# 최적화: 선형 계획법, 정수 선형 계획법, 혼합 정수 선형 계획법

- 주어진 제약 조건 하에서 특정 목적 함수(Objective Function)의 값을 최소화하거나 최대화하는 결정 변수(Decision Variables)의 값을 찾는 수학적 기법
- 자원 할당, 생산 계획, 스케줄링, 포트폴리오 관리 등 다양한 분야에서 가장 효율적인 의사결정을 내리는 데 활용됩니다.

### 적용 가능한 상황

- **자원 할당**: 제한된 자원(예: 예산, 인력, 시간)을 여러 프로젝트나 활동에 어떻게 배분해야 최대의 이익을 얻거나 최소의 비용을 발생시킬 수 있는지 결정할 때.
- **생산 계획**: 생산 설비의 가동률, 원자재 재고, 수요 예측 등을 고려하여 생산량을 최적화할 때.
- **운송 및 물류**: 운송 비용을 최소화하거나 배송 시간을 단축하기 위한 최적의 경로를 찾을 때.
- **포트폴리오 최적화**: 투자 수익을 극대화하면서 위험을 최소화하는 자산 배분 전략을 수립할 때.
- **스케줄링**: 작업, 인력, 장비 등의 일정을 최적화하여 효율성을 높일 때.

## 1. 선형 계획법 (Linear Programming, LP)

- 목적 함수와 모든 제약 조건이 선형 방정식 또는 부등식으로 표현될 수 있는 최적화 문제
- 결정 변수는 연속적인 값을 보유해야 함
  - 결정 변수: 목표함수를 최대화/최소화 하기 위해 조정하는 변수
- 가장 기본적인 최적화 기법 중 하나로, 다양한 산업 분야에서 널리 사용

### 주의사항

- 문제의 모든 관계가 선형이어야 합니다. 비선형적인 관계를 모델링할 수 없습니다.
- 결정 변수는 연속적인 값을 가질 수 있어야 합니다. 이산적인 결정(예: 예/아니오, 개수)을 직접 모델링하기 어렵습니다.

### 코드 예시 (`scipy.optimize.linprog` 사용)

`scipy.optimize.linprog`는 목적 함수를 최소화하는 형태로 문제를 해결합니다. 따라서 최대화 문제는 목적 함수에 -1을 곱하여 최소화 문제로 변환해야 합니다.
`linprog` 함수 자체에는 직접적인 하이퍼파라미터는 없지만, 문제 정의에 사용되는 인자들이 중요합니다.

- `c`: 목적 함수의 계수 벡터입니다. 최소화 문제의 경우 그대로 사용하고, 최대화 문제의 경우 각 계수에 -1을 곱하여 전달합니다.
- `A_ub`, `b_ub`: 부등식 제약 조건 `A_ub @ x <= b_ub`를 정의합니다.
- `A_eq`, `b_eq`: 등식 제약 조건 `A_eq @ x == b_eq`를 정의합니다.
- `bounds`: 각 결정 변수의 하한과 상한을 튜플 리스트 형태로 정의합니다. `(None, None)`은 제약이 없음을 의미합니다.
- `method`: 사용할 최적화 알고리즘을 지정합니다. (예: 'highs', 'interior-point', 'revised simplex'). 'highs'가 일반적으로 빠르고 안정적입니다.

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

# 예시: 생산 계획 문제
# 두 가지 제품 A, B를 생산. 이익을 최대화하는 생산량 결정.
# 제품 A: 생산 시간 1시간, 재료 2단위, 이익 3만원
# 제품 B: 생산 시간 2시간, 재료 1단위, 이익 2만원
# 제약 조건: 총 생산 시간 8시간, 총 재료 6단위

# 목적 함수 계수 (최대화 문제이므로 -1을 곱함)
# -3 * x_A - 2 * x_B 를 최소화
c = np.array([-3, -2])

# 부등식 제약 조건 (Ax <= b)
# 생산 시간: 1*x_A + 2*x_B <= 8
# 재료: 2*x_A + 1*x_B <= 6
A = np.array([[1, 2], [2, 1]])
b = np.array([8, 6])

