Data-Drive Process Systems Engineering: Notes

Update date: February 16, 2022

Contents

1 Big picture

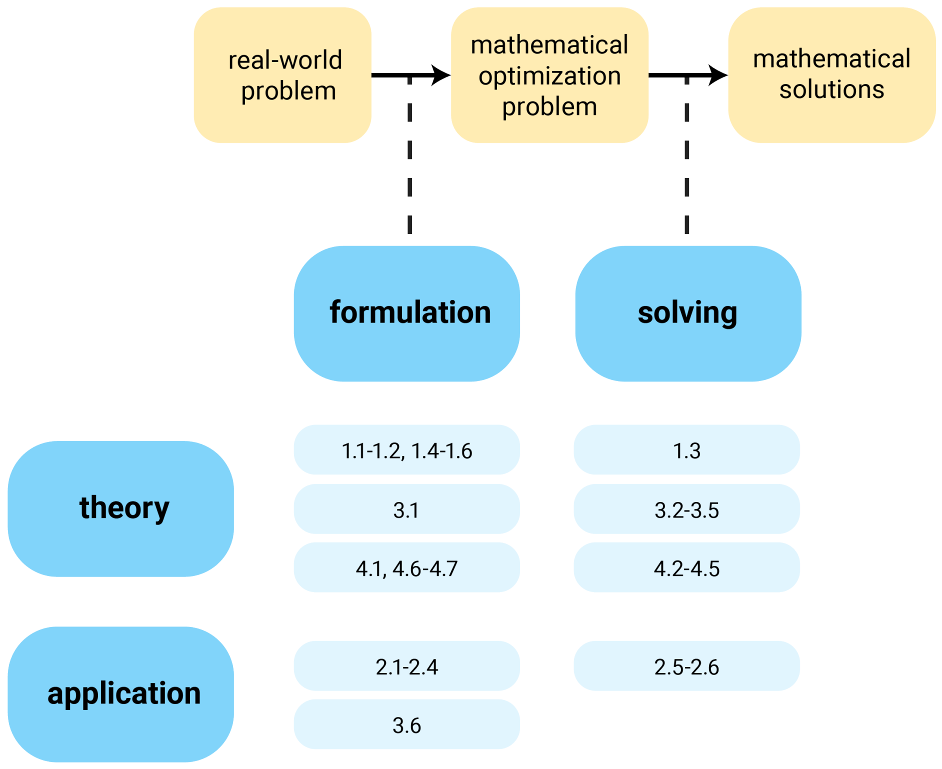


Figure 1: Big picture

Recall that the focus of this course is two steps in optimization: (1) formulating an optimization problem from real-world problems; (2) solving the optimization problems. Meanwhile, we will learn these steps from two different aspects: in application, such as writing an optimization problem in pyomo, and calling a solver to solve it; also in theory, such as learning the mechanism of the simplex method behind the LP solvers. So I made this figure to roughly classify all module lessons in these 4 categories, so that it is easier to think about a specific lesson in terms of the big picture and make connections among different lessons.

2 Week 1: Introduction to optimization

Learning objectives:

- How to *formulate* problems using mathematical functions
- How to *solve* the problems

General steps:

1. identify variables, objective, constraints
2. identify the form of the problem

3. choose appropriate solver based on optimum requirement (local/global) and problem form

Programmer vs optimizer dealing with optimization problems:

- Programmer: enumeration-"loop" through all options; impractical and unable to find true optimum
- Optimizer: formulation + numerical optimization-transform all logic into mathematical equations, then apply mathematical tools (solvers) to target optimum systematically and quickly

2.1 Formulation overview

Notation: optimization formulation

$$\begin{aligned} \text{max/min}_{\text{variables}}\ &\text{objective function} \qquad && \min_{x,y}\ x+y \\ \text{subject to}\ &\text{constraints} && \text{s.t. } x-y \geq 20 \\ &\text{bounds} && \quad x \geq 0, y \geq 0 \end{aligned}$$

Elements:

- Variables: mathematical form of decisions to be made
- Objective function: 1-dimensional function of variables, mathematical form of goal of model, to be maximized or minimized
- Constraints: equality of inequality with respect to (w.r.t.) function of variables, mathematical form of physical limits/specifications/demands to be met; can be equality (=) or inequality ($\leq, \geq$)[1]
- Parameters: constant values used in objective and constraints
- Bounds: variable lower and upper bounds

Notation: variable

- unindexed: x (italic font in this note)
- indexed: $x_i, i \in N$, N as some set
- vector form: $\mathbf{x}$ (bold, roman font in this note)

Notation: inequality constraint

general form:

$$g(\mathbf{x}) \geq a, h(\mathbf{x}) \leq b, \ldots$$

standard from:

$$\bar{g}(\mathbf{x}) \leq 0, \text{ can be obtained by modifying constraints, e.g. } \bar{g}(\mathbf{x}) \equiv a - g(\mathbf{x})$$

indexed form:

$$g_i(\mathbf{x}) \leq 0, i \in N$$

[1]strict inequality is not encouraged as it can affect the existence of the optimal solution (in theory) and may cause numerical issues (in practice).

Notation: equality constraint

similar to inequality constraints, with standard from $h(\mathbf{x}) = 0$

2.2 Variable types

Continuous variables:

- can take any values between lower and upper bounds
- can represent temperatures, concentration, etc.

Integer variables:

- can only take integer values, $1, 2, 3, \ldots$
- can represent number of parallel processes, number of stages
- typically transformed to binary variables[2]

Binary variables:

- can be either 0 or 1
- can represent "yes" or "no"
- can be used to apply the "logic" within the optimization model

Notation: variables

Continuous variable: $\mathbf{x}^{\mathrm{L}} \leq \mathbf{x} \leq \mathbf{x}^{\mathrm{U}}$; $x \in \mathbb{R}$ (x is a real number); $\mathbf{x} \in \mathbb{R}^n$ (the vector $\mathbf{x}$ is a n-dimensional real number)
Binary variable: $\mathbf{y} \in \{0, 1\}^m$ ($\mathbf{y}$ is a m-dimensional binary variable)

Transform of integer variables to binary variable:

- Assume integer variable $y \in a_1, a_2, \ldots, a_n$
- it can be replaced with n binary variables and one continuous variable: let $x_1, x_2, \ldots, x_n$ be binary variables representing which value y takes, then the following x can get the same integer values of y

$$a_1x_1 + a_2x_2 + \ldots a_nx_n = x$$
$$a_1 + a_2 + \ldots a_n = 1.$$

2.3 Formulation types

Linear optimization problem:

- both objective and constraints are linear w.r.t. all variables
- generally follow this form: $c_1x_1 + c_2x_2 + \ldots$, c_i's being parameters

Nonlinear optimization problem:

[2]Reason for doing this is that MILP solvers utilize algorithms on binary variables (which will be discussed in later modules), instead of integer variables

- There exists at least one nonlinear function among objective and constraints
- e.g. $\sqrt{x}, x^2, x \cdot y$ with both x, y being variables, etc.

