Data-Drive Process Systems Engineering: Summarized Notes

Update date: March 1, 2022

Contents

1 Big picture

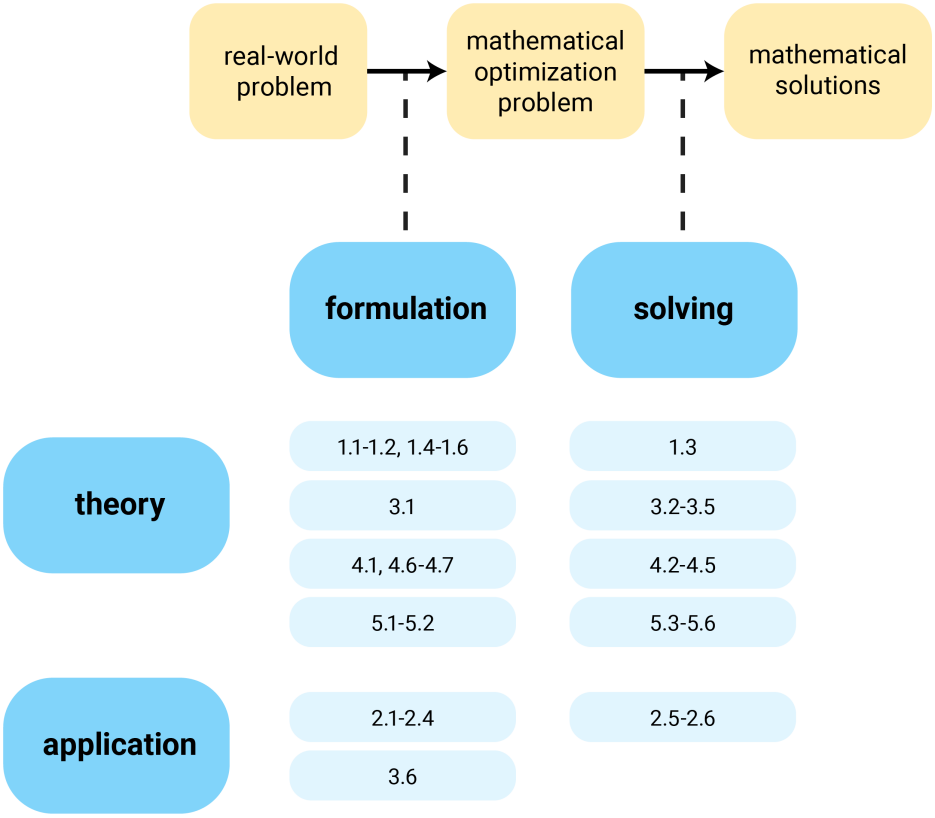


Figure 1: Big picture

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

$\max/\min_{\text{variables}}$ objective function	$\min_{x,y} x + y$
subject to constraints	s.t. $x - y \geq 20$
bounds	$x \geq 0, y \geq 0$

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

$$a_1 x_1 + a_2 x_2 + \ldots a_n x_n = x$$
$$a_1 + a_2 + \ldots a_n = 1.$$

2.3 Formulation types

Linear optimization problem:

- both objective and constraints are linear w.r.t. all variables
- generally follow this form: $c_1 x_1 + c_2 x_2 + \ldots$, 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 $g_i(\mathbf{x}) \leq 0$ 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. $f(\mathbf{x}^*) \leq f(\mathbf{x}) \, \forall \, \mathbf{x} \in S$, with S being the feasible region
- "true" optimum

Local minimizer:

- solution $\mathbf{x}^\dagger$ s.t. $f(\mathbf{x}^\dagger) \leq f(\mathbf{x}) \, \forall \, \mathbf{x} \in \left\{ x : \left\| x - x^\dagger \right\| \leq \varepsilon \right\}$
- 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

```
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)
```

```
9
10  # fetch the length of the set
11  len(model.i)
12
13  # apply the logical operators on sets
14  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

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

```
1  # initialize variable z indexed by set i, requiring it to be nonnegative real
2  # numbers
3  model.z = pe.Var(model.i, initialize={1:1.5, 2:2.5, 3:3.5, 4:4.5},
4  within=pe.NonNegativeReals)
5
6  # access z[1] upper bound
7  model.z[1].ub
8  # dupate z[1] upper bound
9  model.z[1].setub(20)
10
11 # fix z[2]
12 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:

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

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

```
1  # define Python function for constraint rule
2  def diff_rule(m, i):
3    # conditional skip
4    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:

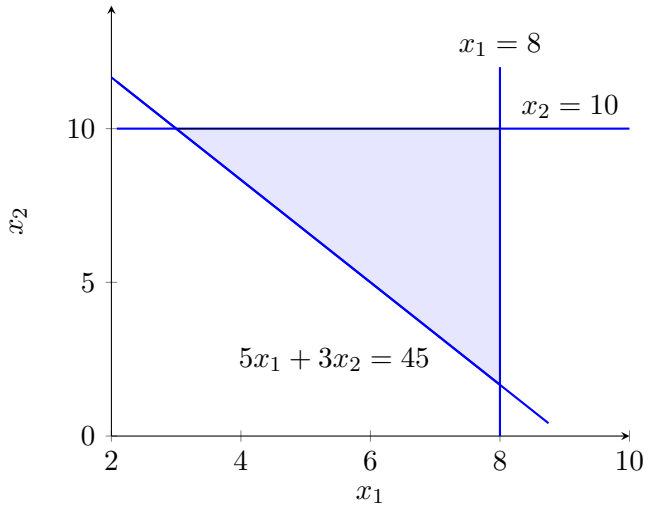
- If an LP has an optimal solution, then at least one of the corner points is optimal
- we only need to check a finite number of corner points to find an optimal solution

4.2.1 Graphical solutions of LP

Problem:

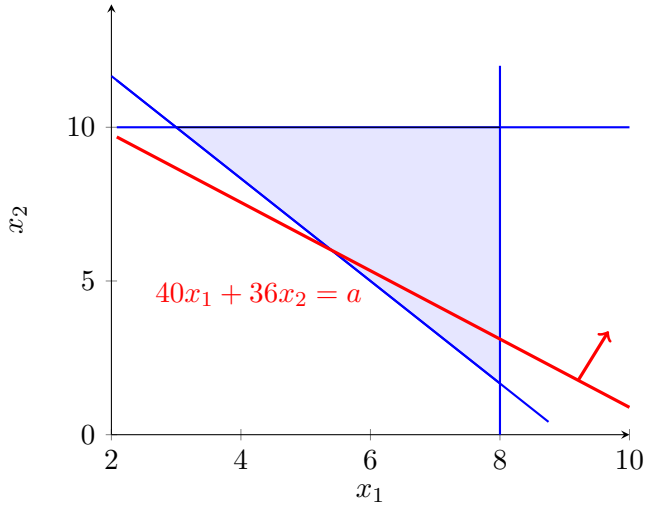
$$\begin{aligned} \max\ & 40x_1 + 36x_2 \\ \text{s.t. } & 5x_1 + 3x_2 \geq 45 \\ & x_1 \in [0, 8], x_2 \in [0, 10] \end{aligned}$$

Step 1: draw feasible region



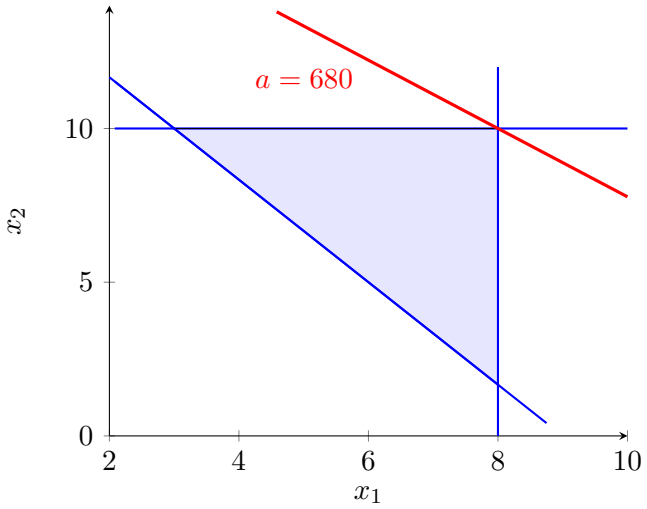
Step 2: draw objective function

- 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

$$
\begin{aligned}
& a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1 \\
& a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2 \\
& \cdots \\
& a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n = b_m \\
& x_1, x_2, \ldots, x_n \geq 0 \\
& \qquad\qquad \Downarrow \\
x_1+ \quad & \bar{a}_{1,m+1}x_{m+1} + \bar{a}_{1,m+2}x_{m+2} + \cdots + \bar{a}_{1,n}x_n = \bar{b}_1 \\
x_2+ \quad & \bar{a}_{2,m+1}x_{m+1} + \bar{a}_{2,m+2}x_{m+2} + \cdots + \bar{a}_{2,n}x_n = \bar{b}_2 \\
& \cdots \\
x_m+ & \bar{a}_{m,m+1}x_{m+1} + \bar{a}_{m,m+2}x_{m+2} + \cdots + \bar{a}_{m,n}x_n = \bar{b}_m \\
& x_1, x_2, \ldots, x_n \geq 0
\end{aligned}
$$

- 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

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} c_i x_i + c_s x_s \\ & x_1 + \qquad\qquad \bar{a}_{1,s} x_s = \bar{b}_1 \\ \text{s.t.} \quad & \quad x_2 + \qquad \bar{a}_{2,s} x_s = \bar{b}_2 \\ & \qquad\qquad\qquad \cdots \\ & \qquad\qquad\quad x_n + \bar{a}_{2,s} x_s = \bar{b}_n \end{aligned}$$

- when x_s changed a unit, $x_i = \bar{b}_i - \bar{a}_{i,s}, i = 1, \ldots, m$; 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: $\mathrm{A} \xrightarrow{k_1} \mathrm{B} \xrightarrow{k_2} \mathrm{C}$, with reaction rates:

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

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.