# 변수 범위 (x_A >= 0, x_B >= 0)
x0_bounds = (0, None) # x_A >= 0
x1_bounds = (0, None) # x_B >= 0

# 선형 계획법 풀이
res = linprog(c, A_ub=A, b_ub=b, bounds=[x0_bounds, x1_bounds], method='highs')

print("--- 선형 계획법 결과 ---")
if res.success:
    print(f"최적 생산량 (제품 A, B): {res.x}")
    print(f"최대 이익: {-res.fun:.2f}만원") # -1을 곱했으므로 다시 -1을 곱함
else:
    print("최적해를 찾지 못했습니다.")
print(f"상태: {res.message}")
'''
최적 생산량 (제품 A, B): [1.33333333 3.33333333]
최대 이익: 10.67만원
상태: Optimization terminated successfully.
'''
```

### 결과 해석 방법

- `res.success`: 최적화가 성공적으로 완료되었는지 여부를 나타내는 불리언 값입니다.
- `res.x`: 최적해를 나타내는 결정 변수들의 배열입니다. 각 변수의 최적 값을 의미합니다.
- `res.fun`: 최적해에서의 목적 함수 값입니다. 최대화 문제의 경우 -1을 곱한 값이므로, 다시 -1을 곱하여 원래의 최대 이익을 구해야 합니다.
- `res.message`: 최적화 과정의 상태 메시지를 제공합니다.

---

## 2. 정수 선형 계획법 (Integer Linear Programming, ILP)

- 선형 계획법과 동일하게 목적 함수와 제약 조건이 선형이지만, 일부 또는 모든 결정 변수가 정수 값만 가질 수 있는 경우에 사용
- 생산량이나 인력의 수가 소수점이 될 수 없는 경우에 유용

### 주의사항

- 선형 계획법보다 훨씬 복잡하고 계산 비용이 많이 듭니다. NP-hard 문제에 속합니다.
  - NP-hard 문제: 다항 시간 내에 풀 수 없는 문제 (지수 방정식으로 풀 수 있는 문제)
- `scipy.optimize.linprog`는 정수 제약을 직접 지원하지 않으므로, `PuLP`, `ortools`, `Gurobi`, `CPLEX`와 같은 전문 최적화 라이브러리가 필요합니다.

### 코드 예시 (`PuLP` 사용)

`PuLP`는 파이썬에서 선형 및 정수 계획 문제를 모델링하고 해결하는 데 사용되는 라이브러리입니다. 설치가 필요합니다 (`pip install pulp`).
`PuLP`는 모델링 언어에 가깝기 때문에 `linprog`와 같은 직접적인 하이퍼파라미터보다는 문제 정의 요소들이 중요합니다.

- `LpProblem(name, sense)`: 최적화 문제를 정의합니다. `name`은 문제의 이름, `sense`는 `LpMaximize` (최대화) 또는 `LpMinimize` (최소화)를 지정합니다.
- `LpVariable(name, lowBound, upBound, cat)`: 결정 변수를 정의합니다. `name`은 변수 이름, `lowBound`와 `upBound`는 변수의 하한과 상한, `cat`은 변수의 유형을 지정합니다. `LpContinuous` (연속), `LpInteger` (정수), `LpBinary` (이진)가 있습니다.

```
from pulp import *

# 예시: 투자 선택 문제
# 3가지 프로젝트에 투자할지 말지 결정. 총 예산 1000만원.
# 프로젝트 1: 비용 400만원, 수익 500만원
# 프로젝트 2: 비용 600만원, 수익 700만원
# 프로젝트 3: 비용 300만원, 수익 350만원

# 문제 정의 (최대화 문제)
prob = LpProblem("Investment Problem", LpMaximize)

# 결정 변수 정의 (이진 변수: 0 또는 1)
# x1, x2, x3는 각 프로젝트에 투자할지 말지를 나타내는 이진 변수
x1 = LpVariable("x1", 0, 1, LpBinary)
x2 = LpVariable("x2", 0, 1, LpBinary)
x3 = LpVariable("x3", 0, 1, LpBinary)

# 목적 함수 (총 수익 최대화)
prob += 500 * x1 + 700 * x2 + 350 * x3, "Total Profit"

# 제약 조건 (총 예산 1000만원)
prob += 400 * x1 + 600 * x2 + 300 * x3 <= 1000, "Budget Constraint"

# 문제 풀이
prob.solve()

print("--- 정수 선형 계획법 결과 ---")
```

```
print(f"상태: {LpStatus[prob.status]}")