Table for formulation types[3]: Table 1

Table 1: Formulation types

	all linear equations	include nonlinear equations
all continuous variables	LP	NLP
all integer variables	ILP	INLP
include both variables	MILP	MINLP

Feasible points: solution that satisfies all constraints

Feasible region: set of all feasible points

- When all variables are continuous, feasible region is a "region"/"space"
- When all variables are binary, feasible region is a collection of points

Active constraint and active set:

- At a given feasible solution $\mathbf{x}^*$, the constraint $g_i(\mathbf{x}) \le 0$ is called active if $g_i(\mathbf{x}^*) = 0$
- the set of active constraint at that point is called the active set[4]

2.4 Optimality

Global minimizer[5]:

- solution $\mathbf{x}^*$ s.t. $f(\mathbf{x}^*) \le f(\mathbf{x}) \forall \mathbf{x} \in S$, with S being the feasible region
- "true" optimum

Local minimizer:

- solution $\mathbf{x}^\dagger$ s.t. $f(\mathbf{x}^\dagger) \le f(\mathbf{x}) \forall \mathbf{x} \in \left\{x : \left\|x - x^\dagger\right\| \le \varepsilon\right\}$
- the "best" solution in its neighbour
- local minimizers are always stationary points, but stationary points are not always local minimizers (they can also be saddle points)
- local minimum can have a huge difference with global minimum

[3]ILP and INLP are less common, especially INLP

[4]this will come in handy in later modules

[5]Typically the "minimizer" refers to the variables that corresponds to the minimal objective value, while minimum refers to the "objective value"

2.5 Convexity

Convexity (definition in the slides): having an outline or surface curved like the exterior of a circle or sphere.[6]

- Important property: if all constraints and the objective are all convex, then all local optimums are also global optimum. This makes the optimization model much easier to solve.

Elements in an optimization model that cause nonconvexity:

- bilinaer term: $x \cdot y$, common in chemical industry (e.g. flowrates multiplying concentrations)
- binary variables: "yes" or "no" choices

Relationship between nonconvexity and problem types:

- LP problems are convex, can be solved globally
- MILP problems can be solved globally, though they are not convex
- NLP problems may or may not be convex; if they are convex, they can be solved globally; if they are nonconvex, global optimum will be prohibitively expensive to obtain; local optimums are cheaper to get, but they are just locally optimal
- MINLP problems are hardest to solve

Optimization vs. heuristic grid search:

- grid search: may miss the real optimums between the sampled points; impractical and inefficient
- optimization: rely on systematically trying different values to improve the solution, can give you different levels guarantees of optimum

2.6 Solver types

Exact deterministic solvers:

- will solve the same problem with the same answer every time
- can theoretically guarantee the optimality of the solution
- can be local or global

Stochastic solvers:

- have "randomness" in the search mechanism
- may give different solutions for the same problem with different runs

3 Week 2: Optimization in Python

Python vs. MATLAB:

- Python: Flexibility, free and open source, supportive community
- MATLAB: Version control[7] and compatibility, specialized toolboxes

[6] I would recommend some further reading in the mathematical definition of the convex set and the convexity function. It may help build a clearer vision of what convexity is.

[7] It is integrated within the MatLab GUI, but you can definitely do the same for Python codes manually

Useful Python packages:

- `numpy` - matrix operations
- `scipy` - scientific algorithm
- `matplotlib` - plotting
- `pyomo` - optimization
- `scikit-learn` - machine learning

Useful external tools for writing Python codes:

- `Spyder` : standalone IDE for `.py` files
- `Jupyter Notebook` : the main interactive tool for this course
- `Anaconda` : package and environment management

3.1 Installation instructions

The following instruction is tested on my personal MacBook with MacOS Big Sur.

- Use `anaconda` (download) to manage packages and versions. A quick Anaconda tutorial is available online.
- After installing `anaconda`, install packages by running the following commands in terminal[8][9][10]:

```
# install common packages
conda install matplotlib numpy scipy scikit-learn
# install jupyter notebook
conda install jupyter
# install pyomo
conda install -c conda-forge pyomo
# install solvers: GLPK, BCB, IPOPT
conda install -c conda-forge coincbc ipopt glpk
```

3.2 Python code example

```python
import numpy as np # import package, and set "nickname" for it
import scipy as sp
import matplotlib as mlp

A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # 3 by 3 matrix
b = np.array([1, 2, 3]) # 3 vector
x = np.linalg.solve(A, b) # solve Ax = b
print(x) # print results
```

[8]Optionally, you can set up a new environment specifically for this course, so that the installation of required packages will not affect existing packages/python versions. Tutorial on environment management is available online.

[9]Please make sure that these commands are executed in your shell instead of inside one of Jupyter notebook, which may lead to weird situations.

[10]For Windows users, run these commands in the Anaconda Prompt

Control flow:

```
# for loop
sum = 0
for n in [1, 2, 3, 4, 5]:
  sum = sum + n

# if statement
if x > 40:
  print(x)
else:
  print(y)
```

Function definition:

```
def abs(x): # x as input
  if x > 0:
    return x # output if x is positive
  else:
    return -x # output if x is negative
```

3.3 Introduction to Pyomo

`pyomo`: Python package for formulating and solving optimization problems, developed by Sandia national lab, developed for real-world problems.

Import `pyomo`:

```
# this syntax is the same as other packges
# calling Pyomo functions needs to add prefix, e.g. pe.ConcreteModel()
import pyomo.environ as pe

# notice the different syntax
# this one can directly call Pyomo functions, e.g. ConcreteModel()
from pyomo.environ import *
```

3.3.1 Building blocks of Pyomo models

Sets: used for indexing variables, constraints and variables

```
# create set with 4 elements
model.i = pe.Set(initialize=[1, 2, 3, 4])

# DON'T DO THIS: model.i = pe.Set([1, 2, 3, 4]), it will not work as you think
# AND DON'T DO THIS: model.i = [1, 2, 3, 4], usually considered poor style

# create set with elements starting from 1, ending at 10, with footstep 1
model.j = pe.RangeSet(1, 10, 1)
```

```

# fetch the length of the set
len(model.i)

# apply the logical operators on sets
model.i | model.j
```

Parameters: numerical or symbolic value used in constraints or objective; can be unindexed or indexed (by a Python dictionary); immutable by default

```
# indexed by set model.i, initialized by Python dict p
p = {1: 1, 2:2, 3:3, 4:4}
model.p = pe.Param(model.i, initialize=p)
```

Variables: can set their lower and upper bounds, domain, initial values, and types using `within` (`Reals` , `PositiveReals` , `Integers` , etc.); can be fixed and unfixed

```
# initialize variable z indexed by set i, requiring it to be nonnegative real
# numbers
model.z = pe.Var(model.i, initialize={1:1.5, 2:2.5, 3:3.5, 4:4.5},
within=pe.NonNegativeReals)

# access z[1] upper bound
model.z[1].ub
# dupate z[1] upper bound
model.z[1].setub(20)

# fix z[2]
model.z[2].fix()
```

Constraints:

- supported constraint types: equality, nonstrict inequality, range constraints (lower bound $\leq$ `expression` $\leq$ upper bound)
- strict inequality can be achieved via changing the value of the right hand side of the constraint[11]
- can be declared using `expr` in an inline manner:

  ```
  model.con = pe.Constraint(rule = model.z[2] - model.z[1] <= 2)
  ```

- can be declared using `rule` with flexible Python function:

  ```
  # define Python function for constraint rule
  def diff_rule(m, i):
    # conditional skip
    if i == 4:
  ```

[11]This is not encouraged as it can affect the existence of the optimal solution (in theory) and may cause numerical issues (in practice).

```
    return pe.Constraint.Skip
  else:
    return m.z[i] <= m.z[i + 1]

# apply the rule; notice "model" correspond to "m" in the argument, and
# "model.i" correspond to the index set for i
model.con_2 = pe.Constraint(model.i, rule=diff_rule)
```

Objective: similar to constraint, can be set with `expr` or `rule`; always an equality[12]; can be set as minimized or maximized with `sense`

```
# the following 3 ways of declaring objectives are the same
model.obj = pe.Objective(expr=pe.summation(model.p, model.z), sense=maximize)

def obj_1(m):
  return pe.summation(m.p, m.z)
model.obj_1 = pe.Objective(rule=obj_1, sense=maximize))

def _obj_2(m):
  obj_value = pe.summation(m.p, m.z)
  return obj_value
model.obj_2 = pe.Objective(rule=obj_2, sense=maximize))
```

