Naval Architecture & Ocean Engineering

Computer Aided Ship design

-Part I. Optimal Ship Design-

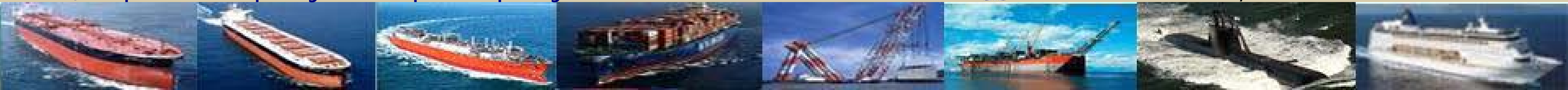
September, 2009

Prof. Kyu-Yeul Lee

**Department of Naval Architecture and Ocean Engineering,
Seoul National University of College of Engineering**

Seoul National Univ.

@SDAL
Advanced Ship Design Automation Lab.
http://asdal.snu.ac.kr

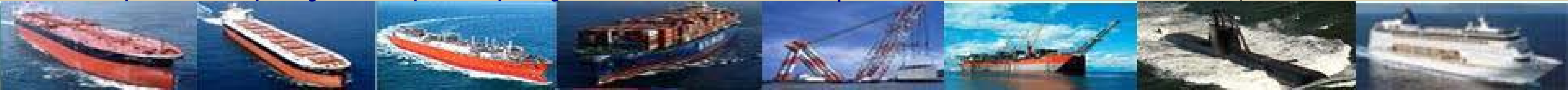


PA#4 SQP 프로그래밍 가이드

2009.10.15

서울대학교 조선해양공학과
선박설계자동화연구실

Seoul National Univ.
@SDAL Advanced Ship Design Automation Lab.
http://asdal.snu.ac.kr



SQP Programming 과제

- 다음의 비선형 제약 최적화 문제 예시 1~4에 대해 CSD 방법으로 최적해를 구하는 프로그램을 작성한다.
  - 비선형 최적화 문제 예시 1~5: 4~8쪽 참고

- 채점 기준
  - 2차 계획 문제를 이용한 Simplex Table 구성: 20점
  - Simplex 방법을 이용한 탐색 방향 결정: 20점
  - 강하함수와 황금분할법을 이용한 탐색 거리 결정: 20점
  - 각 예제 별로 최적해 계산
    - 예제 1, 2, 3, 4: 각 5점 (총 20점)
    - 예제 5: 시작점을 3개 이상으로 하여 Local minimum과 Globlal minimum을 찾을 것 (20점)

  - 유의사항
    - 프로그램 실행 시 5개의 예제를 선택할 수 있도록 함
    - SQP Algorithm 수행 횟수와 x, 그리고 그 때의 목적 함수 값을 출력해야 함
    - Copy 시 0점

비선형 제약 최적화 프로그램을 이용한 최적 설계 예 (1)

