

香 港 大 學
THE UNIVERSITY OF HONG KONG

Algorithms for Constrained Optimization

Prof. Li Jiayang

DASE 7113 Optimization Method for Intelligent Systems

Optimization Problems

Unconstrained Program

$$\begin{aligned} \text{minimize} \quad & f(x) \\ x \in \mathbb{R}^n & \end{aligned}$$

Constrained Program

$$\begin{aligned} \text{minimize} \quad & f(x) \\ x \in \mathbb{R}^n & \\ \text{subject to} \quad & g_i(x) \leq 0, \ i = 1, \dots, m \\ & h_i(x) = 0, \ i = 1, \dots, p \end{aligned}$$

Linear Program

$$\begin{aligned} \text{minimize} \quad & f(x) \quad \text{linear} \\ x \in \mathbb{R}^n & \\ \text{subject to} \quad & g_i(x) \leq 0, \ i = 1, \dots, m \quad \text{linear} \\ & h_i(x) = 0, \ i = 1, \dots, p \quad \text{linear} \end{aligned}$$

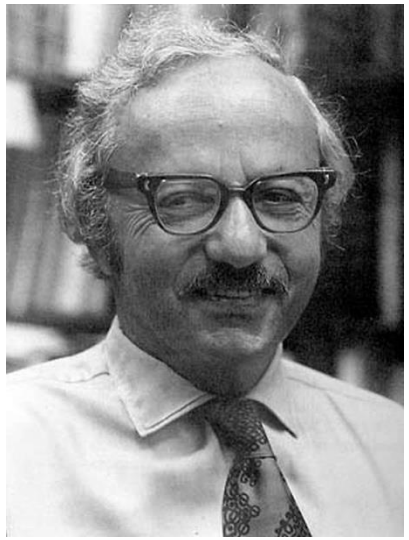
……

“Programming”

Why optimization problems are often called “programs?”



Program: a schedule of logistical supply and deployment of combat units





George B. Dantzig

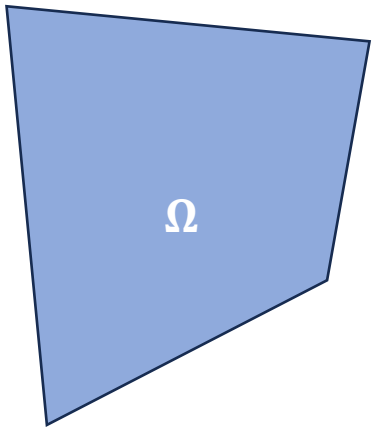
Linear Program

The feasible region to a linear program is a polyhedron

$$\begin{aligned} &\underset{x \in \mathbb{R}^n}{\text{minimize}} && f(x) && \rightarrow \text{linear} \\ &\text{subject to} && g_i(x) \leq 0, \ i = 1, \ldots, m && \rightarrow \text{linear} \\ & && h_i(x) = 0, \ i = 1, \ldots, p && \rightarrow \text{linear} \end{aligned}$$

$$\mathcal{G}_i = \{x : g_i(x) \leq 0\}$$

$$\mathcal{H}_i = \{x : h_i(x) = 0\}$$

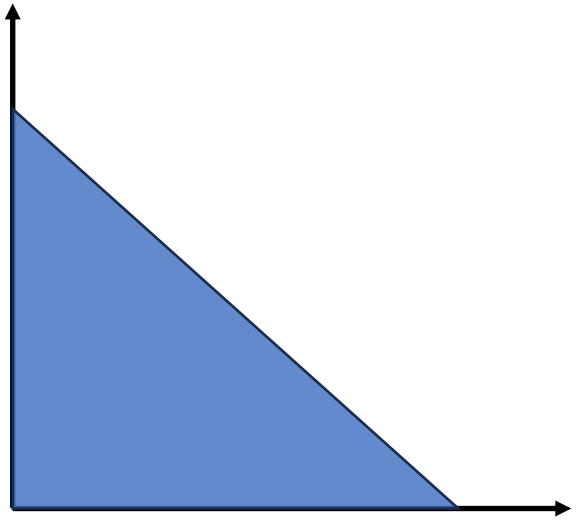


$$\Omega = (\cap_{i=1}^{m} \mathcal{G}_i) \cap (\cap_{i=1}^{p} \mathcal{H}_i)$$

Linear Program

A linear program either has an optimal solution, or is unbounded, or is infeasible.

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & -x_1 - 2x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq c \\ & x_1, x_2 \geq 0 \end{aligned}$$

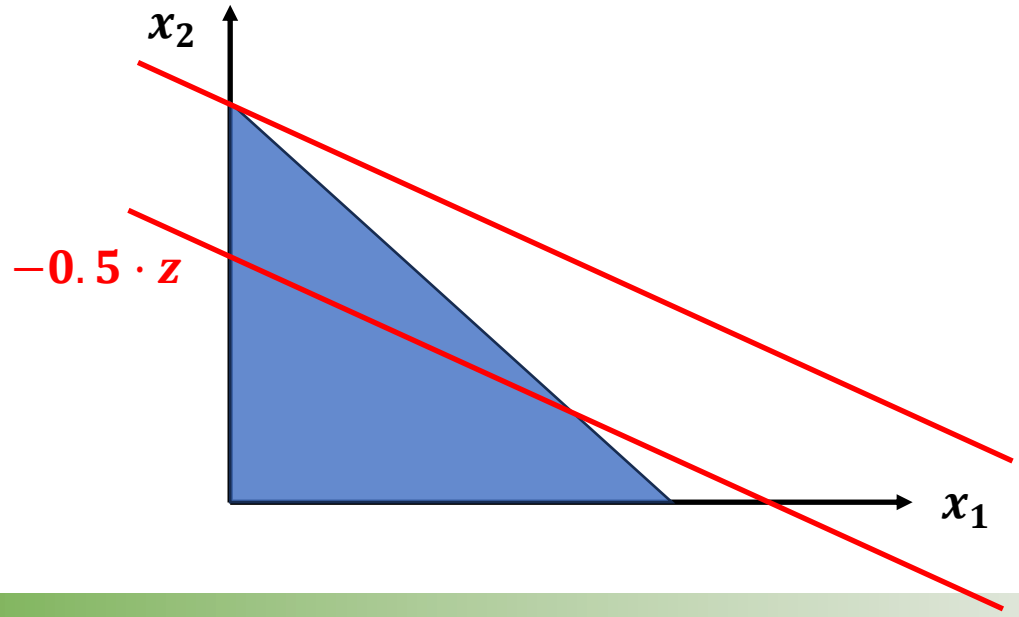


$c < 0$	$c \in [0, \infty)$	$c = \infty$
infeasible	has an optimal solution	unbounded

Linear Program

If a linear program has an optimal solution, then it must have a "vertex" optimal solution

$$\begin{aligned} \text{minimize} \quad & -x_1 - 2x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$