```
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.

```
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: $P = 10 \cdot C_{\mathrm{B,final}} + 3 \cdot C_{\mathrm{C,final}}$

- non-dimensionalize the differential equations

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

- 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)
```

```
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:

$$\nabla f_{\mathbf{x}=\mathbf{x}^*} = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_n}\right]\Big|_{\mathbf{x}=\mathbf{x}^*} = \mathbf{0}.$$

3. *second order necessary* optimality condition:

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

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

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

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 $\mathbf{x}^*$:

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

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

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

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]

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

$\phi(\boldsymbol{\lambda}, \boldsymbol{\mu}) \equiv \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ 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 $P(\mathbf{x}^*) = D(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$

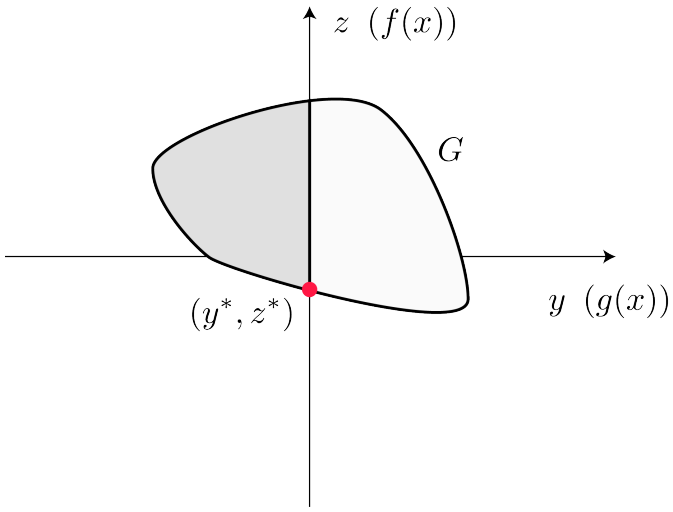


Figure 2: Graphical representation of the primal problem

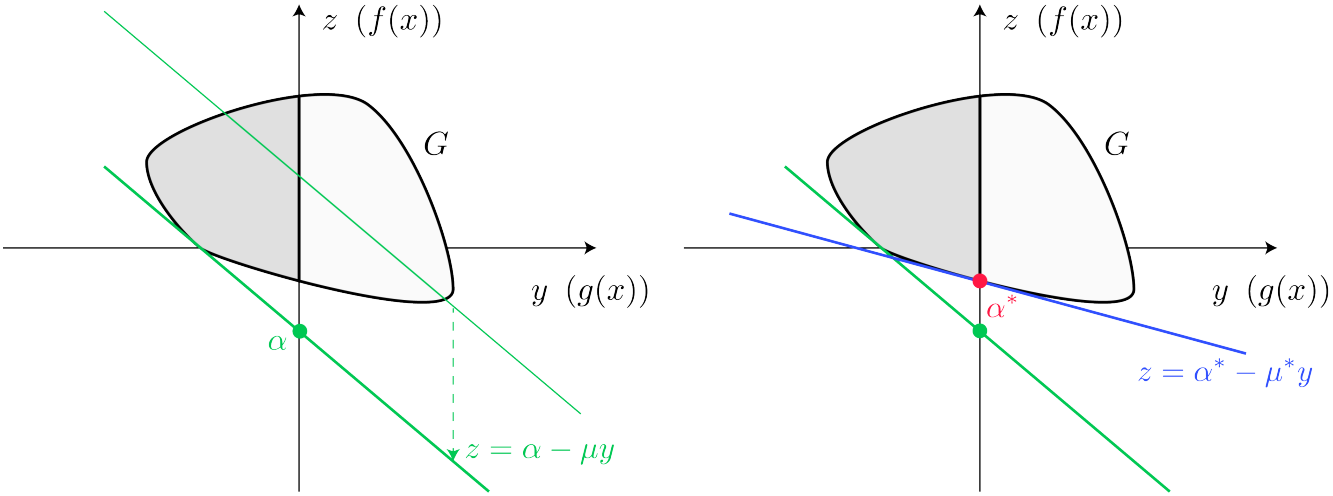


Figure 3: Graphical representation of the dual problem

- 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:

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

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

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

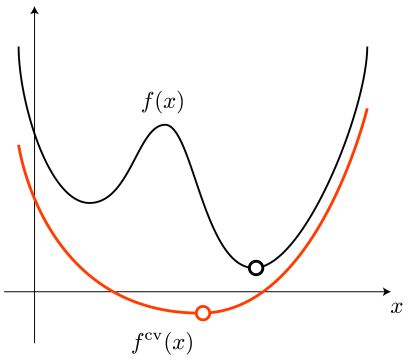
Relaxation methods:

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:

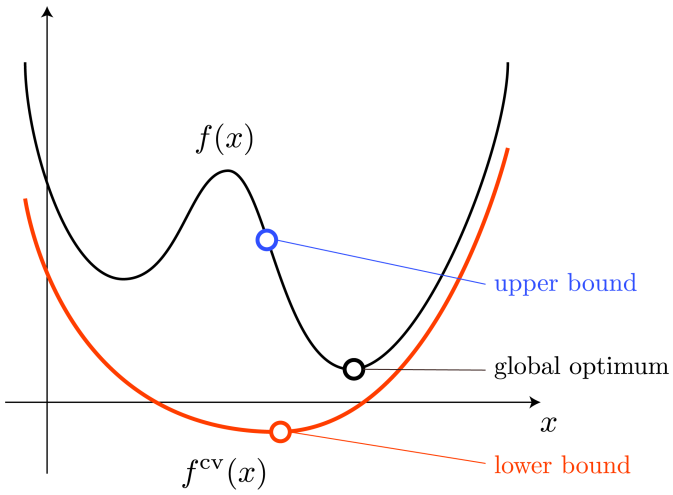
$$\begin{aligned} \min\ & y_1^2 + y_2 \\ \text{s.t. } & y_1 + y_2 \geq 1 \\ & y_1, y_2 \in \{0, 1\} \end{aligned}$$

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

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

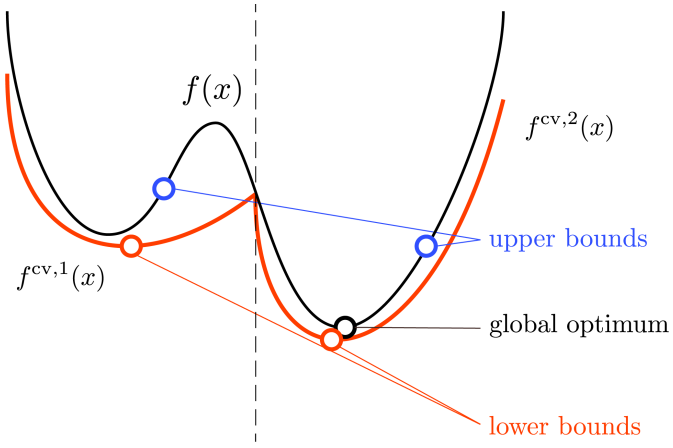
The optimal solution of the NLP is a lower bound for the optimal solution of the MINLP.

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 $LB_i \geq UB$, 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:

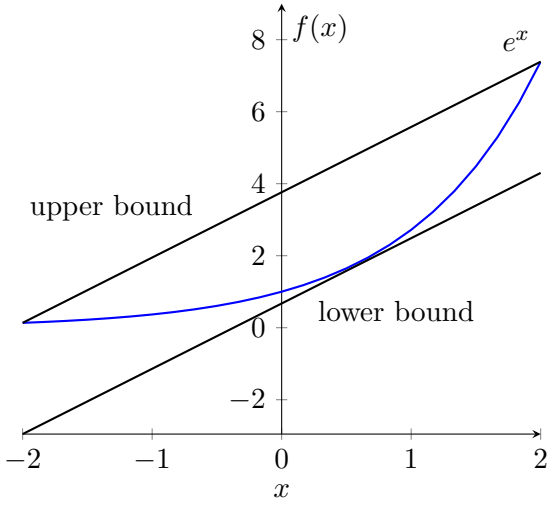
- necessary for nonconvex (MI)NLP problems
- the algorithm usually called spatial branch and bound
- branching partition its feasible region to two smaller regions
- bounding: by solving (usually convex) relaxations (next section)
- Ideally, the relaxation should be tighter on a smaller region, so that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

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

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

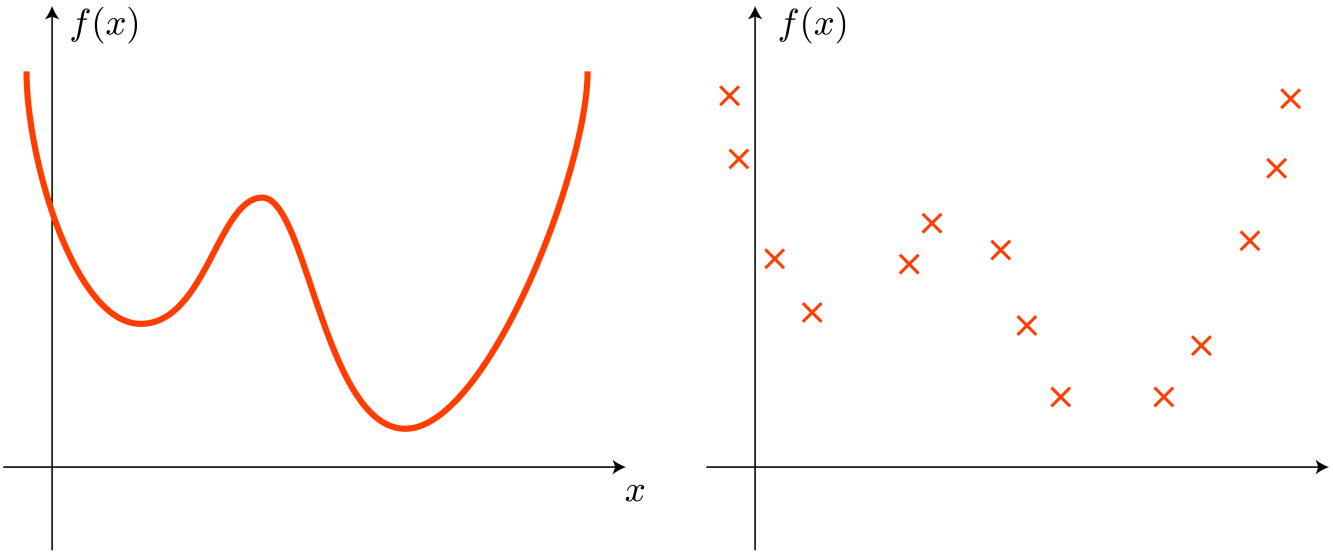
7 Week 6: Introduction to data-driven optimization

7.1 Introduction

Definition 5 (Black-box optimization)**.** We call an optimization problem *black-box* when the objective and a subset or all of the constraints are not available in closed form.

It is also called derivative-free optimization, simulation-optimization or data-driven optimization[22].

[22]In some literature, data-driven optimization may refer to optimization with uncertain parameters. These problems have



objective in closed form

objective without closed form

Figure 4: Objective function with or without closed form

History on black-box optimization:

- Earliest development: 1960s, motivated by expensive derivative calculation and noise in data back then
- Nowadays many applications: benefit from advanced simulation and extensive data

Data source:

- experimental data
- computational data
- some first-principle equations, making the problem actually grey-box

Case studies:

- Oilfield operations: the relationship between injected water pressure and oil extraction is available through intricate reservoir simulation; goal: maximize net profit value
- Supply chain of CO2 capture: the capture process is modeled in ASPEN; goal: minimize the overall cost; variables: where to build new capture plants, which technology to invest on
- Multi-scale engineering: data-drive optimization is necessary to link processes with different time and space scales

7.2 Data-driven optimization (DDO) types

Data availability and usage:

closed form for all equations available, but the values of the parameters are not deterministic. Instead they follow certain probability distributions, which can be described by data

- can be collected randomly, or from a designed experiment
- can be collected within the whole region, or just part of that
- can be used directly and compared against each other for better points, or can be used to fit models within optimization

7.2.1 Sample driven DDO (direct search) vs. Model-based DDO

Direct search:

- directly compare data values in the space of objective function
- pattern search

Model-based:

- use samples to fit approximate models, then optimize them

7.2.2 Search strategy: local vs. global

Direct search:

- local search: only sample data in small part of search space
- global search: sample data in entire search space

Model-based DDO:

- local search: only sample data and fit model in small part of search space
- global search: sample data and fit model in entire search space

7.2.3 Pros and cons

Direct search:

- pros:
  - no extra cost of fitting models