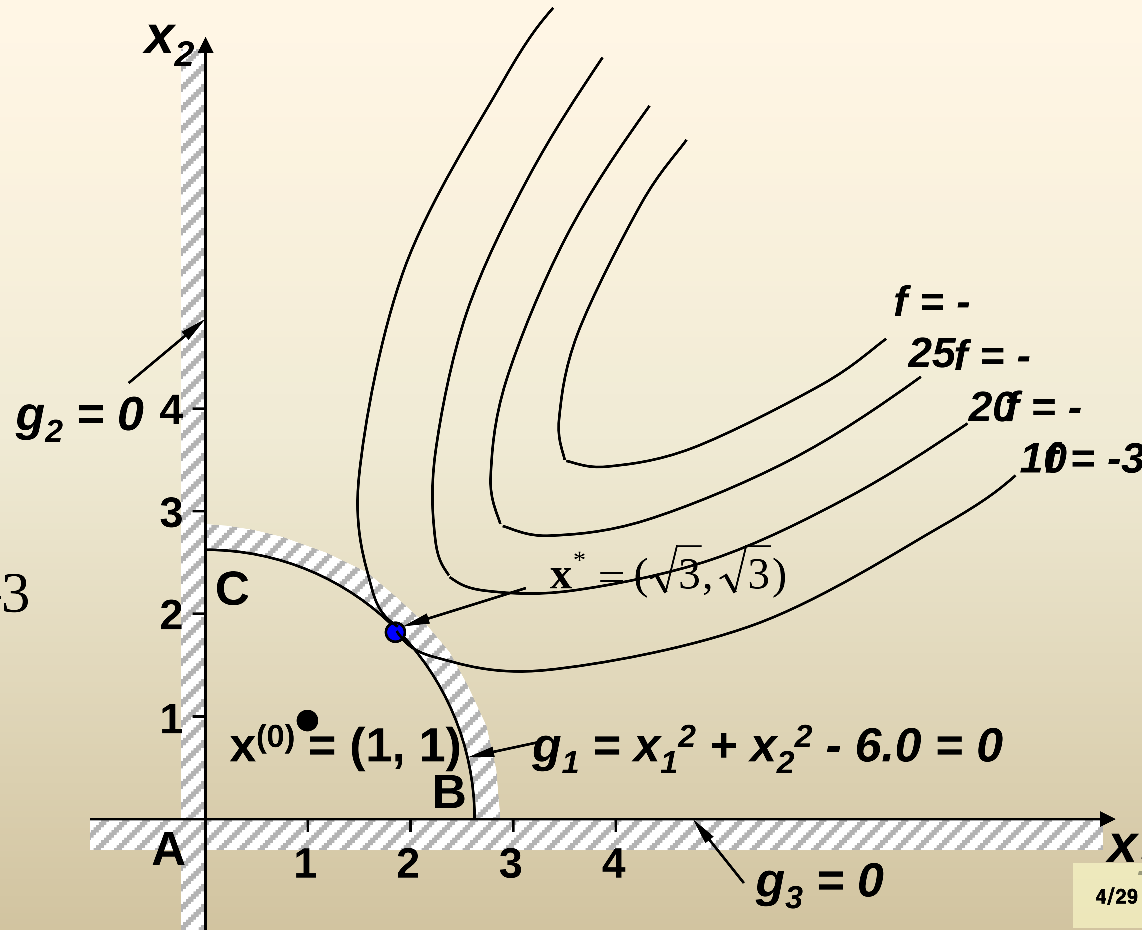
Minimize $f(\mathbf{x}) = x_1^2 + x_2^2 - 3x_1x_2$