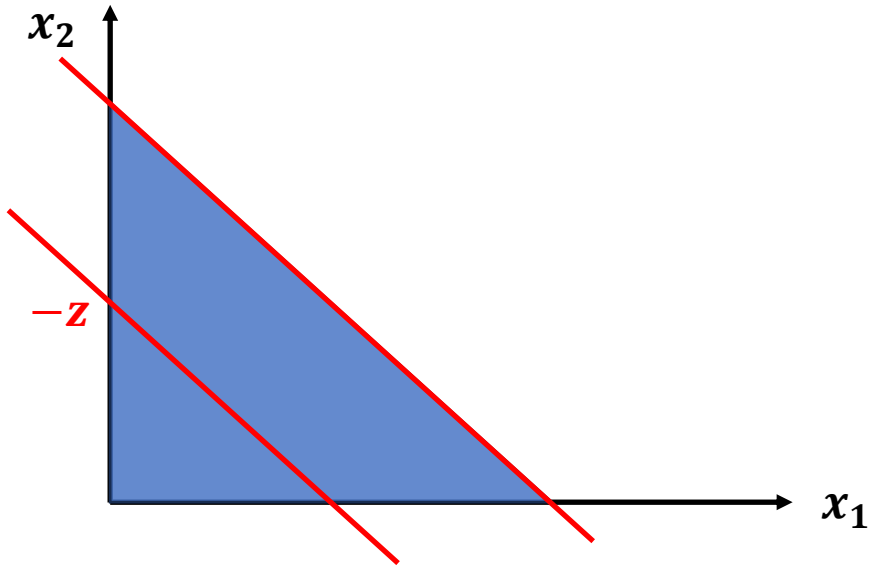
Assume $z = -x_1 - 2x_2$

$$x_2 = -0.5 \cdot x_1 - 0.5 \cdot z$$

Linear Program

If a linear program has an optimal solution, then it must have a "vertex" optimal solution

$$\begin{aligned} \text{minimize} \quad & -x_1 - x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$



Assume $z = -x_1 - x_2$

$$x_2 = -x_1 - z$$

Linear Program

George B. Dantzig developed the simplex method for solving linear programs in 1947

Chapter 1.	**Introduction**	**1**
1.1.	Optimization	1
1.2.	Types of Problems	2
1.3.	Size of Problems	5
1.4.	Iterative Algorithms and Convergence	6
PART I	**Linear Programming**	
Chapter 2.	**Basic Properties of Linear Programs**	**11**
2.1.	Introduction	11
2.2.	Examples of Linear Programming Problems	14
2.3.	Basic Solutions	19
2.4.	The Fundamental Theorem of Linear Programming	20
2.5.	Relations to Convexity	22
2.6.	Exercises	28
Chapter 3.	**The Simplex Method**	**33**
3.1.	Pivots	33
3.2.	Adjacent Extreme Points	38
3.3.	Determining a Minimum Feasible Solution	42
3.4.	Computational Procedure—Simplex Method	46
3.5.	Artificial Variables	50
3.6.	Matrix Form of the Simplex Method	54
3.7.	The Revised Simplex Method	56
*3.8.	The Simplex Method and LU Decomposition	59
3.9.	Decomposition	62
3.10.	Summary	70
3.11.	Exercises	70

Understanding every detail of the method is not easy…

Our focus on linear program today:

Just learn how to solve it using solvers

Linear Program

MATLAB

linprog

Solve linear programming problems

Syntax

```
x = linprog(f,A,b)
x = linprog(f,A,b,Aeq,beq)
x = linprog(f,A,b,Aeq,beq,lb,ub)
x = linprog(f,A,b,Aeq,beq,lb,ub,options)
x = linprog(problem)
[x,fval] = linprog(__)
[x,fval,exitflag,output] = linprog(__)
[x,fval,exitflag,output,lambda] = linprog(__)
```

Description

Linear programming solver

Finds the minimum of a problem specified by

$$\min_x f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

f, x, b, beq, lb, and ub are vectors, and A and Aeq are matrices.

SciPy

scipy.optimize.linprog

linprog(*c*, ***A_ub***=*None*, ***b_ub***=*None*, ***A_eq***=*None*, ***b_eq***=*None*, ***bounds***=*(0, None)*, ***method***='*highs*', ***callback***=*None*, ***options***=*None*, ***x0***=*None*, ***integrality***=*None*)

Linear programming: minimize a linear objective function subject to linear equality and inequality constraints. [source]

Linear programming solves problems of the following form:

$$\begin{aligned} \min_x \ & c^T x \\ \text{such that } & A_{ub} x \le b_{ub}, \\ & A_{eq} x = b_{eq}, \\ & l \le x \le u, \end{aligned}$$

where x is a vector of decision variables; c, b_{ub}, b_{eq}, l, and u are vectors; and A_{ub} and A_{eq} are matrices.

Linear Program

Python

scipy.optimize.

linprog

```
linprog(c, A_ub=None, b_ub=None, A_eq=None, b_eq=None, bounds=(0, None),
method='highs', callback=None, options=None, x0=None, integrality=None)
```

Linear programming: minimize a linear objective function subject to linear equality and inequality constraints. [source]

Linear programming solves problems of the following form:

$$\min_x \ c^T x$$
$$\text{such that } A_{ub} x \leq b_{ub},$$
$$A_{eq} x = b_{eq},$$
$$l \leq x \leq u,$$

where x is a vector of decision variables; c, b_{ub}, b_{eq}, l, and u are ve
matrices.

method : *str, optional*

The algorithm used to solve the standard form problem. The following are supported.

- 'highs' (default)
- 'highs-ds'
- 'highs-ipm'
- 'interior-point' (legacy)
- 'revised simplex' (legacy)
- 'simplex' (legacy)

The legacy methods are deprecated and will be removed in SciPy 1.11.0.

The simplex method is not a default method for most solvers because it can hardly attack large-scale linear programs

Linear Program

To use linprog, we must convert "greater than" constraints to "less than" constraints.

$$
\begin{aligned}
\underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & -x_1 + 4x_2 \\
\text{subject to} \quad & -3x_1 + x_2 \leq 6 \\
& -x_1 - 2x_2 \geq -4 \\
& x_2 \geq -3
\end{aligned}
$$

```
from scipy.optimize import linprog


c = [-1, 4]
A = [[-3, 1], [1, 2]]
b = [6, 4]
x0_bounds = (None, None)
x1_bounds = (-3, None)
res = linprog(c, A_ub=A, b_ub=b, bounds=[x0_bounds, x1_bounds])
print(res.x)
```

Linear Program

Question

A construction firm is building a house for its clients. The table presents the list of activities that the firm must complete for its current client's project. What is the earliest time that firm can finish the project?

Activity	Duration (days)	Predecessor
A	8	—
B	10	A
C	9	B
D	8	B, C
E	5	B
F	10	E
G	2	C, D, F
H	3	F
I	10	G, H

Finished Time
T_A
T_B
T_C
T_D
T_E
T_F
T_G
T_H
T_I

$$\text{minimize} \quad z$$

$$\text{subject to} \quad T_A - 8 \geq 0$$
$$T_B - 10 \geq T_A$$
$$T_C - 9 \geq T_B$$
$$T_D - 8 \geq T_B, T_C$$
$$T_E - 5 \geq T_B$$
$$T_F - 10 \geq T_E$$
$$T_G - 2 \geq T_C, T_D, T_F$$
$$T_H - 3 \geq T_F$$
$$T_I - 10 \geq T_G, T_H$$
$$z \geq T_A, \dots, T_I$$

➡ **Solve it via linprog**

Linear Program

```
import numpy as np
from scipy.optimize import linprog


c = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])  # [TA,TB,TC,TD,TE,TF,TG,TH,TI,z]

bounds = [
    (0, None),  # TA
    (0, None),  # TB
    (0, None),  # TC
    (0, None),  # TD
    (0, None),  # TE
    (0, None),  # TF
    (0, None),  # TG
    (0, None),  # TH
    (0, None),  # TI
    (None, None) # z (unbounded)
]
```

```
A_ub = np.array([
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [-1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, -1, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, -1, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, -1, 1, 0, 0, 0, 0, 0, 0],
    [0, -1, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, -1, 1, 0, 0, 0, 0],
    [0, 0, -1, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, -1, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, -1, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, -1, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, -1, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, -1, 1, 0],
    [-1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, -1, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, -1, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, -1, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, -1, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, -1, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, -1, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, -1, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, -1, 1],
])
b_ub = np.array([8, 10, 9, 8, 8, 5, 10, 2, 2, 2, 3, 10, 10] + [0]*9)
result = linprog(c, A_ub=-A_ub, b_ub=-b_ub, bounds=bounds)
```

Linear Program

Question

The firm is then informed that the client hopes the project can be completed in 40 days. Here are the cost of reducing the duration of each activity by a day and the maximum possible reduction. What is the minimum additional cost required for the project to be completed in 40 days?