3.3.2 Solving Pyomo models

Solving command (in command line mode):

```
# IPOPT can be replaced with solver name, test_file.py can be repalced with
# .py file to be solved
pyomo solve --solver=IPOPT test_file.py
# to show solving progress, add --stream-solver within the line
# to show model summary, add --summary
# to show final results, add --show-results
```

Solving command (in `Jupyter notebook`, or solving the model within the .py script):

```
# set solver, by putting solver name in string and sending it to function
# pe.SolverFactory()
solver = pe.SolverFactory('glpk')
# solve the model by calling the method solve of the solver we just created
# using the model name as the argument
solver.solve(model)
# print result
model.pprint()
```

[12]A more precise description is: the objective is an expression without equality or inequality signs; it should be a 1-dimensional function of variables

3.3.3 Pyomo solvers

- Baron - Global MINLP solver, commercial, available to GT students
- CPLEX, GUROBI - LP/MIP solver, commercial
- CBC - LP/MIP, open source
- GLPK - LP/MIP, open source
- IPOPT - NLP solver, open source

Solver related parameters:

- Solver status: tells how the solver terminates
- Termination conditions: tells why solver terminates (successfully or unsuccessfully), and which type of optimum is achieved

4 Week 3: Linear optimization

4.1 Linear problems

- definition: all constraints are linear inequalities
- easiest to solve, so we should try to formulate the problem as linear problems whenever possible
- subsets: LP-all variables are continuous, ILP-all variables are discrete, MILP-contains both types of variables

Examples of LP:

- planning and scheduling
- network flow
- multicommodity flow
- airlines, transportation, electric grids, etc.

Examples of MILP: LP problems, and

- protein design and structure prediction
- supply chain optimization
- knapsack
- identifying relevant symptoms in patients
- panel/committee assignment
- machine learning models, etc.

LP & MILP solvers:

- State-of-the-art commercial solvers: CPLEX, XPRESS
- solver list available on COIN-OR and CUTEr

4.2 Simplex method

Core algorithm for solving LP.

Basic principle:

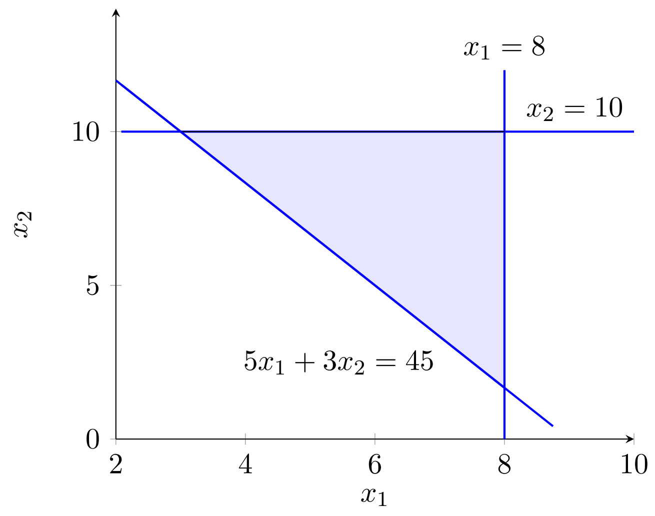
- If an LP has an optimal solution, then at least one of the corner points is optimal
- we only need to check a finite number of corner points to find an optimal solution

4.2.1 Graphical solutions of LP

Problem:

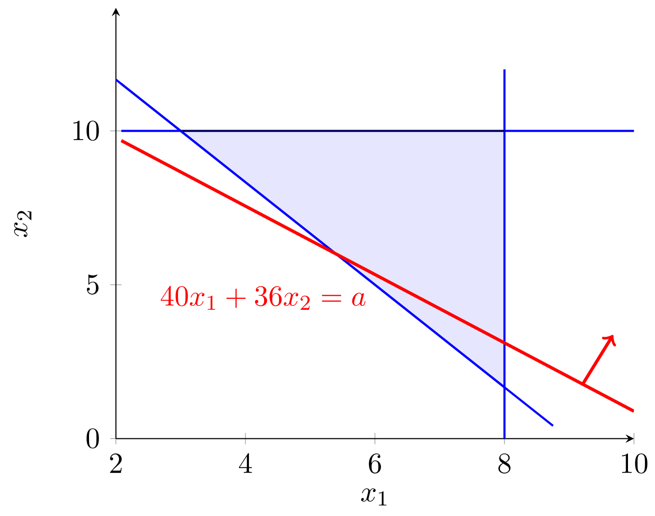
$$\begin{aligned} \max\ & 40x_1 + 36x_2 \\ \text{s.t. } & 5x_1 + 3x_2 \geq 45 \\ & x_1 \in [0, 8], x_2 \in [0, 10] \end{aligned}$$

Step 1: draw feasible region



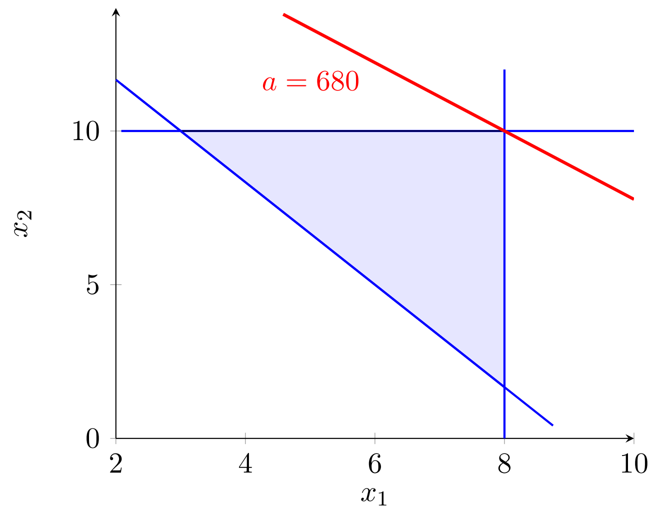
Step 2: draw objective function

- Let $40x_1 + 36x_2 \equiv a$, calculate the slope $-40/36$



Step 3: move a until the objective function just touches the feasible region with one corner point

- Optimal solution: $x_1 = 8, x_2 = 10, a = 680$



4.2.2 Simplex standard form

Form of problem for the simplex method to start:

1. maximization
2. all equality constraints
3. all variables ≥ 0

Minimization problem:

- use the maximization of the opposite of the objective function
- E.g. $\min x_1 + x_2 \Longrightarrow \max -x_1 - x_2$

Inequality constraints:

- add nonnegative variable to the LHS of $\leq$ sign, called slack variables
- E.g. $5x_1 + x_2 \leq 10 \Longrightarrow 5x_1 + x_2 + s = 10$

Negative variable x:

- define new variable x' with $x' = x - x^{\mathrm{L}}$
- E.g. $x + y \leq 10, x \in [-10, 10] \Longrightarrow (x' - 10) + y \leq 10, x \in [0, 20]$

4.2.3 Simplex method basics

- Let n denote the number of variables, m denote the number of equations
- The constraints of a simplex standard form is a $m \times n$ system of equations (plus inequalities for bounds)
- For an optimization model, $n > m$ so that there are degrees of freedom to optimize

Elementary row operations (linear algebra):

- multiply a row by a nonzero constant
- add constant multiple of a row to another

Gauss-Jordan elimination:

- helps find feasible solutions
- transform simplex standard form to canonical form

Canonical (row-echelon) form:

- the system where some variables only participate in 1 equation, with a coefficient of 1
- they do not exist in the other equations

$$
\begin{aligned}
&a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1\\
&a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2\\
&\cdots\\
&a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n = b_m\\
&x_1, x_2, \ldots, x_n \geq 0
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
x_1 + \quad & \bar{a}_{1,m+1}x_{m+1} + \bar{a}_{1,m+2}x_{m+2} + \cdots + \bar{a}_{1,n}x_n = \bar{b}_1\\
x_2 + \quad & \bar{a}_{2,m+1}x_{m+1} + \bar{a}_{2,m+2}x_{m+2} + \cdots + \bar{a}_{2,n}x_n = \bar{b}_2\\
& \cdots\\
x_m + & \bar{a}_{m,m+1}x_{m+1} + \bar{a}_{m,m+2}x_{m+2} + \cdots + \bar{a}_{m,n}x_n = \bar{b}_m\\
& x_1, x_2, \ldots, x_n \geq 0
\end{aligned}
$$