Subject to $g_1(\mathbf{x}) = \frac{1}{6}x_1^2 + \frac{1}{6}x_2^2 - 1.0 \le 0$

$$g_2(\mathbf{x}) = -x_1 \le 0$$

$$g_3(\mathbf{x}) = -x_1 \le 0$$

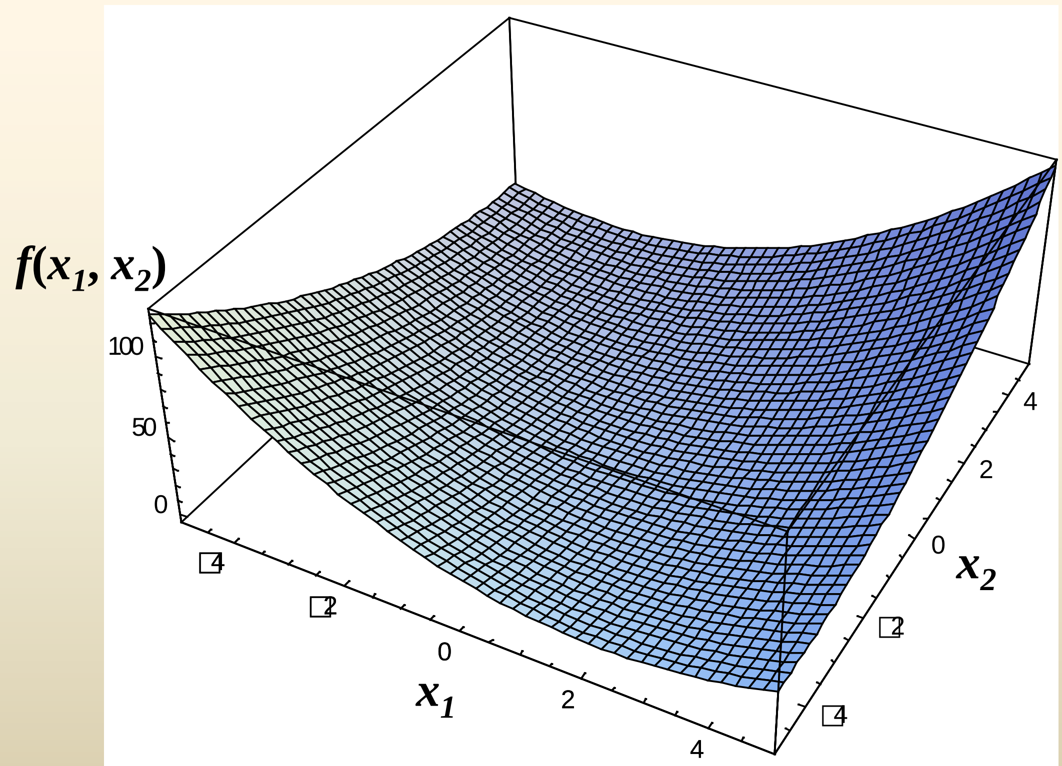
최적해는 $\mathbf{x}^* = (\sqrt{3}, \sqrt{3}),\ f(\mathbf{x}^*) = -3$



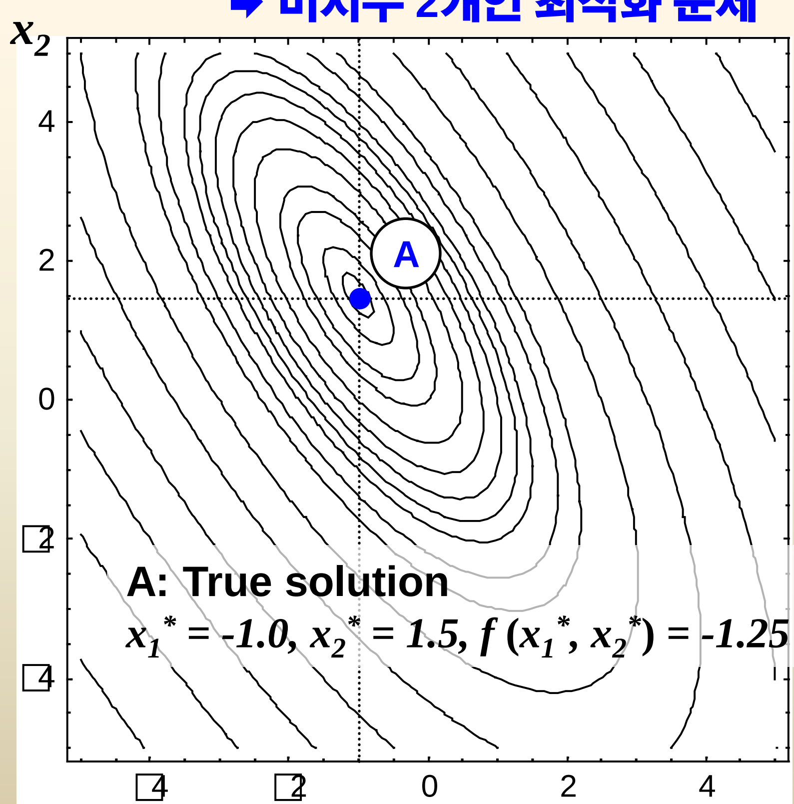
비선형 제약 최적화 프로그램을 이용한 최적 설계 예 (2)

*제약조건이 없는 경우의 문제도 잘 풀리는지 확인하기 위함

$$\textit{Minimize} \quad f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$$



➡ 미지수 2개인 최적화 문제



비선형 제약 최적화 프로그램을 이용한 최적 설계 예 (3)