Reduction cost	Maximum	Reduction
5000	2	x_A
9000	2	x_B
7000	4	x_C
4500	3	x_D
1500	1	x_E
3500	3	x_F
-	0	x_G
4500	1	x_H
9500	3	x_I

$$\text{minimize} \quad 5000 \cdot x_A + 9000 \cdot x_B + 7000 \cdot x_C + 4500 \cdot x_D + 1500 \cdot x_E + 3500 \cdot x_F + 4\,500 \cdot x_H + 9500 \cdot x_I$$

$$\text{subject to} \quad 0 \le x_A \le 2 \qquad T_A - (8 - x_A) \ge 0$$
$$0 \le x_B \le 2 \qquad T_B - (10 - x_B) \ge T_A$$
$$0 \le x_C \le 4 \qquad T_C - (9 - x_C) \ge T_B$$
$$0 \le x_D \le 3 \qquad T_D - (8 - x_D) \ge T_B, T_C$$
$$0 \le x_E \le 1 \qquad T_E - (5 - x_E) \ge T_B$$
$$0 \le x_F \le 3 \qquad T_F - (10 - x_F) \ge T_E$$
$$x_G = 0 \qquad T_G - (2 - x_G) \ge T_C, T_D, T_F$$
$$0 \le x_H \le 1 \qquad T_H - (3 - x_H) \ge T_F$$
$$0 \le x_I \le 3 \qquad T_I - (10 - x_I) \ge T_G, T_H$$
$$T_A, \dots, T_I \le 40$$

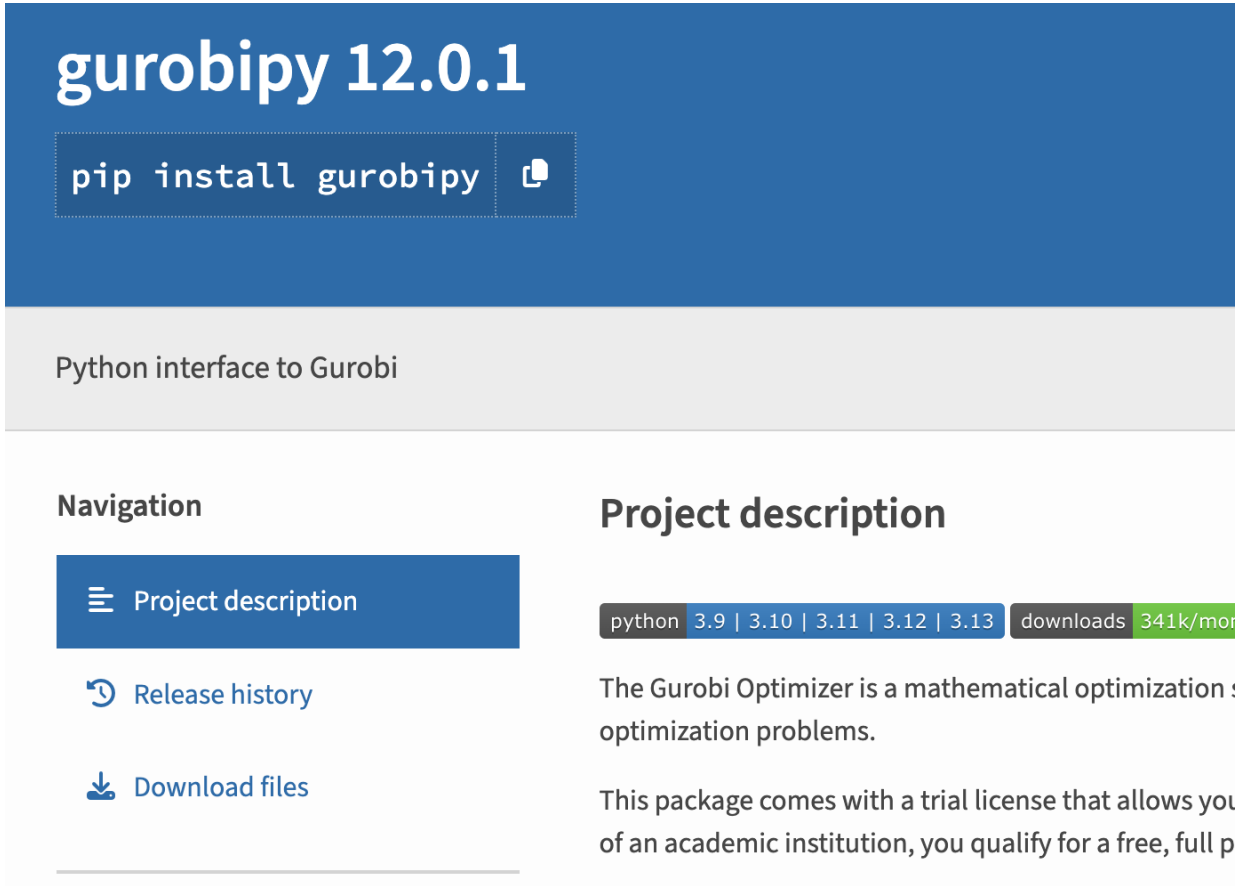
➡ Solve it via linprog?
Could be a nightmare

Linear Program

CVXPY



Gurobipy



Linear Program

```
import cvxpy as cp

xA = cp.Variable(name="xA", nonneg=True)
.....
xI = cp.Variable(name="xI", nonneg=True)

TA = cp.Variable(name="TA", nonneg=True)
.....
TI = cp.Variable(name="TI", nonneg=True)

constraints = [xA <= 2,
   xB <= 2,
   xC <= 4,
   xD <= 3,
   xE <= 1,
   xF <= 3,
   xG <= 0,  # effectively fixes xG to 0
   xH <= 1,
   xI <= 3]
```

```
constraints += [
   TA == 8 - xA,
   TB == TA + 10 - xB,
   TC == TB + 9 - xC,
   TD >= TB + 8 - xD,
   TD >= TC + 8 - xD,
   TE == TB + 5 - xE,
   TF == TE + 10 - xF,
   TG >= TC + 2 - xG,
   TG >= TD + 2 - xG,
   TG >= TF + 2 - xG,
   TH == TF + 3 - xH,
   TI >= TH + 10 - xI,
   TI >= TG + 10 - xI,
   TA <= 40,
   .....
   TI <= 40
]
objective = cp.Minimize(5000*xA + 9000*xB + 7000*xC
+ 4500*xD + 1500*xE + 3500*xF + 4500*xH + 9500*xI)
problem = cp.Problem(objective, constraints)
problem.solve()
```