  - simple implementations
  - easy to conceptualize
- cons:
  - may require too much data
  - no equations or derivatives

Model-based DDO:

- pros:
  - model has equations and derivatives
  - model can be used for prediction (less sampling)
- cons:
  - fitting model is costly and creates uncertainty
  - wrong model could lead to wrong directions

7.3 Optimality in DDO

The following contents are w.r.t. problems with ability to collect extra data at "designed" locations.

7.3.1 Overview

Difference between derivative-based optimization and DDO:

- Unknown information: problem type, derivative information
- Previous knowledge framework does not directly apply
- Potentially unknown information: variables, variables that affect objective and constraints

Convergence mechanisms:

- It is impossible to guarantee convergence to global optimum for truly black-box problems (we cannot assume anything about formulation), unless with complete enumeration or infinite sampling
- Goal: find the best optimum possible with given resources (computational capabilities, data availability, etc.)
- convergence (termination) mechanism depends on sampling strategies, improvement mechanism, space subdivision, bounding

Challenges and critique:

- simpler theory and implementation, less information, harder problem
- noise in data
- expensive function evaluations/sampling
- handling data-dependent constraints
- state-of-the-art methods can handle $\sim$100 variables
- performance/accuracy weaker than derivative-based optimization

7.3.2 Convergence mechanisms for direct search

Local direct search:

- Goal: to converge to stationary points
- Ingredients:
  - mechanism to impose descent direction away from stationarity
  - good control of geometry of sample sets
  - derive the step size parameter to zero for convergence

Global direct search:

- Goal: to approach the global optimum as close as possible
- DIRECT algorithm: sample, branch, and iterate towards better objective values

7.3.3 Convergence mechanisms for model-based methods

Local model-based methods:

- Goal: to converge to stationary points or local optimum
- Ingredients:
  - mechanism to impose descent direction away from stationarity
  - good control of geometry of sample sets
  - derive the step size parameter to zero for convergence
- Process:
  1. start with initial pattern of points in small region of space (trust region)
  2. build response surface model for trust region, and optimize
  3. move the trust region based on the optimal solution, till trust region is small enough

Global model-based methods:

- Goal: to approach the global optimum as close as possible
- Process:
  1. sample in entire space
  2. build response surface model (surrogate)
  3. validate solution by resampling
  4. update response surface, re-optimization till convergence

Convergence may be "easier" in DDO for noisy problems because the derivative information can be misleading can cause many local optimums.

7.4 Direct search methods

Overview:

- basic idea: only use data to guide search
- pros:
  - no extra cost of fitting models
  - simple implementations
  - easy to conceptualize
- cons:
  - require samples at very specific locations
  - may require too much data
  - no equations or derivatives

Local direct search in 1D:

1. select starting point, sample, obtain $f(x_0)$
2. pick step size η
3. move, sample, obtain $f(x_i)$
4. Based on values of $f(x_i)$ and $f(x_{i-1})$, change the step size
   - if $f(x_i) < f(x_{i-1})$, increase η
   - if $f(x_i) > f(x_{i-1})$, decrease η
5. Return to step 3 until converge

Compass search:

- first direct search algorithm in multi dimensions

- move sampling points up, down, left and right to determine moving direction

Global direct search:

- start with a set of samples everywhere in space
- DIRECT algorithm
- evolutionary/stationary points: genetics algorithm, partical swarm optimization, etc.

7.5 Model-based methods

Overview:

- models used in these methods are called approximations, surrogate models, metamodels, regression models, or machine learning models

Example 8. To minimize the problem without knowing f formulation, but knowing f is convex:

$$\begin{aligned} \min\ & f(x) = 2x^2 + \frac{16}{x} \\ \text{s.t. } & 1 \leq x \leq 5. \end{aligned}$$

We can approximate the original problem (formulation unknown, convex) with a quadratic function

1. collect at least 3 points
2. build approximation from sampling

$$q(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2)$$

3. optimization the approximation function

The optimum is obtained with 1.4% relative error

Challenges: the quality of DDO algorithms depends on

1. fit quality
2. if the surrogate is convex
3. how much data can we afford to collect to fit model (more dimensions require more data)

Dealing with bad fit:

- Sample more and refit
- use different function types
- local search model-based algorithm (trust region methods): break the search space into pieces, and fit each one with simple function (quadratic/linear)

If more data and more parameters always better? Depends on:

- the data location
- which type of function is used

Theorem 1 (Weierstrass theorem)**.** *If a function is continuous in an interval, then it can be perfectly*

approximated by a polynomial of high enough order. The higher the order of the polynomial, the better the approximation.

Runge phenomena: equidistant points and a polynomial approximation can diverge from true function.

7.6 Constrained DDO

Constraint types:

- known/glass-box: equation available