Minimize

$$f(x_1, x_2) = -\left[25 - (x_1 - 5)^2 - (x_2 - 5)^2\right]$$

Subject to

$$g_1(x_1, x_2) = -32 + 4x_1 + {x_2}^2 \le 0$$

$$g_2(x_1, x_2) = -x_1 \le 0$$

$$g_3(x_1, x_2) = x_1 \le 10$$

$$g_4(x_1, x_2) = -x_2 \le 0$$

$$g_5(x_1, x_2) = x_2 \le 10$$

Solution

$$x_1^* = 4.374, x_2^* = 3.808, f(x_1^*, x_2^*) = -23.188$$

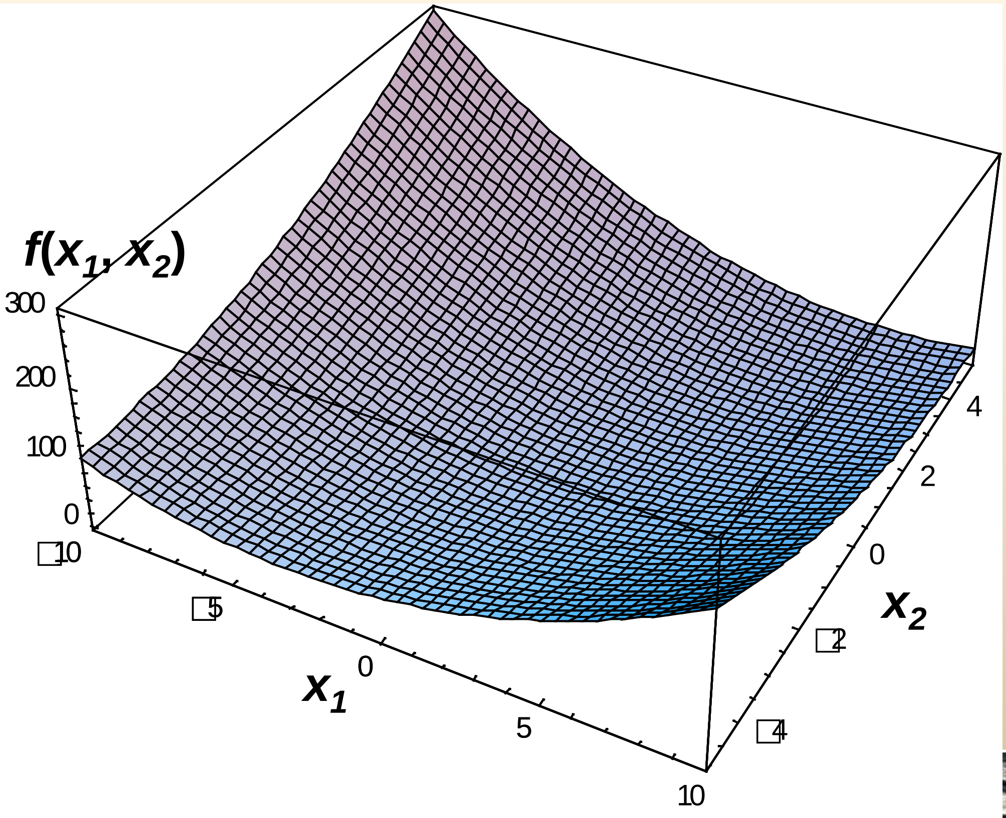
비선형 제약 최적화 프로그램을 이용한 최적 설계 예 (4)