Linear Program

```
from gurobipy import Model, GRB

model = Model()
# model.setParam('Method', 2)
xA = model.addVar(lb=0, ub=2, name="xA")
xB = model.addVar(lb=0, ub=2, name="xB")
xC = model.addVar(lb=0, ub=4, name="xC")
xD = model.addVar(lb=0, ub=3, name="xD")
xE = model.addVar(lb=0, ub=1, name="xE")
xF = model.addVar(lb=0, ub=3, name="xF")
xG = model.addVar(lb=0, ub=0, name="xG")
xH = model.addVar(lb=0, ub=1, name="xH")
xI = model.addVar(lb=0, ub=3, name="xI")

TA = model.addVar(lb=0, name="TA")
TB = model.addVar(lb=0, name="TB")
TC = model.addVar(lb=0, name="TC")
TD = model.addVar(lb=0, name="TD")
TE = model.addVar(lb=0, name="TE")
TF = model.addVar(lb=0, name="TF")
TG = model.addVar(lb=0, name="TG")
TH = model.addVar(lb=0, name="TH")
TI = model.addVar(lb=0, name="TI")

model.setObjective(5000*xA + 9000*xB + 7000*xC + 4500*xD
+ 1500*xE + 3500*xF + 4500*xH + 9500*xI, GRB.MINIMIZE)

model.addConstr(TA == 8 - xA)
model.addConstr(TB == TA + 10 - xB)
model.addConstr(TC == TB + 9 - xC)
model.addConstr(TD >= TB + 8 - xD)
model.addConstr(TD >= TC + 8 - xD)
model.addConstr(TE == TB + 5 - xE)
model.addConstr(TF == TE + 10 - xF)
model.addConstr(TG >= TC + 2 - xG)
model.addConstr(TG >= TD + 2 - xG)
model.addConstr(TG >= TF + 2 - xG)
model.addConstr(TH == TF + 3 - xH)
model.addConstr(TI >= TH + 10 - xI)
model.addConstr(TI >= TG + 10 - xI)
model.addConstr(TA <= 40)
.....
model.addConstr(TI <= 40)

model.optimize()
```

Quadratic Program

Constrained Program

- Linear Program
- Quadratic Program

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \le 0, \ i = 1, \dots, m \\ & h_i(x) = 0, \ i = 1, \dots, p \end{aligned}$$

$f(x)$: quadratic

$g_i(x)$: linear

$h_i(x)$: linear

- .....

Quadratic Program

Quadratic programs, like linear programs, can be solved via specialized algorithms

(Some are based on the KKT conditions)

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && \text{quadratic} \\ \text{subject to} \quad & g_i(x) \leq 0, \ i = 1, \ldots, m && \text{linear} \\ & h_i(x) = 0, \ i = 1, \ldots, p && \text{linear} \end{aligned}$$

Our focus today: how to invoke solvers

Quadratic Program

MATLAB

quadprog

Quadratic programming

Syntax

```
x = quadprog(H,f)
x = quadprog(H,f,A,b)
x = quadprog(H,f,A,b,Aeq,beq)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
x = quadprog(problem)
[x,fval] = quadprog(__)
[x,fval,exitflag,output] = quadprog(__)
[x,fval,exitflag,output,lambda] = quadprog(__)

[wsout,fval,exitflag,output,lambda] = quadprog(H,f,A,b,Aeq,beq,lb,ub,ws)
```

Description

Solver for quadratic objective functions with linear constraints.

quadprog finds a minimum for a problem specified by

$$\min_x \frac{1}{2} x^T H x + f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

H, A, and Aeq are matrices, and f, b, beq, lb, ub, and x are vectors.

You can pass f, lb, and ub as vectors or matrices; see Matrix Arguments.

SciPy

No dedicated algorithms

for

quadratic programs

You may use

CVXPY **Gurobipy**

Quadratic Program

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2} \cdot x^{\mathrm{T}} \cdot P \cdot x + q^{\mathrm{T}} \cdot x \\ \text{subject to} \quad & G \cdot x \leq h \\ & A \cdot x = b \end{aligned}$$

```
import cvxpy as cp

x = cp.Variable(n)
objective = cp.Minimize((1/2)*cp.quad_form(x, P) + q.T @ x),
constraints = [G @ x <= h,  A @ x == b]

prob = cp.Problem(objective, constraints)
prob.solve()

print("The optimal value is", prob.value)
print("A solution x is")
print(x.value)
```

Nonlinear Program with Linear Constraints

Constrained Program

- Linear Program
- Quadratic Program
- Nonlinear Program with Linear Constraints

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) \quad \text{(nonlinear)} \\ \text{subject to} \quad & g_i(x) \le 0, \ i = 1, \dots, m \quad \text{(linear)} \\ & h_i(x) = 0, \ i = 1, \dots, p \quad \text{(linear)} \end{aligned}$$

- ……

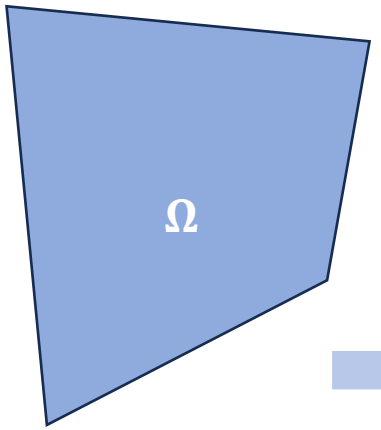
Nonlinear Program with Linear Constraints

$$\begin{aligned} &\text{minimize}_{x \in \mathbb{R}^n} && f(x) && \text{nonlinear} \\ &\text{subject to} && g_i(x) \le 0,\ i = 1, \dots, m && \text{linear} \\ & && h_i(x) = 0,\ i = 1, \dots, p && \text{linear} \end{aligned}$$

$\mathcal{G}_i = \{x : g_i(x) \le 0\}$

$\mathcal{H}_i = \{x : h_i(x) = 0\}$

$$\begin{aligned} &\text{minimize}_{x \in \mathbb{R}^n} && f(x) && \text{nonlinear} \\ &\text{subject to} && x \in \Omega && \text{polyhedron} \end{aligned}$$



polyhedron

$\Omega = (\cap_{i=1}^{m} \mathcal{G}_i) \cap (\cap_{i=1}^{p} \mathcal{H}_i)$

Frank-Wolfe Algorithm

$$\begin{aligned} &\text{minimize} && f(x) \\ &x \in \mathbb{R}^n \\ &\text{subject to} && x \in \Omega \end{aligned}$$

→ nonlinear

core idea: linearization

Start from an $x^0 \in \Omega$

For $k = 0, 1, \ldots$

Solve the following linear program

$$\begin{aligned} &\text{minimize} && f(x^k) + \nabla f(x^k)^{\mathrm{T}} \cdot (y - x^k) \\ &y \in \mathbb{R}^n \\ &\text{subject to} && x \in \Omega \end{aligned}$$

→ the first-order approximation of $f(x)$ at x^k

Frank-Wolfe Algorithm

Start from an $x^0 \in \Omega$

For $k = 0, 1, \ldots$

Solve the following linear program

$$\begin{aligned} \text{minimize} \quad & f(x^k) + \nabla f(x^k)^{\mathrm{T}} \cdot (y - x^k) \\ \text{subject to} \quad & y \in \Omega \end{aligned}$$