- unknown/black-box: need to sample

Current commercial DDO solvers can handle box-constrained problems:

$$\begin{aligned} &\min\ f(\mathbf{x}) \\ &\text{s.t. } \mathbf{x}^{\mathrm{L}} \leq \mathbf{x} \leq \mathbf{x}^{\mathrm{U}} \end{aligned}$$

7.6.1 Handling constraints

Consider the problem

$$\begin{aligned} &\min\ f(\mathbf{x}) \\ &\text{s.t. } \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \\ &\quad \mathbf{x}^{\mathrm{L}} \leq \mathbf{x} \leq \mathbf{x}^{\mathrm{U}} \end{aligned}$$

Penalty approach:

- reformulate the problem by adding a penalty for each constraint

$$\begin{aligned} &\min\ f(\mathbf{x}) + \sum_{i=1}^{K} \lambda_i \cdot \max\left(g_i(\mathbf{x}), 0\right) \\ &\mathbf{x}^{\mathrm{L}} \leq \mathbf{x} \leq \mathbf{x}^{\mathrm{U}} \end{aligned}$$

- pro: transform problem to box-constrained
- cons: hard to find balanced weights, change the objective function, optimum would be affected

Approximations per constraint:

- Consider the problem with some black-box constraints ($\mathbf{g}^{\mathrm{bb}}$) and some glass-box constraints ($\mathbf{h}^{\mathrm{gb}}$); objective function f is also black-box
- approximate both f and g with surrogate models

$$\begin{aligned} &\min\ f^{\mathrm{surr}}(\mathbf{x}) \\ &\text{s.t. } \mathbf{g}^{\mathrm{surr}}(\mathbf{x}) \leq \mathbf{0} \\ &\quad \mathbf{h}^{\mathrm{gb}}(\mathbf{x}) \leq \mathbf{0} \\ &\quad \mathbf{x}^{\mathrm{L}} \leq \mathbf{x} \leq \mathbf{x}^{\mathrm{U}} \end{aligned}$$

- pro: can use derivative-based solvers
- cons: cost of fitting many surrogates; small errors can lead to over/under- estimation of feasible region

Approximation of feasibility:

- group the constraint violation in one quantity

$$\begin{aligned} \min\ & f^{\text{surr}}(\mathbf{x}) \\ \text{s.t. } & g^{\text{surr}}(\mathbf{x}) \leq 0 \\ & \mathbf{h}^{\text{gb}}(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{x}^{\text{L}} \leq \mathbf{x} \leq \mathbf{x}^{\text{U}} \end{aligned}$$

- pros: can use derivative-based solvers; less function fitting
- cons: small errors can lead to over/under- estimation of feasible region; may be difficulty to approximate the feasible region with one function

Direct search:

- also check the constraints when sampling
- same three components: geometry of sample sets, mechanism for descent and feasibility, step size to 0

8 Week 7: Sampling-based DDO

8.1 Local direct search

When to use:

- when equations are not available, but "designed" experiments can be performed at specific locations
- when sampling is cheap, e.g., when simulations takes seconds or minutes
- When convergence to local stationary point is important
- When high-performance computing is available

When not to use:

- large-scale, nonlinear problems with discrete variables
- when equations (closed forms) and derivatives are available

Categories:

- `Direct-Search`: based on positive spanning sets
- `Simplex-based`: based on simplex[23]

Ingredients to converge to stationary points:

- mechanism to impose descent direction away from stationarity
  - `Direct-Search`: by moving towards best function value of positive basis or spanning sets

[23]Notice the difference between simplex (a geometric object) and the simplex method for LP (an algorithm).

  - `Simplex-based`: by moving away from from worst point
- good control of geometry of sample sets
  - `Direct-Search`: cosine measure of positive spanning sets
  - `Simplex-based`: control simplex volume
- derive the step size parameter to zero for convergence

`Nelder-mead`: default local direct search algorithm in MATLAB optimization, `simplex-based`

- core idea: search on a simplex (point, line, triangle, tetrahedron... in any dimension)
- has a set of rules to contract, expand, and shrink the simplex

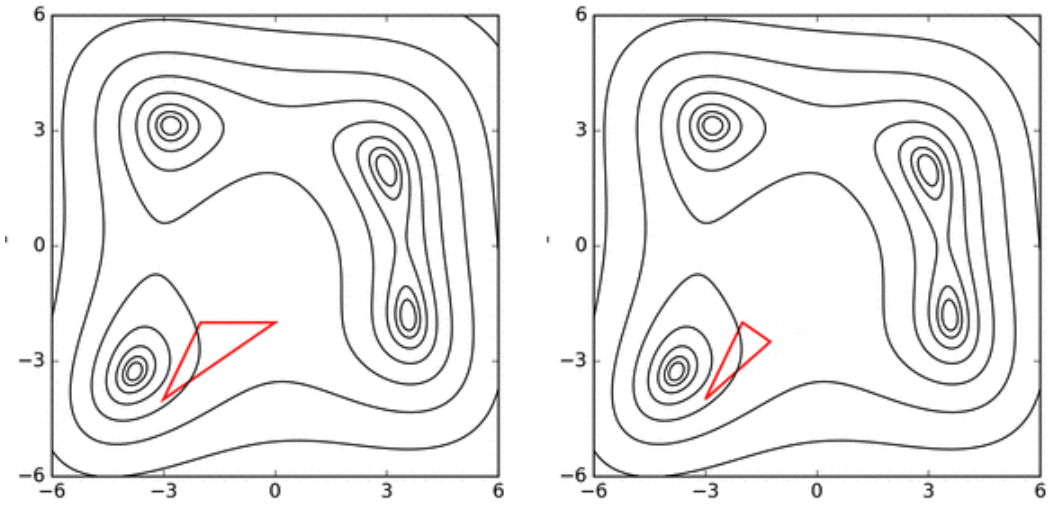


Figure 5: Two steps in Nelder-Mead search over a 2D function

`Direct-search` (pattern-search) algorithms:

- core idea: search over positive spanning basis sets (different sampling design)
- has a set of rules to contract, expand, and shrink sampling space
- may require more samples, but have more consistent and convergent behavior

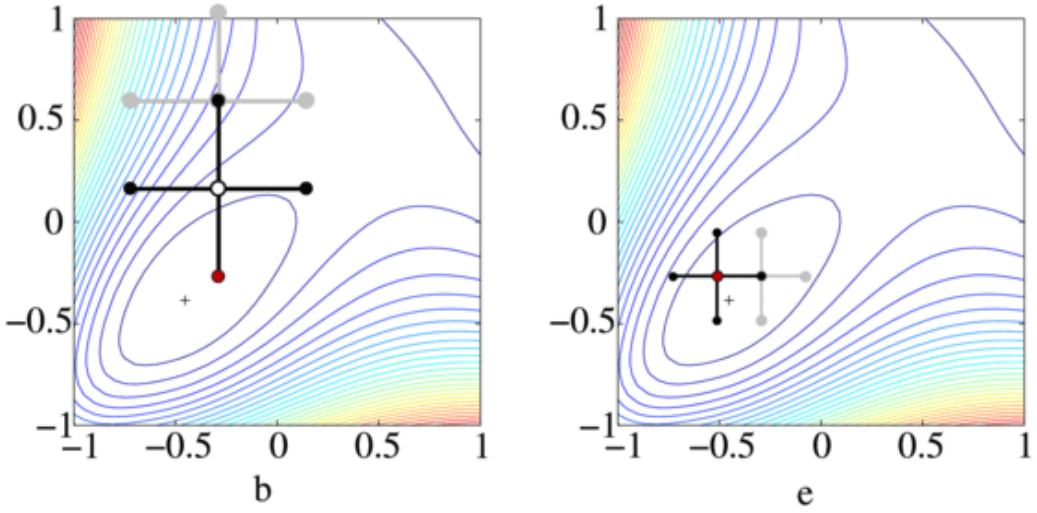


Figure 6: Two steps in pattern search over a 2D function

8.2 Positive spanning sets

Definition 6 ((linear) span, positive span)**.** For a set of vectors $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_r) \in \mathbb{R}^n$, the *(linear) span* of these vectors is the smallest linear subspace that contains them.

The *positive span* of these vectors is the their convex cone

$$\{\mathbf{v} \in \mathbb{R}^n : \mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_r\mathbf{v}_r, a_i \geq 0 \,\forall\, i \in \{1, \ldots, r\}\}\,.$$

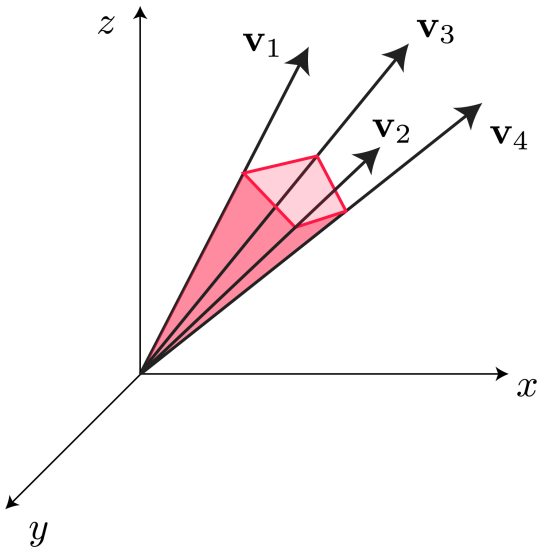


Figure 7: Positive span

Definition 7 (positive spanning set, positive basis)**.** A *positive spanning set* in $\mathbb{R}^n$ is a set of vectors whose positive span is $\mathbb{R}^n$.

A positive spanning set whose vectors are (conically) independent[24] is a *positive basis*. A positive basis contains at least $n+1$ (minimal) and at most $2n$ (maximal) vectors.

Example 9. In $\mathbb{R}^2$,

- $((0,1),(0,-1),(1,0),(-1,0))$ is a maximal positive basis
- $\Big((0,1),(1,0),(-2\sqrt{2},-2\sqrt{2})\Big)$ is a minimal positive basis
- $\Big((0,1),(1,0),(-2\sqrt{2},-2\sqrt{2}),(0,-1)\Big)$ is a positive spanning set, but is not a positive basis: $(0,-1)$ can be represented as a positive linear combination of other vectors:

$$(0,-1) = 1 \cdot (1,0) + \frac{1}{2\sqrt{2}})(-2\sqrt{2},-2\sqrt{2}).$$