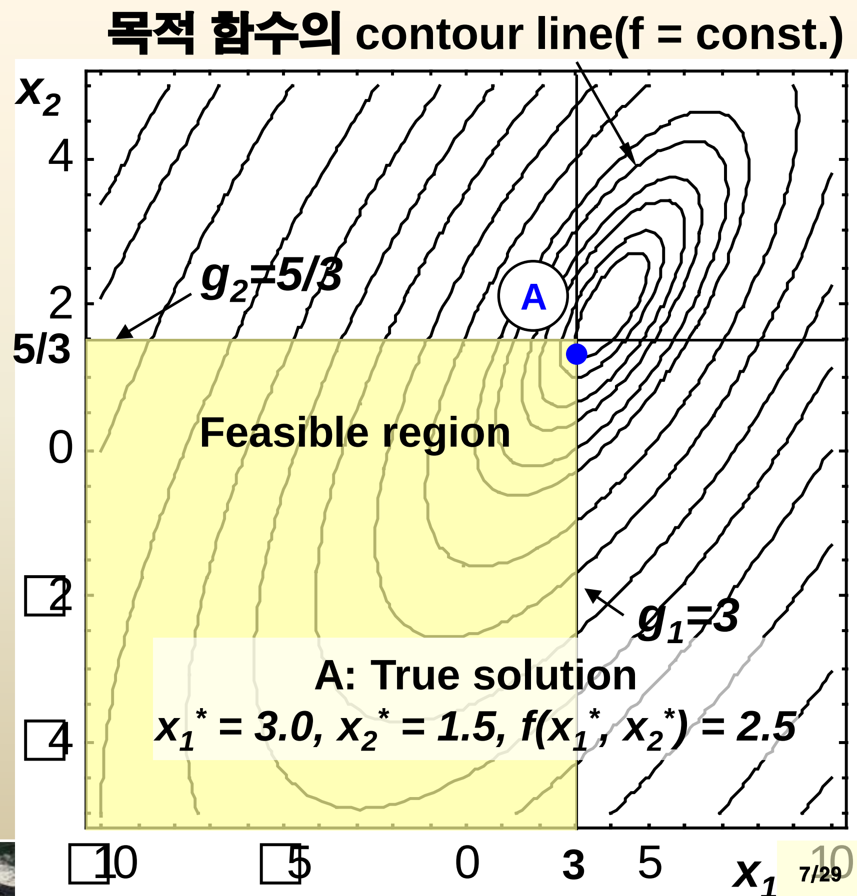
Find $x_1(=B/T), x_2(=1/C_B)$

Minimize $f(x_1, x_2) = x_1^2 + 2x_2^2 - 4x_1 - 2x_1x_2 + 10$

Subject to $g_1(x_1, x_2) = x_1 - 3 \le 0$ ➡ **미지수 2개, 부등호 제약 조건 2개인 최적화 문제**

$g_2(x_1, x_2) = x_2 - 5/3 \le 0$





비선형 제약 최적화 프로그램을 이용한 최적 설계 예 (5)