$d^k = y^k - x^k$

is guaranteed to be a descent direction

$\nabla f(x^k)^{\mathrm{T}} \cdot (y^k - x^k) \leq 0$

Optimal solution: y^k

$\nabla f(x^k)^{\mathrm{T}} \cdot y^k \leq \nabla f(x^k)^{\mathrm{T}} \cdot x, \quad \forall x \in \Omega$ ⇒ $\nabla f(x^k)^{\mathrm{T}} \cdot y^k \leq \nabla f(x^k)^{\mathrm{T}} \cdot x^k$

Frank-Wolfe Algorithm

Start from an $x^0 \in \Omega$

For $k = 0, 1, \ldots$

Let y^k be the solution to the following linear program and then let $d^k = y^k - x^k$

$$\begin{aligned} \underset{y \in \mathbb{R}^n}{\text{minimize}} \quad & f(x^k) + \nabla f(x^k)^T \cdot (y - x^k) \\ \text{subject to} \quad & x \in \Omega \end{aligned}$$

Set $x^{k+1} = x^k + \alpha^k \cdot d^k$ $\qquad$ $\alpha^k \in (0, 1]$

Is x^{k+1} feasible? $\qquad$ Yes for all $\alpha^k \in [0, 1]$ because Ω is a convex set

Frank-Wolfe Algorithm

Start from an $x^0 \in \Omega$

For $k = 0, 1, \ldots$

Let y^k be the solution to the following linear program and then let $d^k = y^k - x^k$

$$\begin{aligned} &\text{minimize} && f(x^k) + \nabla f(x^k)^{\mathrm{T}} \cdot (y - x^k) \\ &y \in \mathbb{R}^n \\ &\text{subject to} && x \in \Omega \end{aligned}$$

Set $x^{k+1} = x^k + \alpha^k \cdot d^k$ $\qquad \alpha^k \in (0, 1]$

Question 1: How to determine α^k?

Question 2: When can we terminate the algorithm?

Projected Gradient Descent

Question 2: When can we terminate the algorithm?

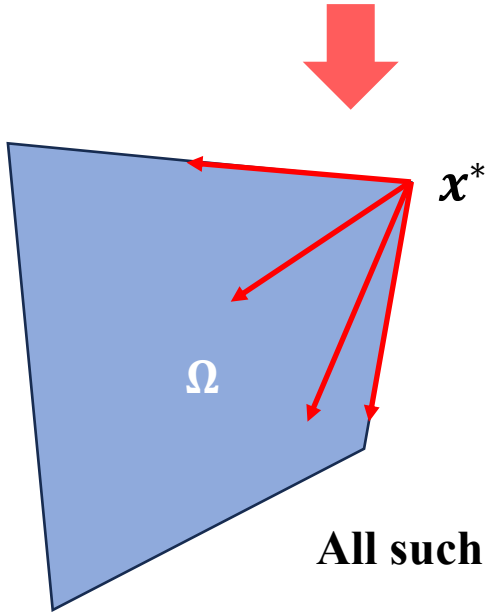
$$\begin{aligned} &\underset{x \in \mathbb{R}^n}{\text{minimize}} && f(x) \\ &\text{subject to} && x \in \Omega \end{aligned}$$

Descent directions:

$$\{d \in \mathbb{R}^n : \nabla f(x)^T \cdot d \leq 0\}$$

x^* is an optimal solution to the problem if and only if

$$\nabla f(x^*)^T \cdot (x - x^*) \geq 0, \quad \forall x \in \Omega$$



All such directions are not descent

Frank-Wolfe Algorithm

Question 2: When can we terminate the algorithm?

Let y^k be the solution to the following linear program and then let $d^k = y^k - x^k$

$$\begin{aligned} \text{minimize} \quad & f(x^k) + \nabla f(x^k)^{\mathrm{T}} \cdot (y - x^k) \\ y \in \mathbb{R}^n & \\ \text{subject to} \quad & x \in \Omega \end{aligned}$$

$$\nabla f(x^k)^{\mathrm{T}} \cdot y^k \leq \nabla f(x^k)^{\mathrm{T}} \cdot x, \qquad \forall x \in \Omega$$

$$\nabla f(x^k)^{\mathrm{T}} \cdot (y^k - x^k) \leq 0$$

Termination condition:

$$-\nabla f(x^k)^{\mathrm{T}} \cdot (y^k - x^k) < \epsilon$$

If $\nabla f(x^k)^{\mathrm{T}} \cdot (y^k - x^k) = 0$, then x^k is the optimal solution

$$\nabla f(x^k)^{\mathrm{T}} \cdot x^k \leq \nabla f(x^k)^{\mathrm{T}} \cdot x, \qquad \forall x \in \Omega$$

Frank-Wolfe Algorithm

Start from an $x^0 \in \Omega$

For $k = 0, 1, \ldots$

Let y^k be the solution to the following linear program and then let $d^k = y^k - x^k$

$$\begin{aligned} \underset{y \in \mathbb{R}^n}{\text{minimize}} \quad & f(x^k) + \nabla f(x^k)^{\mathrm{T}} \cdot (y - x^k) \\ \text{subject to} \quad & y \in \Omega \end{aligned}$$

If $\text{gap} = -\nabla f(x^k)^{\mathrm{T}} \cdot (y^k - x^k) < \epsilon$, break

Set $x^{k+1} = x^k + \alpha^k \cdot d^k$, where $\alpha^k \in (0, 1]$

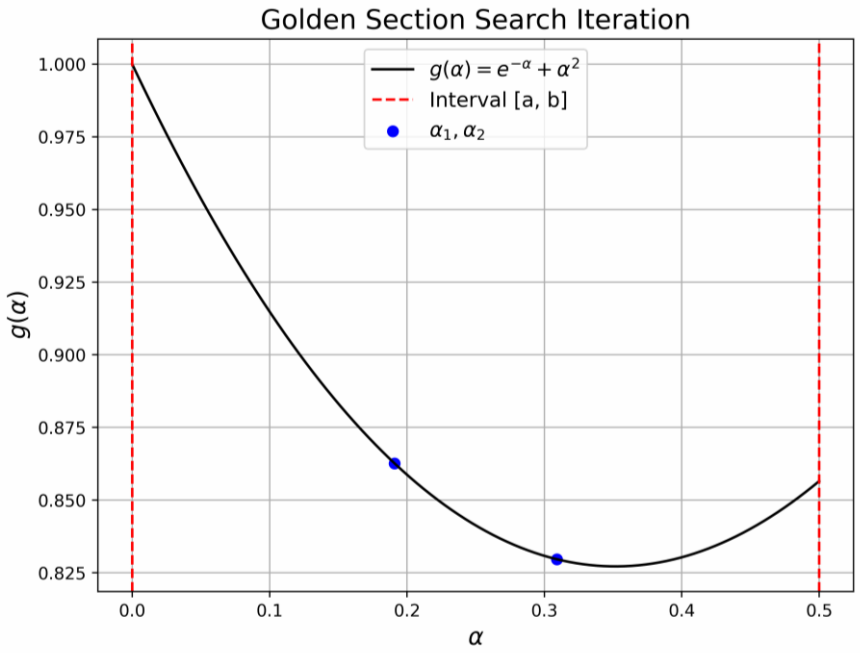
Frank-Wolfe Algorithm

Question 1: How to determine α^k?