- basic/dependent variables: $x_1, x_2, \ldots, x_m$
- nonbasic/independent variables: $x_{m+1}, x_{m+2}, \ldots, x_n$
- advantage of canonical form: if the values of nonbasic variables are fixed, the rest system is $m \times m$ for basic variables and can be solved

Pivot operation:

- Set of elementary row operations that reduce the coefficient of a variable to 1 in one equation and eliminate it from all others

Basic solution:

- the solution where all nonbasic variables are zero
- $x_i = \bar{b}_i, i = 1, \ldots, m$
- number of basic solutions: $\binom{n}{m}$

Basic feasible solution (BFS):

- basic solution with all non-negative values
- BFS is identical to the corner point/vertex of the feasible region
  - This indicates that we only need to visit finite BFS to arrive at an optimal solution
  - with objective function, we do not need to visit all BFS
- objective value: $z = \sum_{i=1}^{m} c_i \bar{b}_i$

- adjacent BFS: a BFS that differs from another BFS with one 1 variable

4.3 Simplex method steps

1. Find initial BFS
2. Improve by finding another BFS with better objective value (as much as possible)
3. Eliminate BFS with worse objective
4. Terminate when there is no better BFS

Relative profit:

- The unit objective improvement of moving from a BFS to an adjacent BFS
- Assume the nonbasic variable x_s is now basic variable in the new BFS; other nonbasic variables remain zero
- The new model is

$$\max \sum_{i=1}^{m} c_i x_i + c_s x_s$$

$$\text{s.t.} \quad \begin{aligned} x_1 + \quad\quad\quad \bar{a}_{1,s} x_s &= \bar{b}_1 \\ x_2 + \quad\quad \bar{a}_{2,s} x_s &= \bar{b}_2 \\ \dots & \\ x_n + \bar{a}_{2,s} x_s &= \bar{b}_n \end{aligned}$$

- when x_s changed a unit, $x_i = \bar{b}_i - \bar{a}_{i,s}, i = 1, \dots, m$; new objective

$$z^{\dagger} = \sum_{i=1}^{m} c_i \left(\bar{b}_i - \bar{a}_{i,s} \right) + c_s, \quad \Delta z = z^{\dagger} - z = c_s - \sum_{i=1}^{m} c_i \bar{a}_{i,s}$$

5 Week 4: Nonlinear optimization

5.1 NLP formulation

Characteristics of NLP:

1. all variables are continuous
2. has at least one nonlinear term in constraints/objectives

Applications in engineering:

- reactor design/separations design
- optimization with embedded machine learning models
- flowsheet optimization with recycle streams
- parameter estimation of nonlinear models
- portoflio selection
- constrained regression

Challenges:

- convex NLP-easier to solve, all local optimums are also global optimums
- nonconvex NLP-local optimums are not necessarily global optimums; easy to solve locally, expensive to solve globally
- nonlinear feasible region, any point within could be optimum (contrast to LP, must be at vertices)

Popular solvers:

- local-IPOPT
- global-BARON

Example 1 (serial reaction in CSTR)**.** Consider a serial reaction: A $\xrightarrow{k_1}$ B $\xrightarrow{k_2}$ C, with reaction rates:

$$\frac{\mathrm{d}c_\mathrm{A}}{\mathrm{d}t} = -k_1 c_\mathrm{A}$$
$$\frac{\mathrm{d}c_\mathrm{B}}{\mathrm{d}t} = k_1 c_\mathrm{A} - k_2 c_\mathrm{B}$$
$$\frac{\mathrm{d}c_\mathrm{C}}{\mathrm{d}t} = k_2 c_\mathrm{B}.$$

Parameters $k_1 = 0.5, k_2 = 0.1, c_{\mathrm{A},0} = 2, c_{\mathrm{B},0} = c_{\mathrm{C},0} = 0.$[13] This is an ODE system.

1. Simulation on concentration profiles: We can simulate the concentration profiles by formulating it as an optimization model using the collocation method. This is not an optimization as there is no degrees of freedom. The following code uses an extension of `pyomo` called `pyomo.dae`, [14] which is capable of handling dynamic models. You don't have to understand the DAE-related details in the following codes.

```python
from pyomo.environ import *
# pyomo.dae is an extension that is capable of handling dae models
from pyomo.dae import *

# parameters
k_A = 0.5
k_B = 0.1
c_A0 = 2.0

m = ConcreteModel()

# we can specify a continuous time period and let the package discretize it
# automatically
m.t = ContinuousSet(bounds=(0, 5))

# define variables
m.c_A = Var(m.t, domain=NonNegativeReals)
m.c_B = Var(m.t, domain=NonNegativeReals)
m.c_C = Var(m.t, domain=NonNegativeReals)

# define derivatives as variables
```

[13] Here all units are neglected-we should make sure that in the same equation, all the units are uniformed correctly; but in optimization models, all terms are treated as pure numbers.

[14] `dae` stands for Differential-algebraic system of equations.

```python
m.d_c_A = DerivativeVar(m.c_A)
m.d_c_B = DerivativeVar(m.c_B)
m.d_c_C = DerivativeVar(m.c_C)

# reaction rate equations written as constraints
def ode_A(m, t):
    if t > 0:
        return m.d_c_A[t] == (- k_A * m.c_A[t])
    else:
        return Constraint.Skip
m.ode_A = Constraint(m.t, rule=ode_A)
def ode_B(m, t):
    if t > 0:
        return m.d_c_B[t] == k_A * m.c_A[t] - k_B * m.c_B[t]
    else:
        return Constraint.Skip
m.ode_B = Constraint(m.t, rule=ode_B)
def ode_C(m, t):
    if t > 0:
        return m.d_c_C[t] == k_B * m.c_B[t]
    else:
        return Constraint.Skip
m.ode_C = Constraint(m.t, rule=ode_C)

# add initial conditions
m.ic = ConstraintList()
m.ic.add(m.c_A[0] == c_A0)
m.ic.add(m.c_B[0] == 0)
m.ic.add(m.c_C[0] == 0)

# transform dae model to algebraic equations
TransformationFactory('dae.collocation').apply_to(m)

# solve the model
SolverFactory('ipopt').solve(m)

# print results at the 5th min
print(f'Outlet c_A: {value(m.c_A[5]):.2f}')
print(f'Outlet c_B: {value(m.c_B[5]):.2f}')
print(f'Outlet c_C: {value(m.c_C[5]):.2f}')
```

```
Outlet c_A: 0.16
Outlet c_B: 1.31
Outlet c_C: 0.52
```

2. Maximizing profit, with residence time being variable:
    - introducing variable: residence time t_f
    - introducing objective: $P = 10 \cdot C_{\text{B,final}} + 3 \cdot C_{\text{C,final}}$

- non-dimensionalize the differential equations

$$\tau = t/t_f \implies \begin{aligned} \frac{\mathrm{d}c_\mathrm{A}}{\mathrm{d}\tau} &= -t_f k_1 c_\mathrm{A} \\ \frac{\mathrm{d}c_\mathrm{B}}{\mathrm{d}\tau} &= t_f \left(k_1 c_\mathrm{A} - k_2 c_\mathrm{B}\right) \\ \frac{\mathrm{d}c_\mathrm{C}}{\mathrm{d}\tau} &= t_f k_2 c_\mathrm{B}. \end{aligned}$$