if LpStatus[prob.status] == "Optimal":
    print(f"최적 투자 결정:")
    for v in prob.variables():
        print(f"{v.name} = {v.varValue}")
    print(f"최대 총 수익: {value(prob.objective)}만원")
else:
    print("최적해를 찾지 못했습니다.")
'''
상태: Optimal
최적 투자 결정:
x1 = 1.0
x2 = 1.0
x3 = 0.0
최대 총 수익: 1200.0만원
'''
```

### 결과 해석 방법

- `LpStatus[prob.status]`: 문제 해결의 상태를 나타냅니다 (예: Optimal, Not Solved, Infeasible).
- `prob.variables()`: 각 결정 변수의 최적 값을 확인할 수 있습니다.
- `value(prob.objective)`: 최적해에서의 목적 함수 값을 반환합니다.

---

## 3. 혼합 정수 선형 계획법 (Mixed Integer Linear Programming, MILP)

- 정수 선형 계획법의 확장으로, 결정 변수 중 일부는 연속형이고 일부는 정수형(또는 이진형)인 경우에 사용
- 실제 세계의 많은 최적화 문제는 연속적인 결정과 이산적인 결정을 동시에 포함
  → MILP는 매우 강력한 모델링 도구이다.

### 주의사항

- ILP와 마찬가지로 계산 비용이 매우 높으며, 문제의 크기가 커지면 해결하기 어려울 수 있습니다.
- `PuLP`와 같은 전문 최적화 라이브러리가 필요합니다.

### 코드 예시 (`PuLP` 사용)

MILP 역시 `PuLP`를 사용하여 모델링할 때 `LpVariable`의 `cat` 인자를 `LpContinuous`와 `LpInteger` (또는 `LpBinary`)를 혼합하여 사용하는 것이 핵심입니다.

```
from pulp import *

# 예시: 생산 및 판매 계획 문제
# 제품 A, B를 생산하여 판매. 제품 A는 연속적으로 생산 가능, 제품 B는 묶음 단위(정수)로
만 생산 가능.
# 제품 A: 생산 시간 1시간, 재료 2단위, 이익 3만원
# 제품 B: 생산 시간 2시간, 재료 1단위, 이익 2만원
# 제약 조건: 총 생산 시간 8시간, 총 재료 6단위
```

```
# 문제 정의 (최대화 문제)
prob = LpProblem("Mixed Production Problem", LpMaximize)

# 결정 변수 정의
x_A = LpVariable("x_A", 0, None, LpContinuous) # 제품 A 생산량 (연속)
x_B = LpVariable("x_B", 0, None, LpInteger)   # 제품 B 생산량 (정수)

# 목적 함수 (총 이익 최대화)
prob += 3 * x_A + 2 * x_B, "Total Profit"

# 제약 조건
prob += 1 * x_A + 2 * x_B <= 8, "Time Constraint" # 생산 시간
prob += 2 * x_A + 1 * x_B <= 6, "Material Constraint" # 재료

# 문제 풀이
prob.solve()

print("--- 혼합 정수 선형 계획법 결과 ---")
print(f"상태: {LpStatus[prob.status]}")

if LpStatus[prob.status] == "Optimal":
    print(f"최적 생산량:")
    for v in prob.variables():
        print(f"{v.name} = {v.varValue}")
    print(f"최대 총 이익: {value(prob.objective)}만원")
else:
    print("최적해를 찾지 못했습니다.")
'''
상태: Optimal
최적 생산량:
x_A = 1.5
x_B = 3.0
최대 총 이익: 10.5만원
'''
```

### 결과 해석 방법

- ILP와 동일하게 `LpStatus[prob.status]`, `prob.variables()`, `value(prob.objective)`를 통해 결과를 해석합니다.

## 장단점 및 대안

| 방법 | 장점 | 단점 |
|---|---|---|
| **선형 계획법 (LP)** | - 효율적으로 해결 가능<br>- 다양한 문제에 적용 가능<br>- 해석이 용이 | - 비선형 관계 모델링 불가<br>- 이산적인 결정 모델링 불가 |

| 방법 | 장점 | 단점 |
| --- | --- | --- |
| **정수 선형 계획법 (ILP)** | - 이산적인 결정(예: 예/아니오, 개수) 모델링 가능<br>- 실제 문제에 더 가깝게 모델링 가능 | - 계산 비용이 매우 높음 (NP-hard)<br>- 대규모 문제 해결 어려움 |
| **혼합 정수 선형 계획법 (MILP)** | - 연속적인 결정과 이산적인 결정을 동시에 모델링 가능<br>- 실제 세계의 복잡한 문제에 가장 적합 | - ILP와 유사하게 계산 비용이 매우 높음<br>- 전문 최적화 솔버 필요 |

대안

- **비선형 계획법 (Non-linear Programming, NLP)**: 목적 함수나 제약 조건 중 하나 이상이 비선형인 경우에 사용됩니다. `scipy.optimize` 모듈에 다양한 비선형 최적화 함수가 있습니다.
- **휴리스틱 및 메타휴리스틱 (Heuristics & Metaheuristics)**: 유전 알고리즘(Genetic Algorithm), 시뮬레이티드 어닐링(Simulated Annealing) 등 복잡하고 대규모의 최적화 문제에서 근사해를 효율적으로 찾는 데 사용됩니다.
- **동적 계획법 (Dynamic Programming)**: 최적화 문제를 여러 개의 작은 하위 문제로 나누어 해결하는 방법으로, 중복되는 하위 문제의 해를 저장하여 효율성을 높입니다.