Set $x^{k+1} = x^k + \alpha^k \cdot d^k$, where $\alpha^k \in (0, 1]$

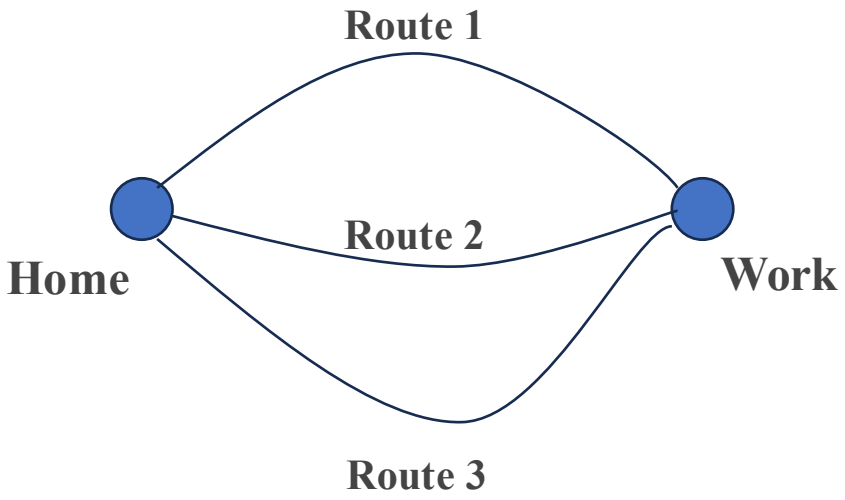
Strategy I: Set $\alpha^k = \frac{1}{k+2}$

Strategy II: Perform an exact line search



Frank-Wolfe Algorithm

Consider a small town with $q = 10$ residents who drive from "Home" to "Work" every morning.



x_i: The number of travelers choosing Route i

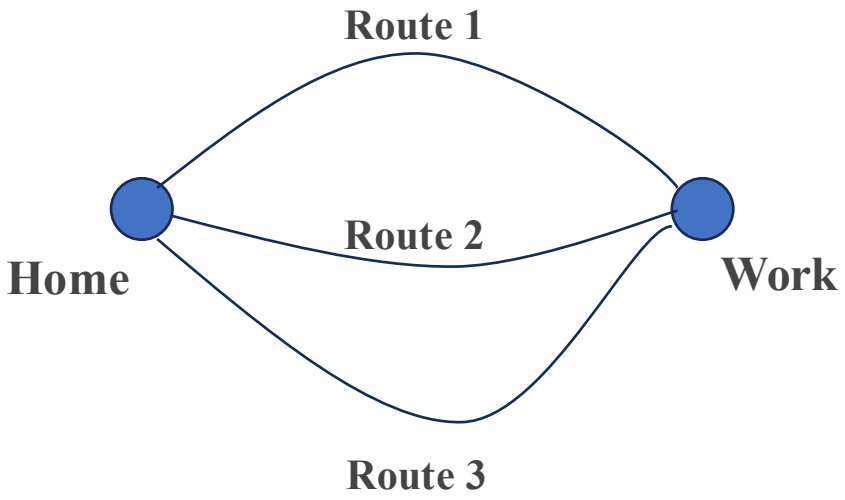
$c_i(x_i)$: The travel time of using Route i

$$\begin{cases} c_1(x_1) = 4 + x_1^2 \\ c_2(x_2) = 25 + x_2^2 \\ c_3(x_3) = 50 + x_3^2 \end{cases}$$

Equilibrium state: A non-optimal route is used by nobody; a used route must be optimal.

Frank-Wolfe Algorithm

Consider a small town with $q = 10$ residents who drive from "Home" to "Work" every morning.



$$f(x) = 4x_1 + \frac{4}{3} \cdot x_1^3 + 25x_2 + \frac{24}{3} \cdot x_2^3 + 50x_x + \frac{50}{3} \cdot x_3^3$$

$$\text{minimize } f(x) = \sum_{i=1,2,3} \int_0^{x_i} c_i(v)\mathrm{d}v$$

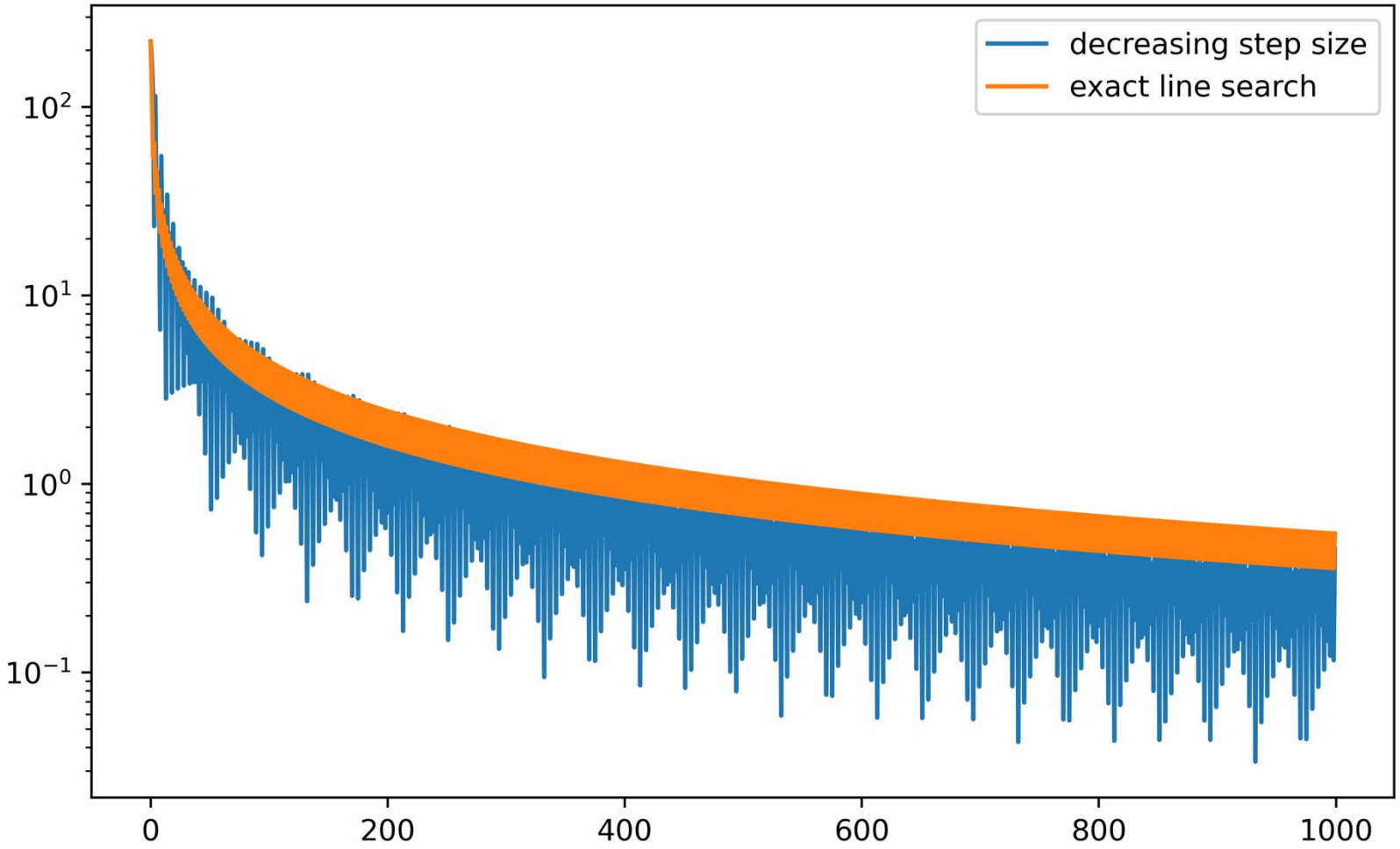
$$\text{subject to } x_1 + x_2 + x_3 = q$$

$$x_1, x_2, x_3 \geq 0$$

Test Frank-Wolfe with different step size strategies

Report how gap $= -\nabla f\left(x^k\right)^{\mathrm{T}} \cdot \left(y^k - x^k\right)$ evolves with iterations

