$\boldsymbol{\lambda}_k, \boldsymbol{\sigma}_k$ are Lagrange multipliers.

- Solve using Active Set or Interior-Point.
- **Line-search SQP:** Use quasi-newton on $\boldsymbol{d}$.
- **Trust-region SQP:** Use trust-region on $\boldsymbol{d}$.

IPOPT (Interior Point OPTimizer)

https://github.com/coin-or/Ipopt

- Designed to find local solutions of optimization problems of the form:

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

$$\text{s.t.} \quad \boldsymbol{g}_L \leq \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{g}_U$$
$$\boldsymbol{x}_L \leq \boldsymbol{x} \leq \boldsymbol{x}_u$$

- The functions $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ can be nonlinear, nonconvex, but must be twice differentiable.

- Main steps:
  - Initialization
  - Check convergence
  - Compute search direction: **Damped Newton**
  - Solve for step size: **Filter Line Search**
  - Update

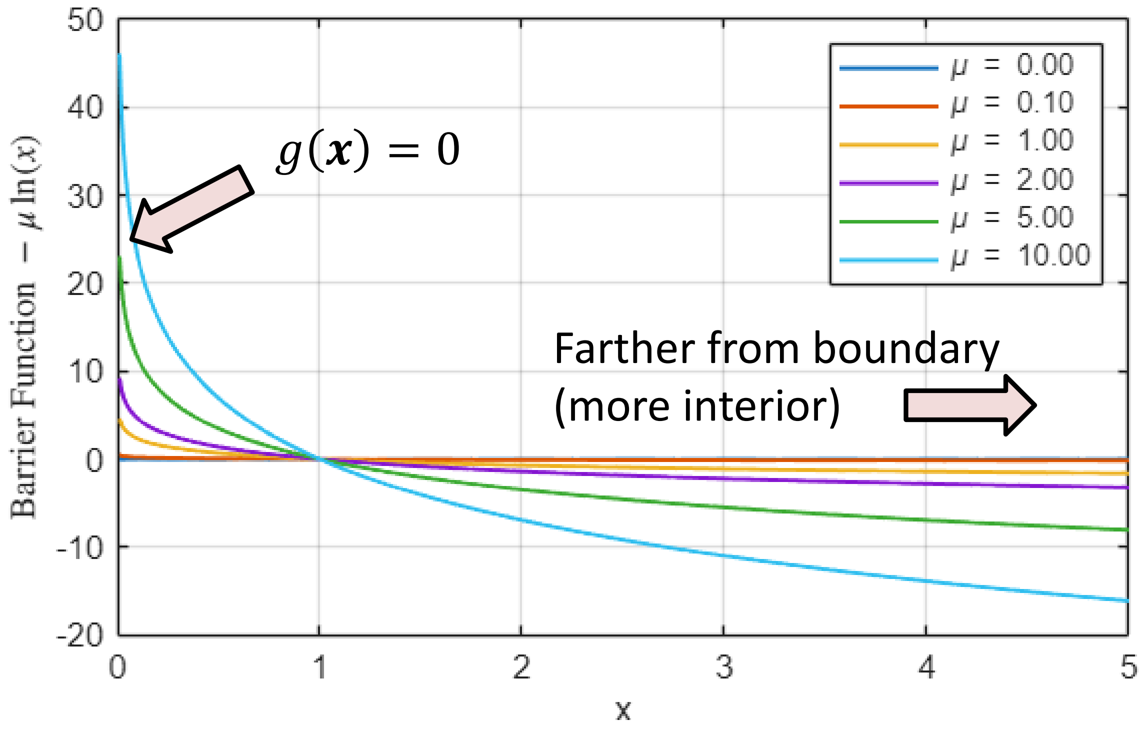
- A. Wächter and L. T. Biegler, On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming, *Mathematical Programming* 106(1), pp. 25-57, 2006.

Barrier Method

Transform the NLP by adding a log-penalty, then decrease the value of *barrier parameter*, μ, to approach the original NLP.