- Nonlinearity is introduced by the term $t_f \cdot c_i$

```python
from pyomo.environ import *
from pyomo.dae import *

k_A = 0.5
k_B = 0.1
c_A0 = 2.0

m = ConcreteModel()

# non-dimensionalized time
m.tau = ContinuousSet(bounds=(0, 1))

m.c_A = Var(m.tau, domain=NonNegativeReals)
m.c_B = Var(m.tau, domain=NonNegativeReals)
m.c_C = Var(m.tau, domain=NonNegativeReals)
m.d_c_A = DerivativeVar(m.c_A)
m.d_c_B = DerivativeVar(m.c_B)
m.d_c_C = DerivativeVar(m.c_C)

# new variable: residence time
m.t_f = Var(domain=NonNegativeReals)

def ode_A(m, tau):
    if tau > 0:
        return m.d_c_A[tau] == (- m.t_f * k_A * m.c_A[tau])
    else:
        return Constraint.Skip
m.ode_A = Constraint(m.tau, rule=ode_A)
def ode_B(m, tau):
    if tau > 0:
        return m.d_c_B[tau] == m.t_f * (k_A * m.c_A[tau] - k_B * m.c_B[tau])
    else:
        return Constraint.Skip
m.ode_B = Constraint(m.tau, rule=ode_B)
def ode_C(m, tau):
    if tau > 0:
        return m.d_c_C[tau] == m.t_f * k_B * m.c_B[tau]
    else:
        return Constraint.Skip
m.ode_C = Constraint(m.tau, rule=ode_C)

m.ic = ConstraintList()
m.ic.add(m.c_A[0] == c_A0)
```

```python
m.ic.add(m.c_B[0] == 0)
m.ic.add(m.c_C[0] == 0)

# add obj
m.obj = Objective(expr=m.c_B[1] * 10 + m.c_C[1] * 3, sense=maximize)

# transform dae model to algebraic equations
TransformationFactory('dae.collocation').apply_to(m)

# solve the model
SolverFactory('ipopt').solve(m)

# print solution
print(f'Optimal profit: ${value(m.obj):.2f}, optimal residence time: {value(m.t_f):.2f}
↪  min.')
```

```
Optimal profit: $14.70, optimal residence time: 4.76 min.
```

5.2 Optimality conditions for NLP

5.2.1 Hessian

Definition 1 (Hessian)**.** Consider a function $f : \mathbb{R}^n \to \mathbb{R}$, then its Hessian is

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \cdots & & & \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- Hessian is always a symmetric matrix;
- The signs of its eigenvalues determines if the Hessian is positive/negative (semi)definite.

Relationship among Hessian eigenvalues, Hessian positive-definiteness and function convexity: Table 2.[15]

5.2.2 Optimality conditions for unconstrained function

Sufficient optimality conditions for one-dimentsion f:

1. f is twice differentiable[16]
2. *first order necessary* optimality condition: $\frac{\mathrm{d}f}{\mathrm{d}x} = 0$; used to identify stationary point
3. *second order necessary* optimality condition: $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} > 0$ for minimization problem

[15]Difference between strictly convex and convex: the former indicates there is only one global optimum, the latter indicates there can be multiple global optimums with the same optimal values.

[16]This means that we want to avoid or reformulate functions that are not (twice) differentiable, e.g., absolute value functions

Table 2: Relationship among Hessian eigenvalues, Hessian positive-definiteness and function convexity

Hessian eigenvalues	Hessian positive-definiteness	function convexity
all positive	positive definite	strictly convex
all nonnegative	positive semi-definite	convex
all negative	negative definite	strictly concave
all nonpositive	negative semi-definite	concave
otherwise	-	nonconvex

- If the sufficient optimality conditions hold at a point x^*, then x^* is at least a local optimum
- If the second order necessary optimality condition holds through the whole domain of f, then f is convex, and x^* is the global optimum
- Otherwise we need to check other local optimums and end points to find global optimums

Sufficient optimality conditions for n-dimentsion f:

1. f is twice differentiable
2. *first order necessary* optimality condition:

$$\nabla f_{\mathbf{x}=\mathbf{x}^*} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \dots & \frac{\partial f}{\partial x_n} \end{bmatrix}\Big|_{\mathbf{x}=\mathbf{x}^*} = \mathbf{0}.$$

3. *second order necessary* optimality condition:

$$\nabla^2 f|_{\mathbf{x}=\mathbf{x}^*} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \dots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}\Bigg|_{\mathbf{x}=\mathbf{x}^*} \quad \text{is positive semi-definite (PSD).}$$

5.3 Convexity, Part 2

Preservation of convexity: if f_1, f_2 are convex,

- f_1, f_2 is convex.
- λf_1 is convex if $\lambda > 0$
- $g \circ f$ is convex if g is monotonically increasing

5.4 Optimality conditions for constrained optimization

Basic idea: reformulate constrained problems via Lagrangean relaxation to unconstrained problems, then apply optimality conditions.

5.4.1 Variable elimination

$$\begin{aligned} &\min\ x_1 \cdot x_2 \cdot x_3 \\ &\text{s.t. } x_1 + x_2 + x_3 = 0 \end{aligned} \overset{x_3=-x_1-x_2}{\Longrightarrow} \min\ x_1 \cdot x_2 \cdot (-x_1 - x_2)$$

Can be applied when an equality constraint can be rearranged to explicitly represent a variable to be eliminated.

5.4.2 Lagrange function

Definition 2 (Lagrange function)**.** For a constrained optimization problem

$$\begin{aligned} &\min\ f(\mathbf{x}) \\ &\text{s.t. } h_i(\mathbf{x}) = 0, i = 1, \ldots, M, \\ &\quad\ g_j(\mathbf{x}) \leq 0, j = 1, \ldots, N \end{aligned} \tag{P1}$$

we can define an unconstrained optimization problem with a new objective function called *Lagrange function*[17]by introducing new variables for each constraint called *Lagrange multipliers*:

$$\min_{\mathbf{x},\boldsymbol{\lambda},\boldsymbol{\mu}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}),\ \text{ where } \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \sum_{i=1}^{M} \lambda_i h_i(\mathbf{x}) + \sum_{j=1}^{N} \mu_j g_j(\mathbf{x}). \tag{P2}$$

Example 2. Consider the problem

$$\begin{aligned} &\min\ x_1^2 + x_2^2 \\ &\text{s.t. } 2x_1 + x_2 - 2 = 0. \end{aligned}$$

We can reformulate it as an unconstrained problem by defining Lagrange function

$$\min \mathcal{L}(x, \lambda) = x_1^2 + x_2^2 + \lambda(2x_1 + x_2 - 2).$$

- For a fixed λ, consider the first order necessary optimality condition:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial x_1} = 2x_1 + 2\lambda = 0 \\ \frac{\partial \mathcal{L}}{\partial x_1} = 2x_2 + \lambda = 0 \end{cases} \Longrightarrow \begin{cases} x_1^* = -\lambda \\ x_2^* = -0.5\lambda \end{cases}$$

- For a fixed λ, consider the second order necessary optimality condition:

$$\nabla^2 f = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},$$

  which is positive definite. So f is convex.
- If the original constraint also holds for the unconstrained problem, then it is *identical to the original problem*, because

[17]Sometimes also called Lagrangean or Lagrangian function.

- (x_1^*, x_2^*) is feasible to the original problem
- the $\lambda(2x_1+x_2-2)$ term becomes 0, so identically we are minimizing $x_1^2+x_2^2$, which is identical to the original problem

Assuming the original constraint holds, then

$$2(-\lambda) - 0.5\lambda - 2 = 0 \Longrightarrow \begin{cases} \lambda^* = -0.8 \\ x_1^* = 0.8 \\ x_2^* = 0.4 \end{cases}.$$

5.4.3 KKT conditions

Interpretation of Lagrange multipliers The change of objective value w.r.t. unit change of RHS of constraints, i.e., the change rate of objective value in terms of constraint deviation.