Frank-Wolfe Algorithm



Projected Gradient Descent

$$\begin{aligned} \text{minimize} \quad & f(x) \\ x \in \mathbb{R}^n & \end{aligned}$$

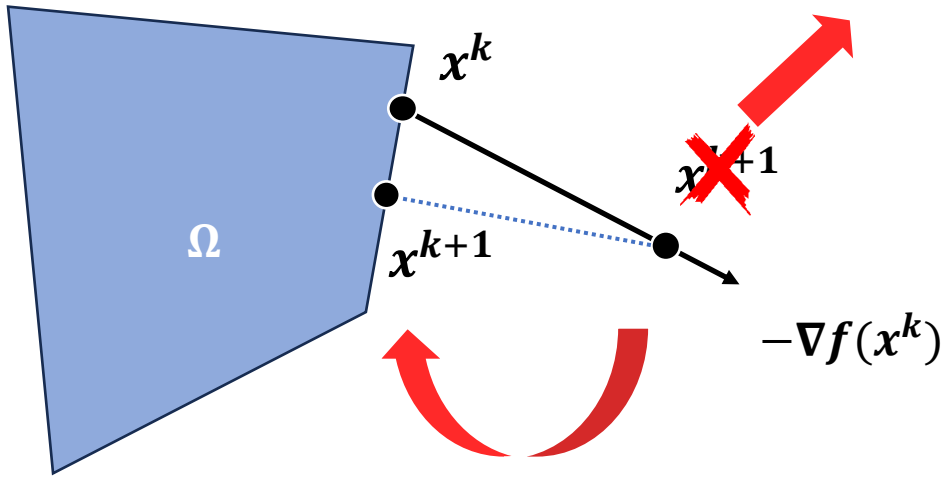
$$\begin{aligned} \text{minimize} \quad & f(x) \\ x \in \mathbb{R}^n & \\ \text{subject to} \quad & x \in \Omega \end{aligned}$$

For $k = 0, 1, \ldots$

Compute $\nabla f(x^k)$

Set $x^{k+1} = x^k - \alpha^k \cdot \nabla f(x^k)$

It can be infeasible



Project back to Ω

Projected Gradient Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

For $k = 0, 1, \ldots$

Compute $\nabla f(x^k)$

Set $x^{k+1} = x^k - \alpha^k \cdot \nabla f(x^k)$

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \\ &\text{subject to} \quad x \in \Omega \end{aligned}$$

For $k = 0, 1, \ldots$

Compute $\nabla f(x^k)$

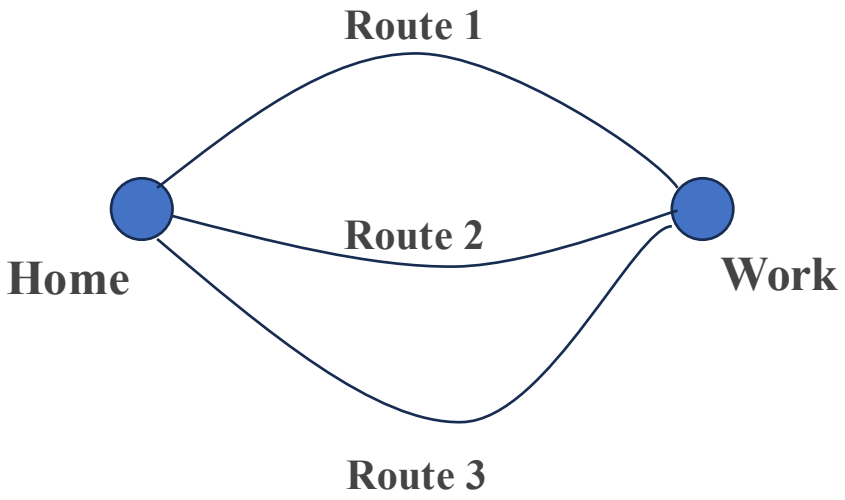
Set $y^k = x^k - \alpha^k \cdot \nabla f(x^k)$

Set $x^{k+1} = \underset{x \in \Omega}{\text{argmin}} \, ||x - y^k||_2^2$

Quadratic program

Projected Gradient Descent

Consider a small town with $q = 10$ residents who drive from "Home" to "Work" every morning.



$$f(x) = 4x_1 + \frac{4}{3} \cdot x_1^3 + 25x_2 + \frac{24}{3} \cdot x_2^3 + 50x_x + \frac{50}{3} \cdot x_3^3$$

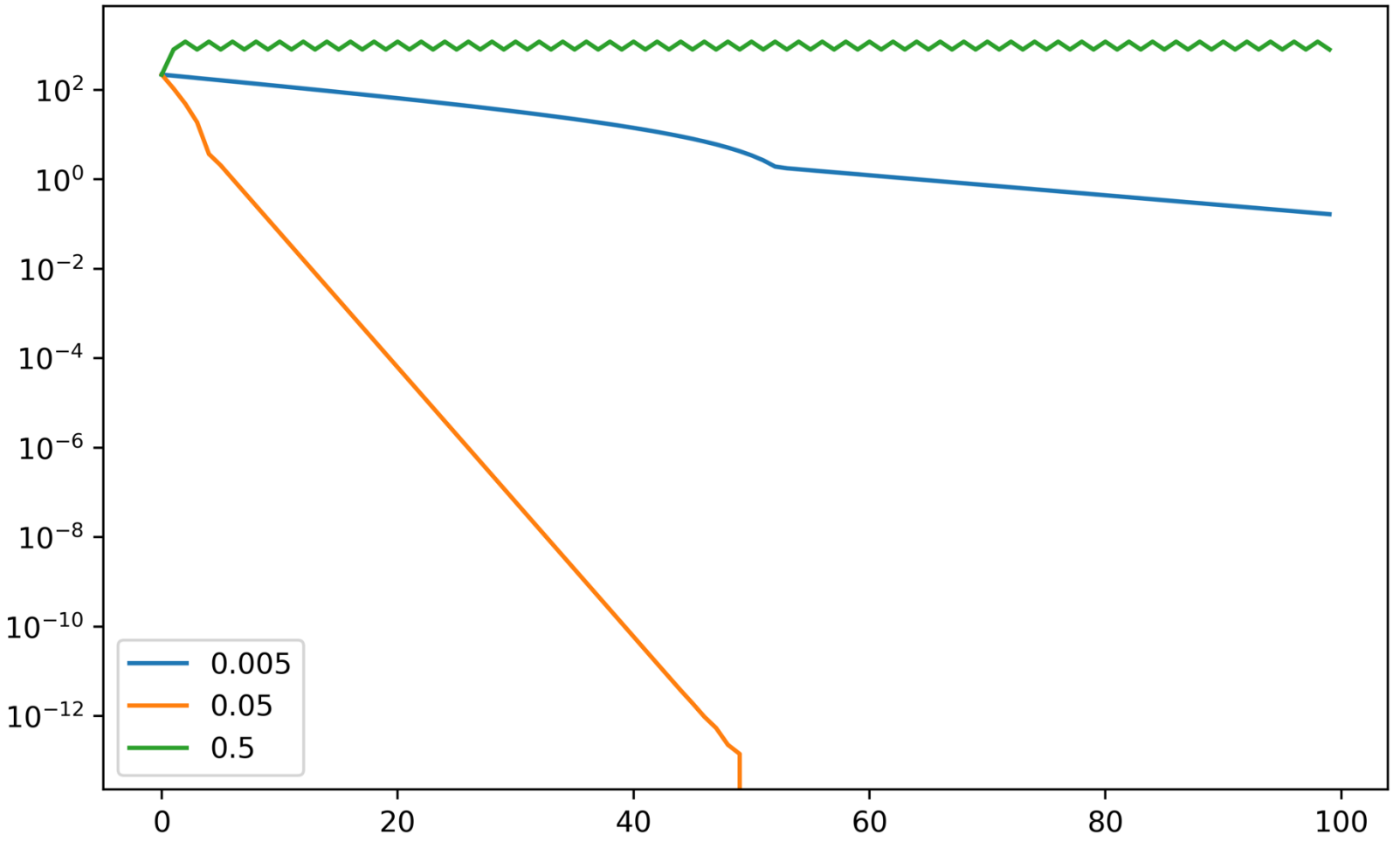
$$\text{minimize } f(x) = \sum_{i=1,2,3} \int_0^{x_i} c_i(v)\mathrm{d}v$$