Goldstein-Price Function

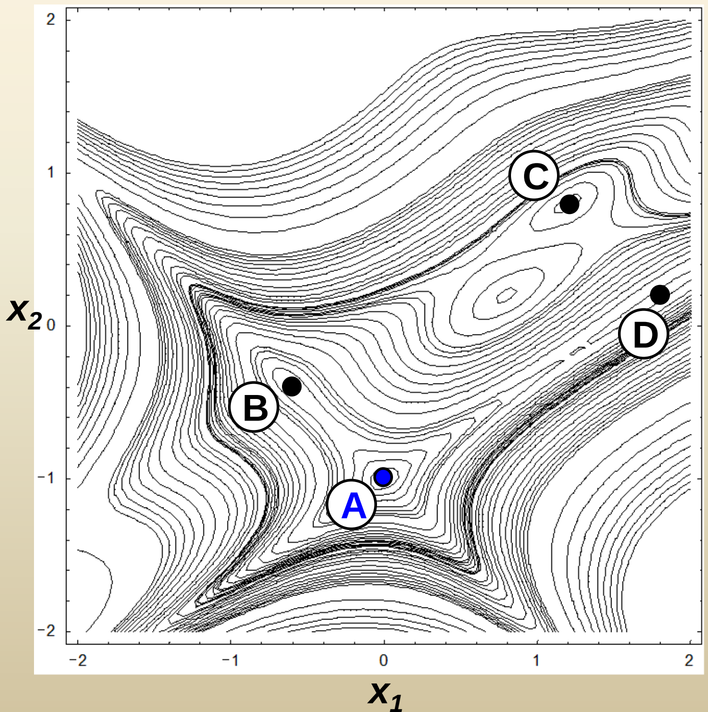
Minimize

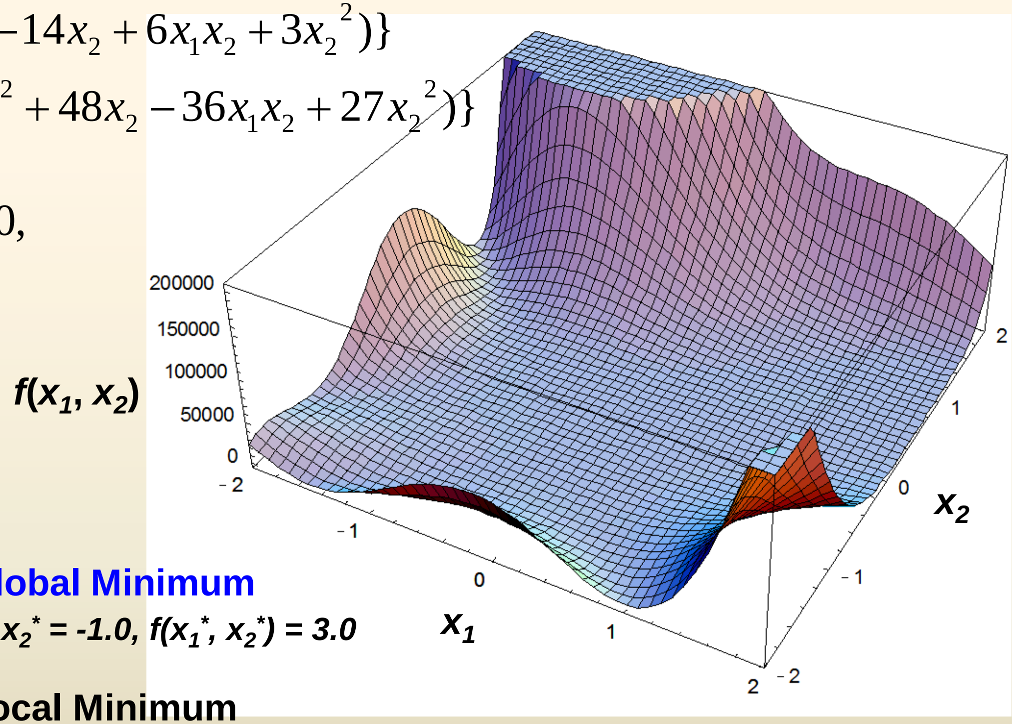
$$f(x_1, x_2) = \{1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\} \cdot \{30 + (2x_1 - 3x_2)^2 \cdot (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\}$$

Subject to

$$g_1(x_1, x_2) = -2 - x_1 \le 0,\ g_2(x_1, x_2) = -2 - x_2 \le 0,$$

$$g_3(x_1, x_2) = x_1 - 2 \le 0,\ g_4(x_1, x_2) = x_2 - 2 \le 0$$





A : Global Minimum

$x_1^* = 0.0,\ x_2^* = -1.0,\ f(x_1^*, x_2^*) = 3.0$

B : Local Minimum

$x_1^* = -0.6,\ x_2^* = -0.4,\ f(x_1^*, x_2^*) = 30.0$

C : Local Minimum

$x_1^* = 1.2,\ x_2^* = 0.8,\ f(x_1^*, x_2^*) = 840.0$

D : Local Minimum

$x_1^* = 1.8,\ x_2^* = 0.2,\ f(x_1^*, x_2^*) = 84.0$

SQP 알고리즘의 Flow Diagram

주어진 설계점 $x^{(k)}$에 대해
탐색 방향 $d^{(k)}$ 및 Lagrange multiplier를 구하기
위한 2차 계획 문제를 구성함

Lagrange 함수 및 K-T 필요 조건을 이용해
2차 계획 문제의 해(탐색 방향 $d^{(k)}$ 및 Lagrange
multiplier)를 구함

2차 계획 문제를 반복적으로 품 (Sequential Quadratic Programming)

탐색 방향의 크기와 최대 위반 제약에 대한
수렴 조건의 만족 여부를 검사함
If $\left|d^{(k)}\right| \le \varepsilon_2$ and $V_k \le \varepsilon_1$