Example 3. Consider the problem

$$\begin{aligned} &\min\ x_1^2 + x_2^2 \\ &\text{s.t. } 2x_1 + x_2 - 2 = b. \end{aligned}$$

We can reformulate it as an unconstrained problem by defining Lagrange function

$$\min \mathcal{L}(x, \lambda) = x_1^2 + x_2^2 + \lambda(2x_1 + x_2 - 2 - b).$$

Following the similar steps in the previous example, we can get

$$\begin{cases} \lambda^* = -0.8 - 0.4b \\ x_1^* = 0.8 + 0.4b \\ x_2^* = 0.4 + 0.2b \end{cases},$$

which is a function of b.

The derivative of the objective function w.r.t. b at the optimal solution is

$$\left.\frac{\partial \mathcal{L}}{\partial b}\right|_{x_1^*, x_2^*, \lambda^*} = -\lambda.$$

Active/inactive constraints At a given point $\mathbf{x}^*$,

- if the equality holds for an inequality constraint, i.e., $g_j(\mathbf{x}^*) = 0$, then we call this constraint *active* at $\mathbf{x}^*$.
- if the inequality holds for an inequality constraint, i.e., $g_j(\mathbf{x}^*) < 0$, then we call this constraint *inactive* at $\mathbf{x}^*$.

Definition 3 (Karush-Kuhn-Tucker (KKT) conditions/necessary optimality conditions for constrained problems)**.** For problem (P1), if the objective function and the constraints are differentiable, we call the following equations the *KKT condition* at a given point $\mathbf{x}^*$:

$$\begin{array}{ll}
\nabla f(\mathbf{x}^*) + \boldsymbol{\lambda}^{\mathrm{T}}\nabla \mathbf{h}(\mathbf{x}^*) + \boldsymbol{\mu}^{\mathrm{T}}\nabla \mathbf{g}(\mathbf{x}^*) = 0 & \text{(stationary point-first-order derivative of } \mathcal{L} \text{ w.r.t } \mathbf{x}) \\
\mathbf{h}(\mathbf{x}^*) = \mathbf{0} & \text{(feasibility-original constraints)} \\
\mathbf{g}(\mathbf{x}^*) \le \mathbf{0} & \text{(feasibility-original constraints)} \\
\mu_j g_j(\mathbf{x}^*) = 0, j = 1, \dots, N & \text{(complementary constraints)} \\
\mu_j \ge 0, j = 1, \dots, N & \text{(complementary constraints)}
\end{array}$$

If $\mathbf{x}^*$ satisfies the KKT condition, then it is a feasible and stationary point for (P1).

Complementary constraints These constraints ensures that inequality constraints does not affect the value of the Lagrange function whether or not the constraints are active or inactive:

- If g_j is active at $\mathbf{x}^*$, then $g_j(\mathbf{x}^*) = 0$, the $\mu_j g_j(\mathbf{x}^*)$ term in $\mathcal{L}$ is 0
- If g_j is inactive at $\mathbf{x}^*$, then the $\mu_j g_j(\mathbf{x}^*)$ term in $\mathcal{L}$ is still 0 as the complementary constraints force μ_j to be 0

convexity in constrained problems If the objective function and all constraints are convex w.r.t. $\mathbf{x}$, then the problem is also convex; if any of them is nonconvex, then the problem is nonconvex.

5.5 Duality

Definition 4 (primal/dual problem)**.** Consider a general nonlinear, constrained optimization problem

$$\begin{aligned} &\min f(\mathbf{x}) \\ &\text{s.t. } \mathbf{h}(\mathbf{x}) = \mathbf{0}. \\ &\quad\;\; \mathbf{g}(\mathbf{x}) \le \mathbf{0} \end{aligned} \tag{P}$$

Assume $f, \mathbf{h}, \mathbf{g}$ are continuous and relatively smooth (derivatives exists and are bounded), and the problem is feasible. We call this problem the *primal problem.* Its *dual problem* is a maximization problem of its Lagrange function w.r.t. Lagrange multipliers, with an inner minimization problem w.r.t. $\mathbf{x}$:[18]

$$\max_{\boldsymbol{\lambda}, \boldsymbol{\mu} \ge \mathbf{0}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}), \text{ where } \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) + \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{h}(\mathbf{x}) + \boldsymbol{\mu}^{\mathrm{T}}\mathbf{g}(\mathbf{x}). \tag{D}$$

$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) \equiv \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ is called *dual function.*

Example 4. Consider the primal problem

$$\begin{aligned} &\min_x \; (x-1)^2 \\ &\text{s.t. } 2x - 1 = 0 \end{aligned},$$

[18]We assume that optimums of both the inner problem and the outer problem always exist. The existence of these optimums is out of the scope of this class.

its dual problem is

$$\max_{\lambda} \min_{x} \ (x-1)^2 + \lambda(2x-1).$$

Clearly the optimal solution of the primal problem is $x^* = 0.5$, with objective value 0.25.

For the dual problem, we first consider the stationary condition for $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial x} = 2x - 2 + 2\lambda \Longrightarrow x^* = 1 - \lambda.$$

Also $\frac{\partial^2 \mathcal{L}}{\partial x^2} = 2 > 0$, so $\mathcal{L}$ is convex w.r.t. x, and x^* is a global minimum.

Plugging x^* back into the dual problem:

$$\max_{\lambda} (1-\lambda-1)^2 + \lambda(2(1-\lambda)-1) = \max_{\lambda} -\lambda^2 + \lambda.$$

Its maximum is 0.25 when $\lambda^* = 0.5$. Meanwhile, $x^* = 1 - \lambda^* = 0.5$.

The optimal x^* and objective value are identical to the primal problem.

5.5.1 Graphical representation of duality

Consider the following problem:

$$\begin{array}{l} \min \ f(x) \\ \text{ s.t. } g(x) \leq 0, x \in X, \end{array} \quad \text{let } y \equiv g(x), z \equiv f(x) \Longrightarrow \quad \begin{array}{l} \min \ z \\ \text{ s.t. } y \leq 0. \end{array}$$

Define $G \equiv \{(y,z) : \exists x \in X \text{ s.t. } y = g(x), z = f(x)\}$, i.e., G is the collection of all potential values of y and z corresponding to X. Then we can plot y, z in Figure 2. And to minimize the original problem, we want to find the lowest point in the LHS of G.

Consider the dual problem:

$$\max_{\mu \geq 0} \min_{x} \ f(x) + \mu(g(x)) = \max_{\mu \geq 0} \min \ z + \mu y.$$

- The dual function is $\phi(\mu) = \min \ z + \mu y$. Assume the optimal objective value is α, then the visualization of the solution in the (y,z) space is a line touching G with slope $-\mu$ (LHS of Figure 3). The minimization can be seen as moving the line "downwards" while letting it still touch G.
- The maximization of the dual problem can be seen as changing the slope of the line such that the intercept is maximized (RHS of Figure 3).

5.5.2 Properties of primal-duals

Assume the primal is a minimization problem. Denote the optimum of (P) as $P(\mathbf{x}^*)$, and the optimum of (D) as $D(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$.