$$\text{subject to } x_1 + x_2 + x_3 = q$$

$$x_1, x_2, x_3 \geq 0$$

Test projected gradient descent with different step sizes

Projected Gradient Descent



Optimization Problems

Constrained Program

- Linear Program
- Quadratic Program
- Nonlinear Program with Linear Constraints
- General Nonlinear Program

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \le 0, \ i = 1, \dots, m \\ & h_i(x) = 0, \ i = 1, \dots, p \end{aligned}$$

$f(x)$: nonlinear

$g_i(x)$: nonlinear

$h_i(x)$: nonlinear

Sequential Quadratic Programming

$$\begin{aligned} \underset{x \in \mathbb{R}^n}{\text{minimize}} \quad & f(x) && \text{nonlinear} \\ \text{subject to} \quad & g_i(x) \le 0, \ i = 1, \dots, m && \text{nonlinear} \\ & h_i(x) = 0, \ i = 1, \dots, p && \text{nonlinear} \end{aligned}$$

For $k = 0, 1, \dots$

Set $x^{k+1} = x^k + d$, where d is the optimal solution to the following quadratic program

$$\begin{aligned} \underset{d \in \mathbb{R}^n}{\text{minimize}} \quad & f(x^k) + \nabla f(x^k)^{\mathrm{T}} \cdot d + d^{\mathrm{T}} \cdot \nabla^2 f(x^k) \cdot d && \text{second-order approximation} \\ \text{subject to} \quad & g_i(x^k) + \nabla g_i(x^k)^{\mathrm{T}} \cdot d \le 0, \ i = 1, \dots, m && \text{first-order approximation} \\ & h_i(x^k) + \nabla h_i(x^k)^{\mathrm{T}} \cdot d = 0, \ i = 1, \dots, p && \text{first-order approximation} \end{aligned}$$

Sequential Quadratic Programming

$$\begin{aligned} \underset{x \in \mathbb{R}^2}{\text{minimize}} \quad & x_1^4 + x_2^2 \\ \text{subject to} \quad & x_1^2 - x_1 - x_2 + 1 \leq 0 \\ & x_1^2 - 4x_1 - x_2 + 6 \leq 0 \\ & x_1^2 - 3x_1 + x_2 - 2 \leq 0 \end{aligned}$$

```
import numpy as np
import cvxpy as cp
from scipy.optimize import minimize

def f(x):
    return x[0]**4 + x[1]**4

def g1(x):
    return x[0] ** 2 - x[0] + 1 - x[1]

def g2(x):
    return x[0] ** 2 - 4 * x[0] + 6 - x[1]

def g3(x):
    return x[1] + x[0] ** 2 - 3 * x[0] - 2
```

scipy

```
x0 = np.array([0.0, 0.0])
cons = [
    {'type': 'ineq', 'fun': lambda x: -g1(x)},
    {'type': 'ineq', 'fun': lambda x: -g2(x)},
    {'type': 'ineq', 'fun': lambda x: -g3(x)}
]
res = minimize(f, x0=x0, constraints=cons)
print("SciPy SLSQP's solution:", res.x)
```

cvxpy

```
x = cp.Variable(2)
objective = cp.Minimize(x[0]**4 + x[1]**4)
constraints = [
    x[1] >= x[0]**2 - x[0] + 1,
    x[1] >= x[0]**2 - 4*x[0] + 6,
    x[1] <= -x[0]**2 + 3*x[0] + 2
]
problem = cp.Problem(objective, constraints)
problem.solve()
print("CVXPY's solution':", x.value)
```

Sequential Quadratic Programming

```
def gradient_f(x):
    return np.array([4 * x[0] ** 3, 4 * x[1]
** 3])

def hessian_f(x):
    return np.array([
        [12 * x[0] ** 2, 0],
        [0, 12 * x[1] ** 2]
    ])

def gradient_g1(x):
    return np.array([2 * x[0] - 1, -1])

def gradient_g2(x):
    return np.array([2 * x[0] - 4, -1])

def gradient_g3(x):
    return np.array([2 * x[0] - 3, 1])
```

```
for k in range(max_iter):
        grad = gradient_f(x)
        g1_val = g1(x)
        g2_val = g2(x)
        g3_val = g3(x)
        grad_g1 = gradient_g1(x)
        grad_g2 = gradient_g2(x)
        grad_g3 = gradient_g3(x)
        H = hessian_f(x)

        def qp_objective(d):
            return 0.5 * d.T @ H @ d + grad @ d

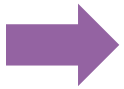
        constraints = [
            {'type': 'ineq', 'fun': lambda d: -g1_val - grad_g1 @ d},
            {'type': 'ineq', 'fun': lambda d: -g2_val - grad_g2 @ d},
            {'type': 'ineq', 'fun': lambda d: -g3_val - grad_g3 @ d}
        ]
        res = minimize(qp_objective, x0=x, constraints=constraints)
        d = res.x
        x += d
        if np.linalg.norm(d) < tol:
            break
```

Sequential Quadratic Programming

For $k = 0, 1, \ldots$

Set $x^{k+1} = x^k + d$, where d is the optimal solution to the following quadratic program

$$\begin{aligned} \underset{d \in \mathbb{R}^n}{\text{minimize}} \quad & f(x^k) + \nabla f(x^k)^{\mathrm{T}} \cdot d + d^{\mathrm{T}} \cdot \nabla^2 f(x^k) \cdot d \\ \text{subject to} \quad & g_i(x^k) + \nabla g_i(x^k)^{\mathrm{T}} \cdot d \leq 0, \ \ i = 1, \ldots, m \\ & h_i(x^k) + \nabla h_i(x^k)^{\mathrm{T}} \cdot d = 0, i = 1, \ldots, p \end{aligned}$$



Unfortunately, this algorithmic framework is not robust for all types of nonlinear programs

Sequential Quadratic Programming

A much more practical version of SQP is SLSQP (Sequential Least Squares Programming)

minimize(method='SLSQP')

```
minimize(fun, x0, args=(), method=None, jac=None, hess=None, hessp=None,
bounds=None, constraints=(), tol=None, callback=None, options=None)
```

Minimize a scalar function of one or more variables using Sequential Least Squares Programming (SLSQP).

See also

For documentation for the rest of the parameters, see `scipy.optimize.minimize`