Yes

$x^* = x^{(k)}$라
두고 종료

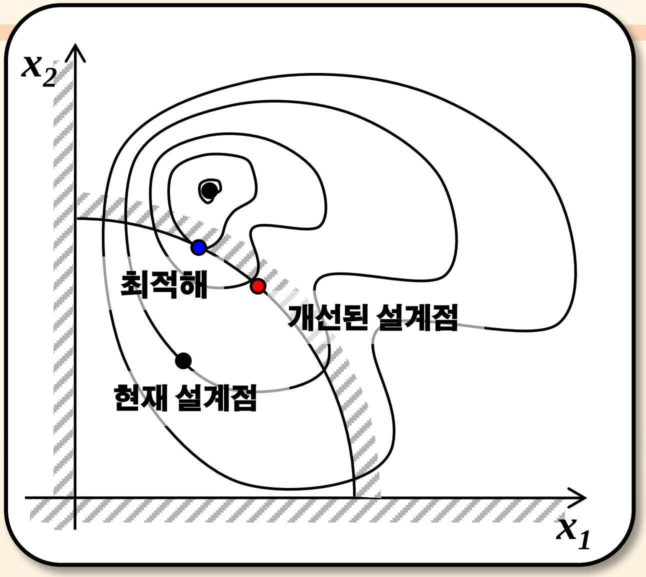
No

1차원 탐색법(예: 황금 분할법)을 사용하여 탐색 방향 ($d^{(k)}$)으로 강하 함수를 최소화 하는 개선된 설계점 ($x^{(k+1)}$)구함

Yes

$k = k + 1$

SQP 알고리즘 요약

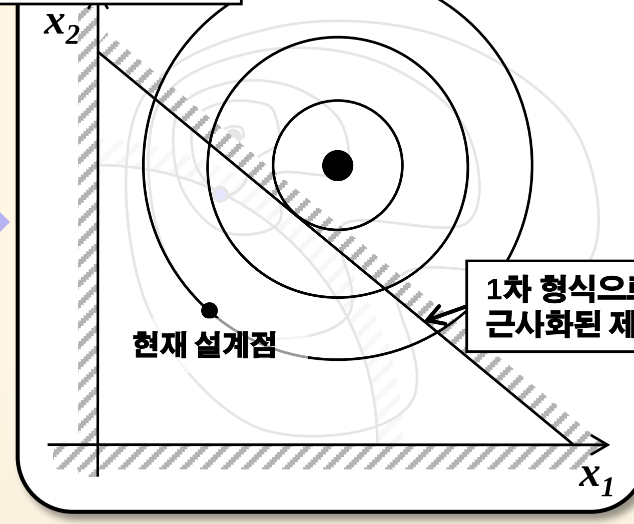


과정 1

현재 설계점에서
2차 계획 문제(QP)로
근사화 한다.

2차 형식으로 근사화된 목적 함수
CSD에서는 2차 미분 값에 해당하는 Hessian Matrix를 Identity Matrix로 가정하기 때문에 목적 함수가 2변수 함수라면 동심원 모양으로 근사화 된다.