- If (P) is linear, (D) is also linear, and $P(\mathbf{x}^*) = D(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$

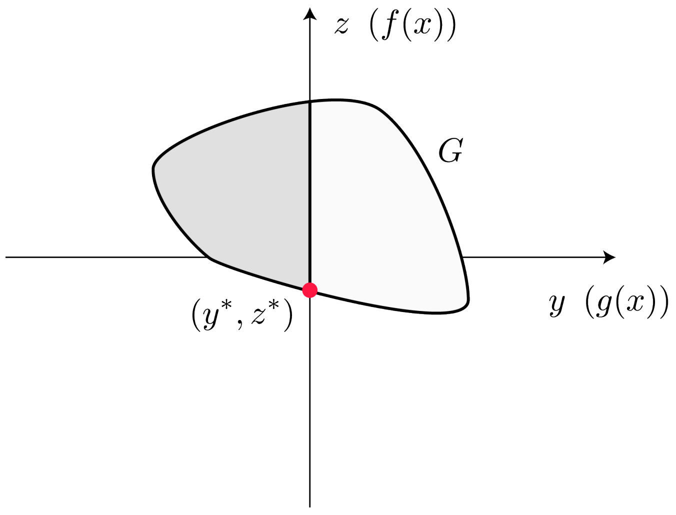


Figure 2: Graphical representation of the primal problem

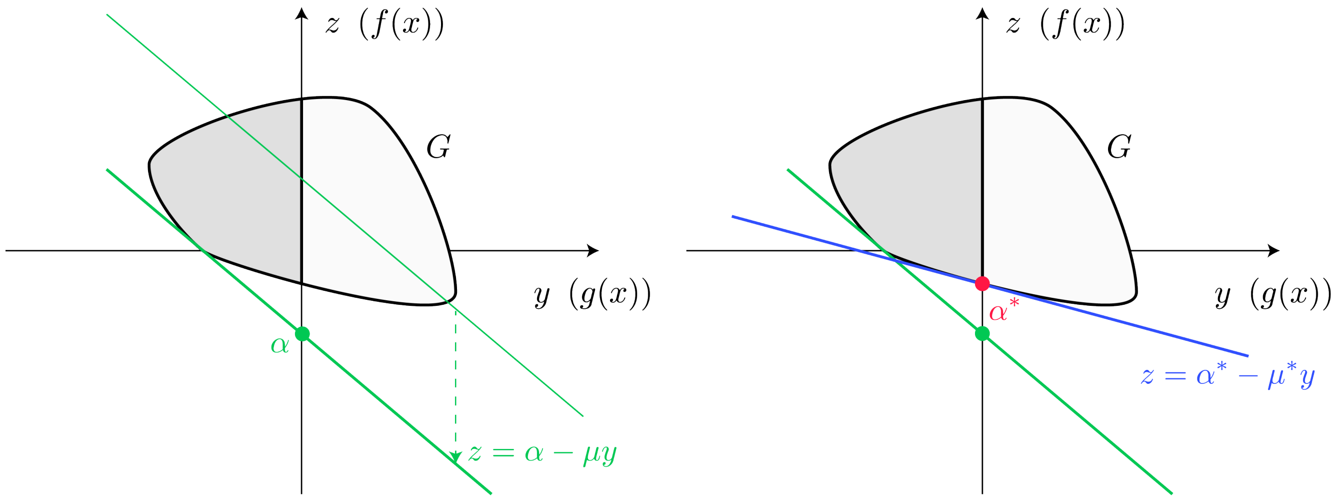


Figure 3: Graphical representation of the dual problem

- If (P) is convex, (D) is also concave, and $P(\mathbf{x}^*) = D(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$
- If (P) is nonconvex, but is continuous, differentiable and numerically stable, then $P(\mathbf{x}^*) \geq D(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$; i.e., the optimum of dual is a lower bound of the optimum of primal

6 Week 5: Mixed-integer nonlinear optimization

6.1 MINLP introduction and challenges

Applications:

- Optimization with embedded machine learning models (regression trees, ReLu networks)
- Flowsheet synthesis
- Production planning
- Scheduling
- Supply chain management
- Water distribution network design

Challenges:

1. combinatorial difficulty + nonlinearity
2. fewer solvers available
3. even with available solvers, the solving process can be prohibitively expensive

Solvers:

- BARON: global solver, best commercial solver
- ANTIGONE: global
- DICOPT: global for convex MINLPs
- Couenne: global

Solving strategies:

- MINLP problems can be convex or nonconvex (in terms of its constraints)
- *decomposition* idea: when integer variables are fixed, the rest problem can be convex, linear, or quadratic-easier to solve
- *convexifying* idea: nonconvex terms can be under/over-estimated by convex terms-can be used to bound the original problem

6.2 Mixed-integer formulations

6.2.1 Logic representation

let y_i be a binary variable, $i \in S$. *It is usually be used to represent "yes/no" decisions, and constraints containing binary variables "if... else..." conditions.* For example, y_i denote if reactor i is chosen ($y_i = 1$) or not chosen ($y_i = 0$).

- at least/at most/exactly m reactors are chosen:

$$\sum_{i\in S} y_i \geq / \leq / = m.$$

- if reactor A is chosen, then its volume v_A must be within 5 to 10 L, otherwise its value should be 0:

$$5y_\text{A} \leq V_\text{A} \leq 10y_\text{A}$$

  It works as follows:
  - when $y_\text{A} = 1, v_\text{A} \in [5, 10]$;
  - when $y_\text{A} = 0, v_\text{A} \in [0, 0] \Longrightarrow v_\text{A} = 0$.

6.2.2 Logical operators

Logical operators are unary/binary operators on logic statements (which can be represented by constraints with binary variables); logic statement containing logical operators can also be represented by constraints with binary variables.

Let Y_1, Y_2 denote two logic statements, and y_1, y_2 denote two binary variables corresponding to the statements. Below we give the definition of each logical operator and the binary variable representation when the overall statement is true.

- negate: $\neg Y_1$, which is true only when Y_1 is false
  binary variable representation: $1 - y_1 \geq 1$ or $(1 - y_1 = 1)$
- logic and: $Y_1 \wedge Y_2$, which is true only when both Y_1 and Y_2 are true
  binary variable representation: $y_1 \geq 1, y_2 \geq 1$ (or $y_1 + y_2 \geq 2$, or $y_1 = 1, y_2 = 1$)
- logic and: $Y_1 \wedge Y_2$, which is true only when either Y_1 or Y_2 is true
  binary variable representation: $y_1 + y_2 \geq 1$
- logic exclusive or (xor): $Y_1 \otimes Y_2$, which is true only when exactly one of Y_1 and Y_2 is true, and exactly one of them is false
  binary variable representation: $y_1 + y_2 = 1$
- implies: $Y_1 \Rightarrow Y_2$, which is true except Y_1 is true and Y_2 is false (the whole implication statement is true when (1) Y_1 is true, Y_2 is true; (2) Y_1 is false, Y_2 is true; (3) Y_1 is false, Y_2 is false)
  binary variable representation: $y_1 \leq y_2$

6.3 Global optimization methods

6.3.1 Relaxation

Relaxation

- an simplified and easier optimization problem compared with the original one
- Its feasible space $\mathcal{F}$ should include the original feasible space $\mathcal{F}_0$, and is typically larger

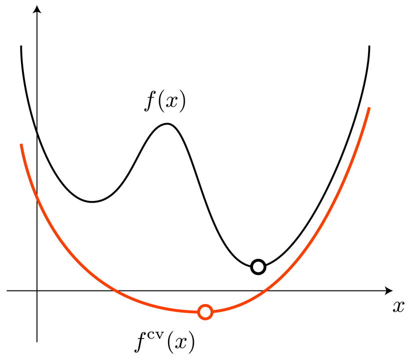
Relaxation methods:

1. removing some constraints (relaxing feasible region)
2. relax binary variables from $\{0, 1\}$ to $[0, 1]$
3. use under/over-estimators of original constraints

Validity of relaxation:

- Is it easier to solve than the original problem?
- Does its feasible region include the feasible region of the original problem?

Example 5. convex underestimator of nonconvex objective function: easier to solve, with the same feasible region



6.3.2 Bounding

Relaxation provides lower bounds for the original problems (for minimizations): As $\mathcal{F} \supset \mathcal{F}_0$, the optimal solution of the original problem x_0^* is also in $\mathcal{F}$. In other words, x_0^* is also a feasible solution to the relaxation, and the optimal solution of the relaxation x^* must be no worse than x_0^*.