8.2.1 Geometry for descent direction

Proposition 1. *Given any nonzero vector* $\mathbf{v} \in \mathbb{R}^n$*, there is at least one vector* $\mathbf{d}$ *in the positive spanning set such that* $\mathbf{v}$ *and* $\mathbf{d}$ *form an acute angle.*

Definition 8 (descent direction)**.** A *descent direction* for function f at $\mathbf{x}$ is a vector $\mathbf{d}$ such that there exists a constant $\alpha > 0$ s.t.

$$f(\mathbf{x} + \alpha\mathbf{d}) < f(\mathbf{x}).$$

[24] any vector cannot be represented as a positive, linear combination of other vectors

Assume f is continuously differentiable, and $\mathbf{v} \equiv -\nabla f(\mathbf{x})$. Then any vector $\mathbf{d}$ that forms acute angle with $\mathbf{v}$ is a descent direction.

8.2.2 Geometry for sampling design

When designing sampling for expensive functions:

- we should try to guarantee to improve the objective function at each step, but not to sample more than necessary
- If the sample vectors form positive basis, then it is guaranteed that a descent direction is found
- Need no more than $n+1$ (minimal) vectors at each step

Assume we sample around point $\mathbf{x}$ with step size α on $n+1$ points: $\mathbf{x}_i = \mathbf{x} + \alpha \mathbf{d}_i, i \in \{1, \ldots, n+1\}$, and $\mathbf{d}_i$'s form a positive basis.

- As long as $\nabla f(\mathbf{x}) \neq 0$, $\exists\, \alpha > 0$ s.t. $f(\mathbf{x}_i) = f(\mathbf{x} + \alpha \mathbf{d}_i) < f(\mathbf{x})$ for some i
- If all new samples are no better than $f(\mathbf{x})$, then $\nabla f(\mathbf{x})$ is at the same order of the magnitude with α
- To keep on improving the objective value, we need to repeat sampling and moving while decreasing α
- The value of effective α (step size that can help find better samples in the next step) is a function of the nonlinearity of f and geometry of the sample set
- As $\alpha \to 0$, the gradient also approaches 0, which leads to stationary points.
- So theoretically the searching will termination after finite steps (this is a systematic search)

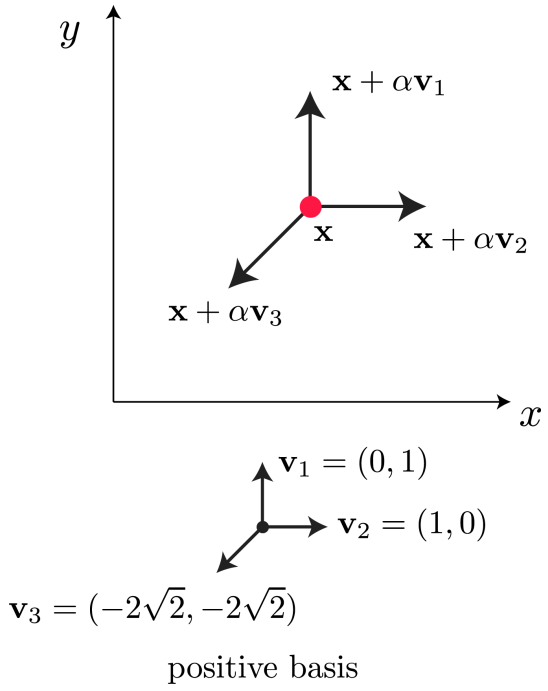


Figure 8: Direct search in 2D: at least one of $\mathbf{x} + \alpha \mathbf{v}_i$ is descent direction

8.3 Convergence in sampling-based optimization

Definition 9 (Lipschitz continuity, Lipschitz constant)**.** A function $f : E \subset \mathbb{R}^n \to \mathbb{R}$ is *Lipschitz* if there exists $v > 0$ with

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq v \cdot |\mathbf{x} - \mathbf{y}|$$

for any $\mathbf{x}, \mathbf{y}$ in E. v is called *Lipschitz constant*.

- It is related to how fast a function can change
- if f is differentiable and Lipschitz, then its derivatives are bounded, and v is the upper bound of absolute values of derivatives

Example 10. Show $f(x) = x^3$ is Lipschitz on $[1, 2]$, and calculate a Lipschitz constant.

For all$x_1, x_2 \in [1, 2]$,

$$\begin{aligned} \left|x_1^3 - x_2^3\right| &= \left|(x_1 - x_2)\left(x_1^2 + x_1 x_2 + x_2^2\right)\right| \\ &\leq |x_1 - x_2| \cdot \left|x_1^2 + x_1 x_2 + x_2^2\right| \end{aligned}$$

As $x_1, x_2 \in [1, 2]$, $|x_1^2 + x_1 x_2 + x_2^2| \leq 12$. Therefore,

$$|g(x_1) - g(x_2)| \leq 12|x_1 - x_2|.$$

Example 11. $f(x) = \sqrt{x}, x \in [0, 1]$ is not Lipschitz at 0: the derivative of f around 0 is

$$f'(x) = \frac{1}{2\sqrt{x}} \to 0 \text{ when } x \to 0.$$

Definition 10 (cosine measure)**.** The *cosine measure* of a positive spanning set D is defined by

$$\mathrm{cm}(D) = \min_{\mathbf{v} \neq \mathbf{0}} \max_{\mathbf{d} \in D} \frac{\mathbf{v}^{\mathrm{T}} \mathbf{d}}{\|\mathbf{v}\| \, \|\mathbf{d}\|}.$$

When $n = 2$, and with the positive basis $D = \begin{bmatrix} 1 & 0 & -\cos\theta \\ 0 & 1 & -\sin\theta \end{bmatrix}$ with $\theta \in (0, \frac{\pi}{4}], \mathrm{cm}(D) = \cos\left(\frac{\pi - \theta}{2}\right)$.

Cosine measure can be used to quantify the "quality" of the geometry for a certain positive spanning set: the bigger the better.

8.3.1 Gradient estimation

With

- Lipschitz constant v
- Positive spanning set D
- cosine measure $\mathrm{cm}(D)$
- canter point $\mathbf{x}$, step size α
- radius of ball around $\mathbf{x}$: $\Delta \equiv \alpha \max \|\mathbf{d}\|$

The gradient is bounded:

$$\|\nabla f(\mathbf{x})\| \leq \frac{v}{2 \cdot \mathrm{cm}(D)}\Delta = \kappa_{\mathrm{eg}}\Delta, \quad \kappa_{\mathrm{eg}} \equiv \frac{v}{2 \cdot \mathrm{cm}(D)}.$$

8.3.2 Pattern search algorithm

Simplified description[25]:

In each iteration,

1. [optional search step] evaluate f at finite random points for potential improvements
2. [pull step] evaluate f at $\mathbf{x}_k + \alpha_k \mathbf{d} \, \forall \, \mathbf{d} \in D$. If there is a better solution at $\mathbf{x}_k + \alpha_k \mathbf{d}_k$, the iteration succeeds, set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$; otherwise, $\mathbf{x}_{k+1} = \mathbf{x}_k$
3. [mesh (step size) parameter update] If the iteration succeeds, then increase step size: $\alpha_{k+1} \in [1, \gamma] \cdot \alpha_k$; otherwise, decrease step size: $\alpha_{k+1} \in [\beta_1, \beta_2] \cdot \alpha_k$

8.4 Global sampling methods and DIRECT algorithm

Difference of an approximate global optimum and an exact local optimum:

- The former may have better objective values
- The latter must have zero derivatives
- Need to pick different types of optimums based on the application

Overview:

- Global sampling methods collect samples everywhere in the space, NOT around a small region
- "Global" does not imply that they always find global optimum, but they attempt to by searching broadly
- Categories:
  - determinisitic global search: DIRECT algorithm
  - stochastic global search: stochastic & evolutionary methods

Properties:

- More explorative than local sampling-search (local sampling-search can achieve this by initializing at multiple points)
- No derivative information, no guarantee to stationary; But could get closer to global optimum
- require many samples for exact solution; suitable for problems with "cheap" sampling
- convergence methods:
  - guarantee to diversify samples in the process, so that global optimums will be find with infinite sampling
  - its efficiency greatly depends on sampling heuristics

[25]For full description, see Chapter 7 in [1]

8.4.1 DIRECT algorithm

Basic idea[26]:

- Treat the search space as a hypercube
- Sample each hypercube at its center
- divide each hypercube in all dimensions
- choose the hypercube with better objective values, and repeat

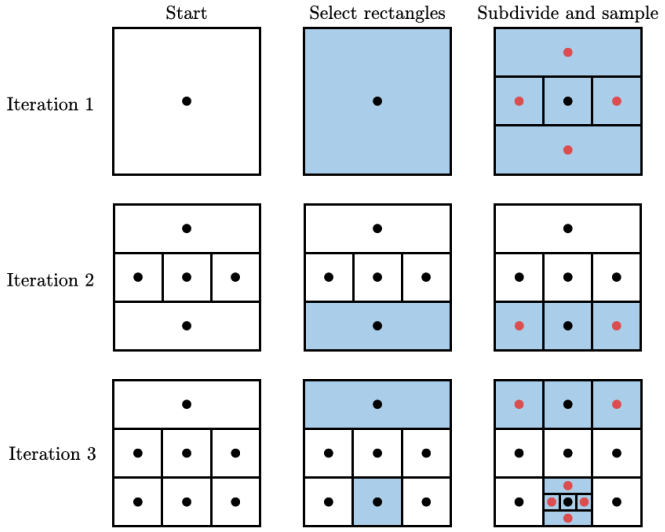


Figure 9: schematic for DIRECT algorithm

Connection Lipschitz constant:

- Consider a center point $\mathbf{c}_j$ for the jth hypercube. Other samples around it can be represented as the pair of (1) its distance with $\mathbf{c}_j$, d; (2) the objective value there, f', as a function of d.

[26]Full description: [2]

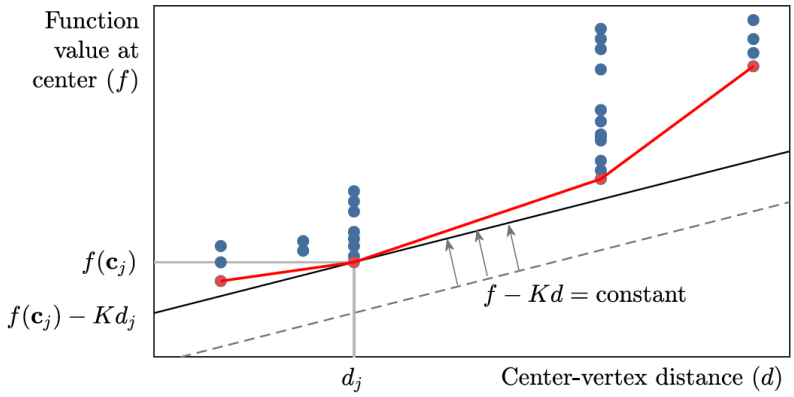


Figure 10: Approximating Lipschitz constant

- from the definition of Lipschitz constant (we denote it using K here), if other center points for other hypercubes are represented as $\mathbf{c}_j + \mathbf{d}_i$, and $\|\mathbf{d}_i\| = d$:

$$\|f(\mathbf{c}_j) - f(\mathbf{c}_j + \mathbf{d}_i)\| \leq K \cdot \|\mathbf{c}_j - (\mathbf{c}_j + \mathbf{d}_i)\| \Longrightarrow f(\mathbf{c}_j + \mathbf{d}_i) \geq f(\mathbf{c}_j) - K \cdot d$$

- Sampling can be seen as a process of approximating Lipschitz constant

Advantages:

- most popular deterministic global sampling method
- simple concept connected to Lipschitz theory
- work well with problems with up to 10 variables
- approaches the global optimum fast, but need many samples to get exact solution

8.5 Sampling-based stochastic and evolutionary methods

Overview:

- can be used for problems with discrete variables
- many of them inspired by nature

The meaning of "stochastic":

- stochastic methods have randomness in search criteria
- populations of samples generated by random perturbations
- the same method re-initialized will result in different samples
- When run infinitely, they will converge to the same solution; with limited times, they solutions may vary
- tend to require many samples
- different from "stochastic optimization"

Heuristics:

- stochastic sampling-based methods purely based on heuristics
- they are rules of thumb that are considered to work in practice , but with no theoretical guarantees

8.5.1 Simulated annealing

Basic idea:

- Inspired by metallurgic technique
- to melt and re-crystallize metal to alter they properties
- efficiency affected by heating/cooling rates

Algorithm:

1. start with a guestimate $f(x_0)$, and a temperature T
2. Evaluate a neighbor $f(x)$
    - If $f(x) < f(x_0)$ (for minimization problem), move to x
    - Else, based on a probability calculation w.r.t. T and $f(x)$, move to x to accept a worse solution, or stay at x_0
3. Update T based on cooling schedule, and repeat

Pros:

- Helps avoid local optimums, better chance to find global optimum
- many online implementation

Cons:

- no theoretical guarantee for global optimum
- performance depends on heuristic parameters
- solutions may not be local optimum

8.5.2 Genetic algorithms

Basic idea:

- Mimic the evolution process in nature
- start with a population of guestimates, each of them is a chromosome (an individual) with various genes (characteristics)
- use heuristics to improve population using natural selection ideas
    - inheritance
    - mutation
    - crossover
    - selection

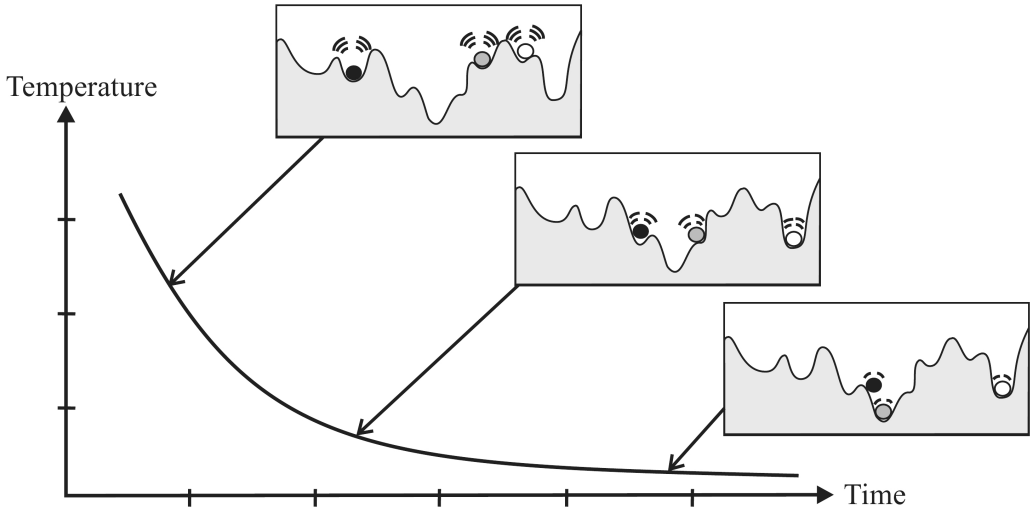


Figure 11: Schematics for simulated annealing process

Algorithm:

1. generate an initial population
2. small numbers of individuals are allowed to mate randomly or based on their solution quality
3. the offspring becomes a new generation, and best individuals of the previous generation are retained
4. evaluate the new generation, and repeat

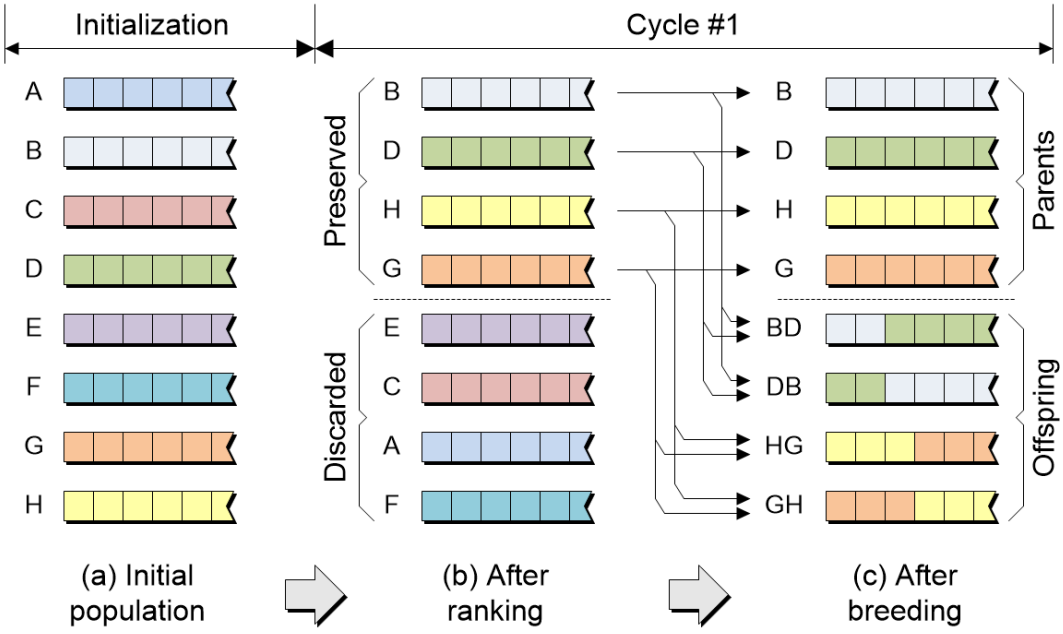


Figure 12: Schematics for genetic algorithms

Pros:

- many implementations
- possible to find the global optimum
- any cost function may be used
- widely used especially in bio-related fields

Cons:

- challenging to find the best setting
- process slow, may not converge
- rarely get exact solutions

8.5.3 Particle swarm optimization

Basic idea:

- originally developed to simulate social behavior, mimicking how fish or birds behave

Algorithm:

1. Initialize population of candidate solutions (particles)
2. Move particles in space using certain formulae, based on their best known position and the relative position of "swarm"

Pros:

- Many implementations available
- Sample-based, may find the global optimum more accurately (many particles in the same area)

Cons:

- requires many samples
- no guarantee for global optimum
- many particles in the same area, making find the exact global optimum more costly

8.5.4 Discussion of stochastic methods

Good for:

- when approximate global solution is good enough: do not require optimality guarantee allow stochastic nature (re-initialization causes different solutions)
- problems with discontinuities, noise

Not good for:

- when sampling is expansive
- when equations exist

9 Week 8: Model-based DDO