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \quad \Longrightarrow \quad \min_{\mathbf{x}\in\mathbb{R}^n} f(\boldsymbol{x}) - \mu \sum_i \ln(g(\boldsymbol{x}_i))$$

$$\text{s.t.} \quad g(\boldsymbol{x}) \geq 0$$

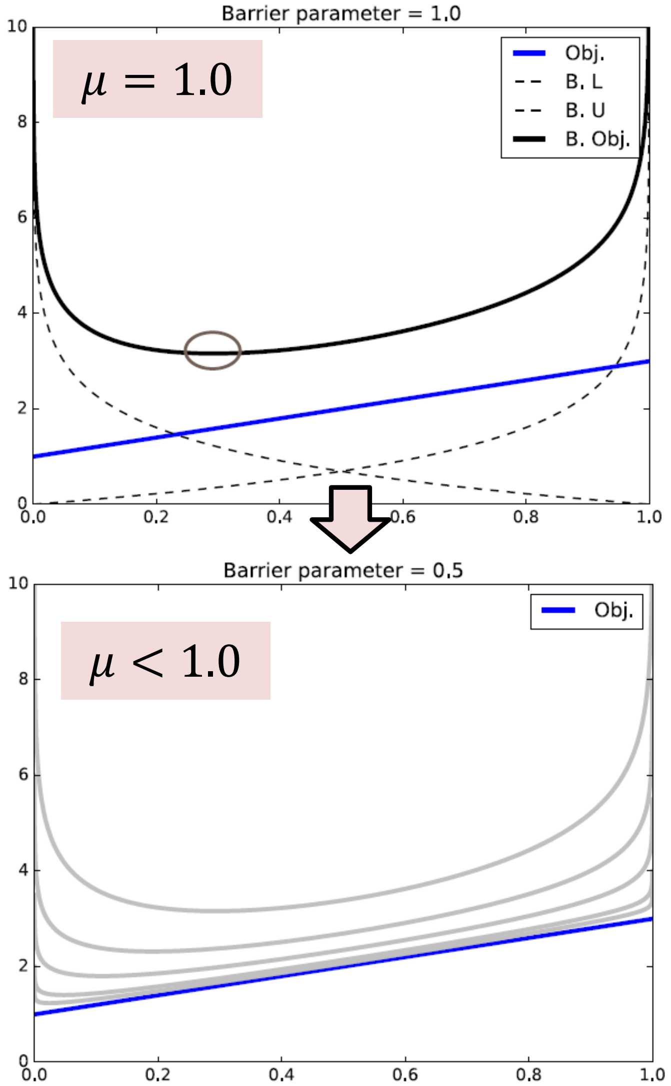


Source: https://www.youtube.com/watch?v=bJ0Kkf4u9bo

IPOPT (Interior Point OPTimizer)

https://github.com/coin-or/Ipopt

- Effect of the barrier term:

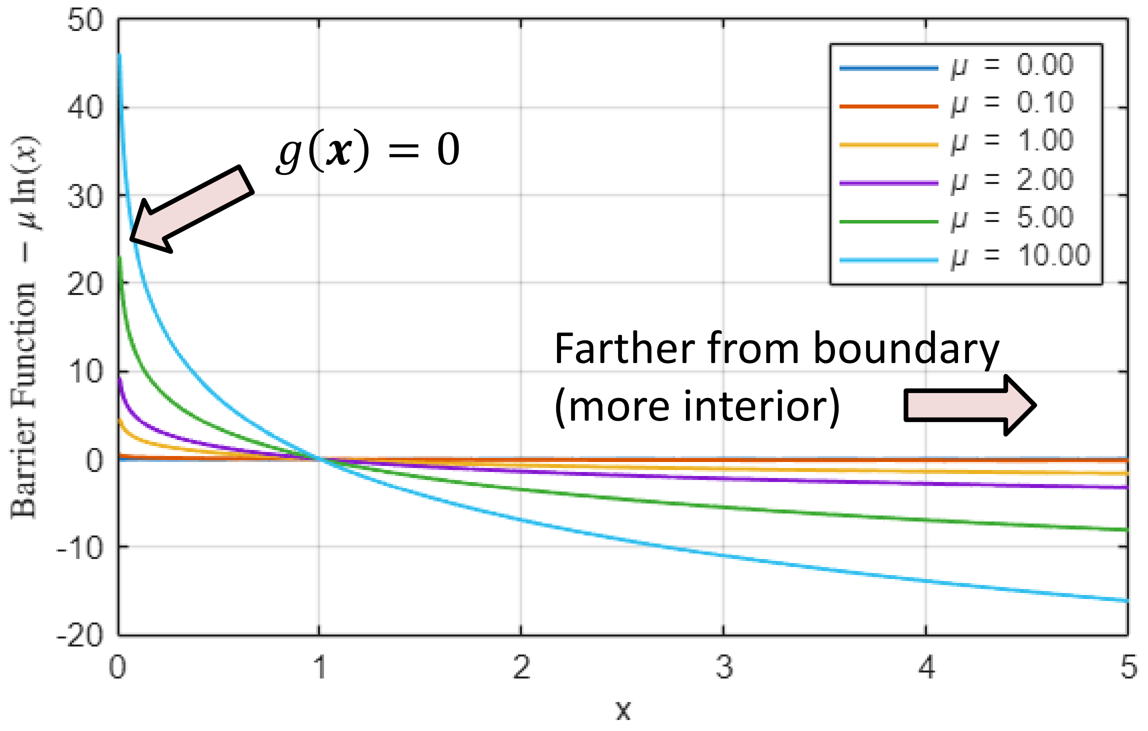


Barrier Method

Transform the NLP by adding a log-penalty, then decrease the value of *barrier parameter*, μ, to approach the original NLP.

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\boldsymbol{x}) \quad \Longrightarrow \quad \min_{\mathbf{x}\in\mathbb{R}^n} f(\boldsymbol{x}) - \mu \sum_i \ln(g(\boldsymbol{x}_i))$$

$$\text{s.t.} \quad g(\boldsymbol{x}) \geq 0$$



Source: https://www.youtube.com/watch?v=bJ0Kkf4u9bo

Common Pitfalls of NLP Solvers

1. Termination at nonoptimal point.
   A. Typically, due to poor derivative accuracy.
   B. No good search directions found.
   C. Successive objective function values converge, but KKT conditions are not satisfied.

2. Termination at non feasible point.
   A. Problem may have conflicting constraints
   B. Initial guess is poorly located
   C. Functions may have discontinuities / undefined regions.

3. Solver takes a long time
   A. Possibly no solution exists
   B. Solver may not be appropriate
   C. Tolerances are too small

Some workarounds:

1. Avoid using functions with undefined regions such as sqrt(-x), log(x), etc. Use x^2, exp(x), etc. instead.

2. Make sure initial guess is well-informed by physical intuition and it is feasible.

3. Try lowering tolerance or changing the initial guess if solver is stuck.

4. Scaling the variables to similar magnitudes improves numerical stability.

5. Try solving a smaller problem first, then add constraints or variables incrementally.

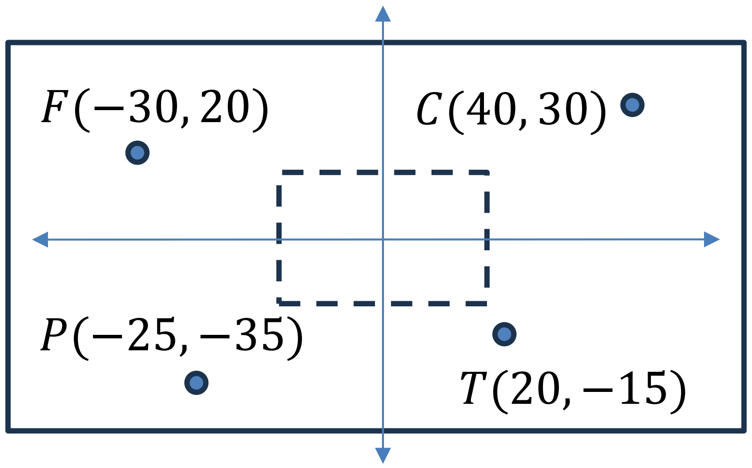
Class Exercises

Problem 1 Minimize the Himmelblau function within a circle:

$$\min (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

$$\text{s.t. } x_1^2 + x_2^2 \leq 9$$

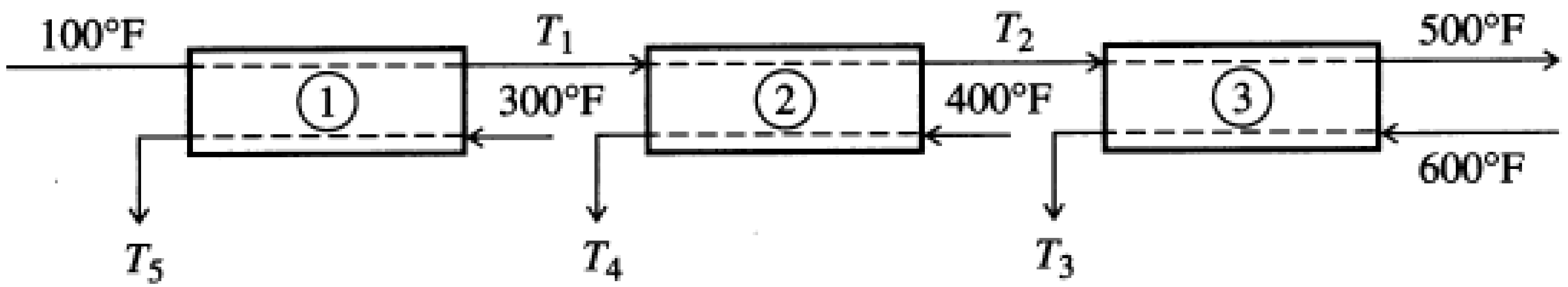
Problem 2 **[Rao (2009). Engineering Optimization: Theory and Practice]**
The layout of a processing plant consisting of a pump (P), a water tank (T), a compressor (C), and a fan (F), is shown. The locations of the various units in terms of (x, y) coordinates are also indicated. It was decided to add a new unit, a heat exchanger (H), to the plant. To avoid congestion, it was decided to locate H within a rectangular area defined by $-15 \leq x \leq 15$ and $-10 \leq y \leq 10$. Find the location of H that minimizes the sum of its Euclidean distances from the existing units, P, T, C, and F.



Class Exercises

Problem 3 **[EHL Problem 8.41]**

For the purposes of planning, you are asked to determine the optimal heat exchanger areas for the sequence of 3 exchangers as shown.



Exchanger	U	Area	Duty
1	120	A_1	Q_1
2	80	A_2	Q_2
3	40	A_3	Q_3

Given that $mC_p = 10^5 \frac{Btu}{hr°F}$, find the temperatures T_1, T_2, T_3, such that ΣA_i is a minimum. Assume that the following design equation is valid for ends 1 and 2 in each heat exchanger:

$$Q = UA\frac{\Delta T_1 + \Delta T_2}{2}$$

A Taxonomy of NLP Solvers

Zero-order Methods

Derivative-free, black-box

- Grid Search / Exhaustive Search
- Random Search
- Nelder-Mead Simplex

- Metaheuristic Search
  - Genetic Algorithms
  - Particle Swarm
  - Simulated Annealing
  - Differential Evolution
  - CMAES

- Bayesian Optimization / Surrogate-based

First-order Methods

Uses 1st derivative information

- Steepest Descent or Gradient Descent
- Stochastic Gradient Descent (SGD)
  - RMSProp, Adam, etc.
- Conjugate Gradient methods (e.g. Fletcher-Reeves)
- Coordinate Descent

Second-order Methods

Uses 1st and 2nd derivative information

- *Newton's method*
- *Quasi-newton method (BFGS)*

- Levenberg-Marquardt
- Trust Region Newton Methods
- Active Set and Interior-Point Methods
- Sequential Quadratic Programming (SQP)
- IPOPT