Example 6. Consider an MINLP problem:

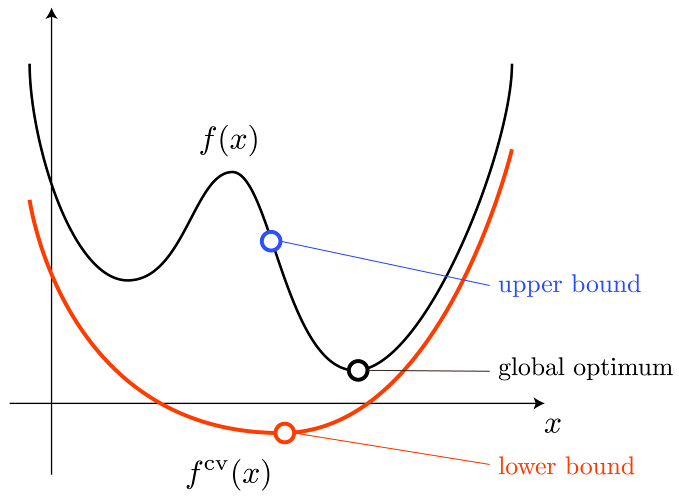
$$\begin{aligned} \min\ & y_1^2 + y_2 \\ \text{s.t. } & y_1 + y_2 \geq 1 \\ & y_1, y_2 \in \{0, 1\} \end{aligned}$$

If we relax the binary variables to continuous variables, we get an NLP problem:

$$\begin{aligned} \min\ & y_1^2 + y_2 \\ \text{s.t. } & y_1 + y_2 \geq 1 \\ & y_1, y_2 \in [0, 1] \end{aligned}$$

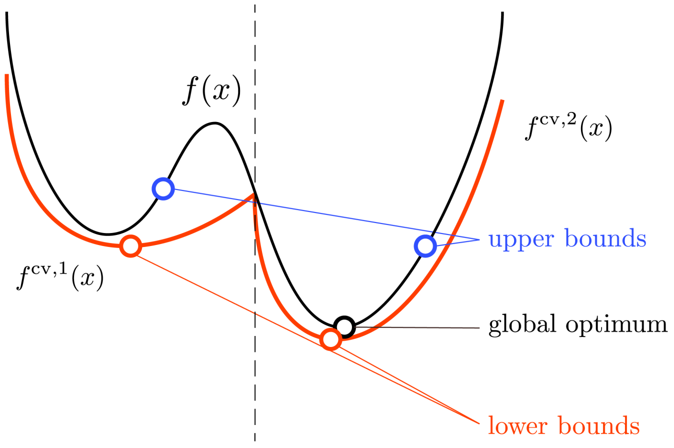
The optimal solution of the NLP is a lower bound for the optimal solution of the MINLP.

Any feasible solution provides an upper bound for the original problem (for minimization), as the optimal solution must be no greater than any feasible solution.



6.3.3 Basic premise

- Solving a relaxation and locally solving[19] the original problem can give us a lower bound and an upper bound, and let us know an interval containing the best objective value.
- It is difficult to find the perfect lower bound in one iteration
- To further refine the interval, we can either (i) find tighter relaxation, or (ii) decompose the feasible region, and solve relaxations and locally solve the original problems in smaller spaces
- The global optimum is obtained (the global optimization algorithm converges) when the lower bound and the upper bound[20]have the same value.



6.4 Branch and bound

Basic idea:

[19]Actually you do not even need to locally solve it-any feasible solution can work as an upper bound

[20]Because we have various lower bounds and upper bounds for different subspaces, here we mean the best (greatest) lower bound and the best (least) upper bound.

- sequentially *branch* in the feasible region, and solve multiple relaxations in subspace
- use *bounding* to find the global optimum

Basic rules:

- branching:
  - For binary variables, fix it twice (to 0 and to 1) to generate two subproblems
  - For continuous variables, partition its feasible region to two smaller regions
- pruning: eliminate branches that are not worth exploring

Steps (for MILP minimization)[21]:

(1) Formulate fully-relaxed problem (relax all binary variables to continuous variables) at the root node (node 0)
(2) Find lower bound (LB) of node 0 by solving the relaxed LP problem
(3) Find any feasible solution as upper bound (UB). If UB = LB, terminate; otherwise, go to (4).
(4) Branch on one variable by fixing it to 0 or 1. Solve the corresponding new problem, and identify UB_i or LB_i for node i
   - If $\text{LB}_i \geq \text{UB}$, prune node
   - If node i results in UB_i (integer solution), prune node
(5) Update UB and LB with the best available bounds in all nodes. If UB = LB, terminate; otherwise, go to (4).

Pruning rules:

1. relaxation gives feasible solutions of original problem: we no longer need to further branch, as this is already the optimal solution
2. the lower bound of the node is greater than current best upper bound (for minimization problems): we no longer need to further branch, as solutions of its subnodes cannot be better than the upper bound of other nodes

Branch heuristics:

- which variable to branch on
- what value to branch on
- which order to evaluate nodes in the tree
  - depth-first: choose node and investigate all subnodes until end or pruned
  - width-first: evaluate all nodes in the same level first before moving to the next level
- Best heuristics are problem-specific, no general “better” ones

Branch and bound on continuous variables:

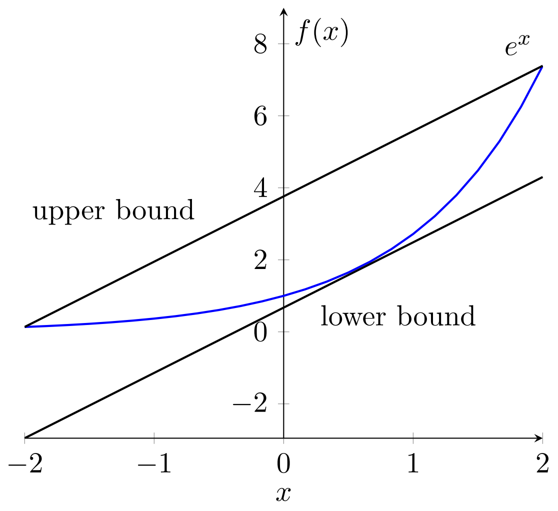
- necessary for nonconvex (MI)NLP problems
- the algorithm usually called spatial branch and bound
- branching partition its feasible region to two smaller regions
- bounding: by solving (usually convex) relaxations (next section)
- Ideally, the relaxation should be tighter on a smaller region, so that the algorithm can finally converge

[21]Basically all the descriptions in the notes are for minimization problems, which can be different from the videos which sometimes deal with maximization problems

- Various methods exist for relaxation

6.5 Bounding functions

Example 7. Exponential function e^x can be bounded by two linear functions



6.5.1 McCormick relaxation for bilinear terms

Let $w = xy$, where $x \in \left[x^{\mathrm{L}}, x^{\mathrm{U}}\right], y \in \left[y^{\mathrm{L}}, y^{\mathrm{U}}\right]$, then it can be bounded by a set of linear constraints:

$$\begin{aligned} w &\geq x^{\mathrm{L}}y + xy^{\mathrm{L}} - x^{\mathrm{L}}y^{\mathrm{L}} \\ w &\geq x^{\mathrm{U}}y + xy^{\mathrm{U}} - x^{\mathrm{U}}y^{\mathrm{U}} \\ w &\leq x^{\mathrm{L}}y + xy^{\mathrm{U}} - x^{\mathrm{L}}y^{\mathrm{U}} \\ w &\leq x^{\mathrm{U}}y + xy^{\mathrm{L}} - x^{\mathrm{U}}y^{\mathrm{L}} \end{aligned}$$

A good 3D visualization of McCormick relaxation for bilinear terms is available here (original function on P2, relaxation on P4).