9.1 Surrogate modeling

Definition 11 (surrogate model)**.** A *surrogate model* is a fitted model that approximates the relationship between the inputs (variables) and the outputs (objective function, and potentially constraints), and is trained with data from ODE/PDEs, simulation models, historical data or designed lab data.

Difficulty of simulation-based optimization:

- Growing need to optimization problem with embedded simulations from ASPEN, gPROMS, COMOSL, etc.
- These problems are usually highly nonlinear, or contains large-scale systems of PDEs, or contains external proprietary models
- motivates surrogate modeling

Types of surrogate models (ranking with increasing complexity: nonlinearity, number of terms and parameters increase):

- linear regression
- polynomial regression
- support vector regression
- regression trees
- artificial neural networks

Surrogate-based optimization steps:

1. generate data via simulation or historical data
2. (optional) use design of experiment to perturb and collect extra data
3. use machine learning to fit data into an optimization model
4. solve the optimization problem

Challenges:

- find the best surrogate model with few samples/limited data
- different samples or surrogate models lead to different optima
- noise brings extra uncertainty

Adaptive surrogate-based optimization:

- basic idea: after solving the optimization problem, validate the global optimum via simulation/extra data; If the error is large, then re-fit the surrogate model with the extra data point
- sampling $\rightarrow$ surrogate model training $\rightarrow$ optimization $\rightarrow$ solution validation $\rightarrow$ sampling ...

9.2 Regression vs. interpolation

- A surrogate model is a *interpolation* model if it passes through all data points
  - useful when data is exact, e.g., from computer simulation
  - useful when the number of data is the same as the number of parameters

- A surrogate model is a *regression* model if it minimizes prediction-data error without guaranteeing to exactly predict data
  - useful when data is noisy

Parametric vs. non-parametric models:

- Both models have parameters that need to be trained
- *parametric models*
  - Parameters have certain meaning/interpretations
  - e.g., linear, quadratic, polynomial, generalized linear models
  - usually have fewer parameters than data → regression
- *non-parametric models*
  - Parameters are associated with complex terms with no physical meaning
  - usually use kernel transformation
  - e.g., Gaussian process, neural networks, support vector regression
  - usually have as many parameters as data → interpolation
  - some non-parametric models are regressors or can be both (e.g. neural network)

Categories of surrogate models (from regression to interpolation, with exceptions):

- linear regression
- polynomial regression
- support vector regression
- regression trees
- artificial neural networks

Implications for optimization:

- number of nonlinear terms affects optimization complexity (especially with many constraints)
- types of nonlinear terms (convex/nonconvex) affects optimization speed/quality, which affects the quality of found optimum

9.3 Sampling

Key factors for sampling for surrogate modeling:

- how many data points
- where to sample
- impact of surrogate model type

9.3.1 Design of experiments

Design of experiments (DOE):

- techniques that place sampling points to study the effects of various inputs on the outputs
- linked to how the data is used to "fit" a simplified statistical model that "maps" the inputs to the outputs
- *Experimental DOE*: need to consider stochastic effects on the noise in data from physical experi-

ments, replications
- *Computational DOE*: data from deterministic simulation
- important to know the source of data in surrogate modeling: "designed" data (course focus) or historical data

Independent vs. dependent inputs:

- Independent/controllable inputs: if we know and have them, then we can directly design experiments
- High-dimensional, potentially "correlated"/"dependent" data: typically the case for historical data we have no control over; need further analysis: identify correlations, remove redundant inputs, transform input space, identify data density and uniformity

Deterministic vs. stochastic data-generation system:

- deterministic: fixed input leads to the same output every time
- stochastic: fixed input leads to slight different output every time; replication of data may be important

9.3.2 DOE with known degrees of freedom

Factorial/uniform grid designs (usually for experimental DOE):

- aims:
  - screen effects
  - can capture linear (full factorial designs), nonlinear (fractional factorial designs) or cross-correlation effects
  - minimize sampling number
  - use collect data to perform statistical tests
  - repetitions in the same location can be used to quantify level of noise
- types: full factorial ∼, fractional factorial ∼, composite ∼, box Behnkin ∼, Plackett-Burman ∼
- scalability: n dimensions, m levels (points in each dimension): $m^n \rightarrow$ explode. *curse of dimensionality*

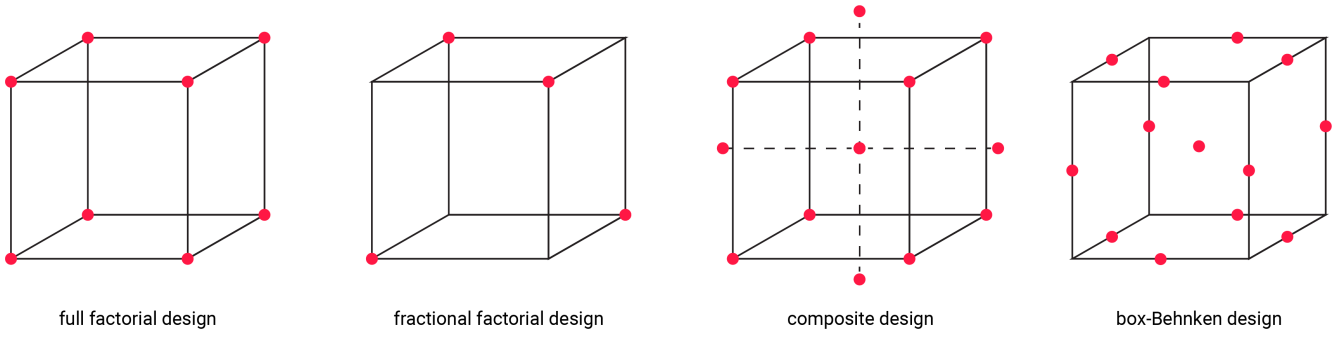


Figure 13: Factorial/uniform grid designs

Computer DOE:

- data from computer simulation
- goals:
  - collect representative set of samples from entire region
  - minimize sampling requirements
  - allow for nonlinear interactions to be captured
  - when simulation is deterministic, no repetitions are needed

Space filling design: Latin hypercube sampling (LHS)

- most popular technique with computer DOE
- basic idea: divide the range of n variables into m equally probable intervals, then place m sample points such that there is only one sample in each interval for each variable

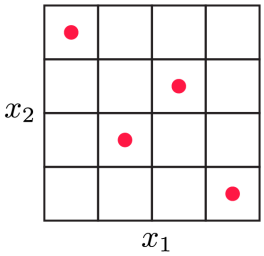


Figure 14: Latin hypercube sampling for 2 variables, 4 samples (levels)

- stochastic sampling method; reduce effect of curse of dimensionality
- criterion: several exist to guide the generation of sampling locations; "max-min" rule: maximize the minimal distance between two samples (avoid clustering)

Other space filling design:

- Sobol sequences
- Orthogonal-array LHS
- Sphere packing

9.4 Relationship between sampling and surrogate model

Example 12. We want to study the relationship between one output y and two inputs x_1, x_2.

- When we only collect 4 samples, we have enough data to screen effect, but we cannot fit a predictive quadratic model
- When we collect 9 samples, we can fit a quadratic model, and have enough extra data to validate the model

Cross-validation:

- techniques to assess if the generated model will generalize to an independent data set
- Ideally, we should have enough data to split into
  1. training set: data used to fit parameters
  2. validation set: left-out data to calculate the error of prediction of the trained trained data

3. (optional) test set: separate data set not used in training and validation, for final error calculation after cross-validation

- *k-fold cross-validation*: In k iterations, randomly select $1 - \frac{1}{k}$ of data as training set and $\frac{1}{k}$ of data as validation set. Check if the error of k models are consistent to see if the model is in general predictive.

9.5 Linear models and regularization

9.5.1 Linear regression model

- Assumptions: all inputs (variables) are linearly correlated with output (objective function):

$$y = a_0 + \sum_{i=1}^{n} a_i x_i$$

- number of parameters: $n + 1$
- common objectives: sum of squared error, sum of absolute error

9.5.2 Generalized linear regression models

- Assumptions: the input *features* are linearly correlated with output:

$$y = b_0 + \sum_{j=1}^{m} X_j,$$

  where X_j can be functions of x_i: $x_1, x_2, x_1 x_2, x_1^2, e^{x_1}, \ldots$
- It is linear in terms of the parameters to be fitted $b_0, b_1, \ldots, b_m$

9.5.3 Regularization

Error vs. model complexity:

- When the number of parameters (features) increases, the training error is decreasing, as the model is more complex
- But the validation error decreases first then increases, as we are overfit the model after some point

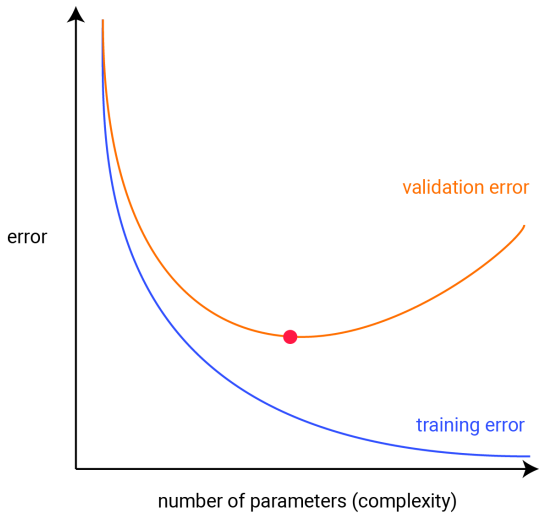


Figure 15: Error vs. model complexity

- Aim: find the minimal validation error → regularization

Definition 12 (regularization)**.** *Regularization* refers to penalizing the size of the surrogate model (or the number of parameters) while fitting the model.

When the model has fewer parameters, it is usually:

- more generalizable
- more interpretable
- potentially easier to optimize
- less prone to overfitting

ℓ_2 regularization (ridge regression):

- ℓ_2 norm of vector $\mathbf{n}$:

$$\|\mathbf{n}\|_2 = \sqrt{\sum_{i=1}^{k} n_i^2}$$

- basic idea: penalize the size of parameters
- fitting model:

$$\min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \sum_{j=0}^{m} b_j^2$$

$$\text{s.t. } \hat{y}_i = \sum_{j=0}^{m} b_j X_j, i = 1, 2, \ldots, n$$

  where
  - y_i: actual output value
  - $\hat{y}_i$: predicted output value
  - X_j: input features

  - b_j: parameters to be fitted
  - $\sum_{j=0}^{m} b_j^2$ is the square of the ℓ_2 norm of the parameter vector $\mathbf{b} = (b_0, b_1, \ldots, b_m)$
- pro: formulation is convex
- con: does not guarantee feature selection (some parameters nearly zero)

ℓ_1 regularization (LASSO):

- ℓ_1 norm of vector $\mathbf{n}$:

$$\|\mathbf{n}\|_1 = \sum_{i=1}^{k} |n_i|$$

- basic idea: penalize the sum of absolute values of parameters
- fitting model:

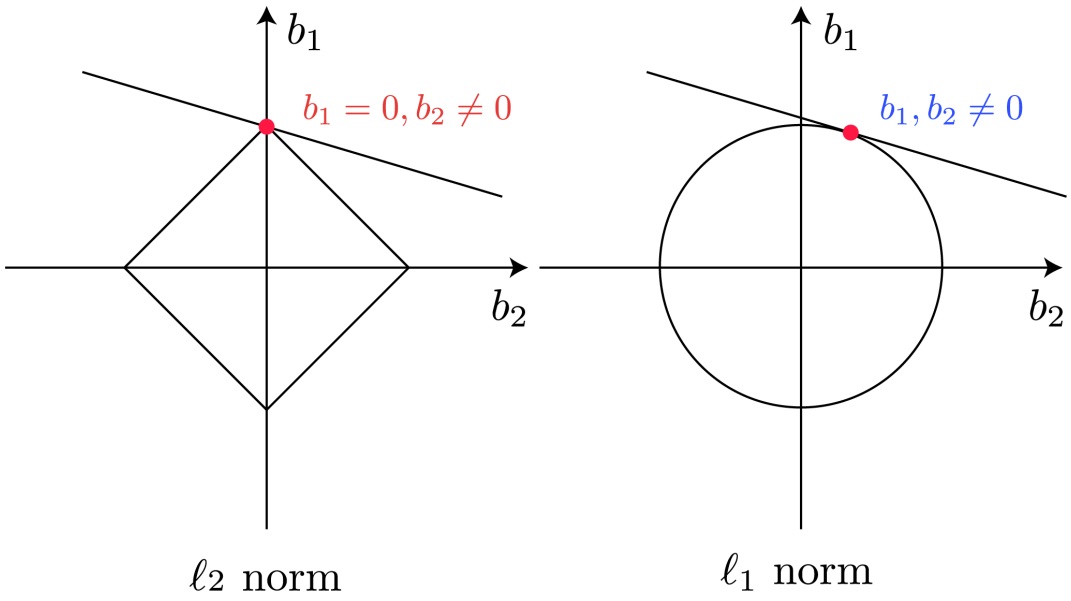
$$\begin{aligned} \min \ & \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ \text{s.t.} \ & \sum_{j=0}^{m} |b_j| \leq t \\ & \hat{y}_i = \sum_{j=0}^{m} b_j X_j, i = 1, 2, \ldots, n \end{aligned}$$

  where
  - t: user-defined parameter to determine degree of regularization
- We can use Lagrange function to transform the constraint $\sum_{j=0}^{m} |b_j| \leq t$ into the objective function

$$\begin{aligned} \min \ & \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=0}^{m} |b_j| \\ \text{s.t.} \ & \hat{y}_i = \sum_{j=0}^{m} b_j X_j, i = 1, 2, \ldots, n \end{aligned}$$

  where
  - λ: Lagrange multiplier
  - t: removed from the objective as it is a constant, not a variable
- pro: built-in feature selection
- con: problem is nonconvex

Visualization of ℓ_1, ℓ_2 regularization: why ℓ_1 usually leads to fewer nonzero parameters



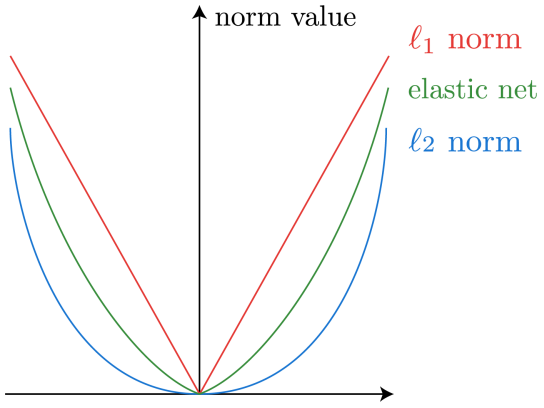
Elastic-net (EN) regularization

- basic idea: have both ℓ_1, ℓ_2 norms in objective function
- fitting model:

$$\min \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=0}^{m} b_j^2 + (1-\lambda)\sum_{j=0}^{m}|b_j|$$

$$\text{s.t. } \hat{y}_i = \sum_{j=0}^{m} b_j X_j, i = 1, 2, \ldots, n$$

Comparison of 3 types of regularizations:



Limitations of feature selection using these methods: they all assume the features are linearly independent, so may not work well if they are dependent with each other.

References

[1] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*. SIAM, 2009.

[2] D. R. Jones and J. R. Martins. The direct algorithm: 25 years later. *Journal of Global Optimization*, pages 1–46, 2020.