2차 계획 문제의 정의
- 목적 함수: 2차 형식
- 제약 조건: 1차 형식

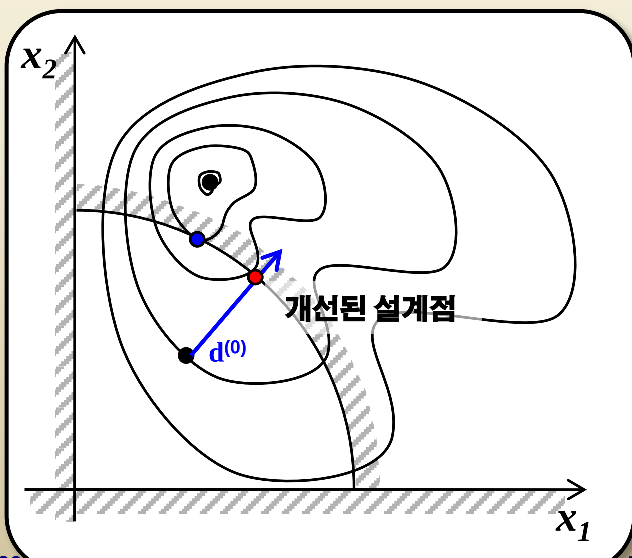


1차 형식으로 근사화된 제약 조건

과정 2

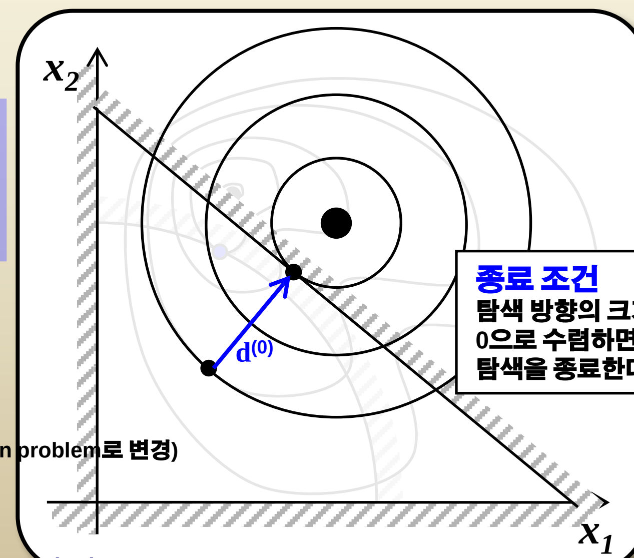
2차 계획 문제(QP)를 풀어서
탐색 방향($d^{(0)}$)을 찾는다.

개선된 설계점에서 부터
과정 1을 다시 수행한다.



과정 3

Penalty Function을 정의한 후 탐색 방향으로 1차원 탐색을 수행하여 탐색 거리를 결정한다.



종료 조건
탐색 방향의 크기 $|d^{(0)}|$가 0으로 수렴하면 탐색을 종료한다.

- Penalty Function:
제약 조건의 위배량을 원래 목적 함수에 더한 수정된 목적 함수
(제약 최적화 문제를 Unconstrained optimization problem로 변경)

- 1차원 탐색의 예: 황금 분할법

SQP 알고리즘의 요약

- 단계 1: k=0으로 둔다. $x^{(0)}$으로 설계 변수의 초기값을 추정한다. 벌칙 매개 변수 R_0,
허용되는 제약 조건의 위배 정도와 수렴 기준으로 작은 수 ε_1, ε_2의 적절한 초기값을 선정한다.

- 단계 2: $x^{(k)}$에서 목적 함수, 제약 조건과 이들의 경사도 (gradient)를 계산한다. 또한 최대 위배 제약 조건 V_k를 계산한다.

- 단계 3: 목적 함수, 제약 조건과 이들이 경사도를 이용하여 2 차 계획 문제를 정의하고, 이를 풀어 탐색 방향 $d^{(k)}$(= $x^{(k+1)}$ - $x^{(k)}$)와 Lagrange multiplier $v^{(k)}$, $u^{(k)}$를 구한다.

SQP 알고리즘의 요약

- 단계 4: 수렴 기준 $|d^{(k)}| \le \varepsilon_2$을 만족하는지 확인한다. 그리고 최대 위반 제약 조건 $V_k \le \varepsilon_1$을 확인한다. 만일 수렴 기준을 만족하면 현재의 $x^{(k)}$가 최적해라고 가정하고 종료한다. 그렇지 않다면 다음 단계로 간다.

- 단계 5: Lagrange multiplier의 합 r_k를 계산하여 R = max{R_k, r_k}로 둔다.

- 단계 6: 새로운 설계 변수 $x^{(k,,j)} = x^{(k)} + \alpha_{(k,j)} d^{(k)}$로 둔다. 여기서 $\alpha = \alpha_{(k,j)}$는 적절한 이동 거리이다. 이동 거리는 탐색 방향 $d^{(k)}$를 따라 제약 조건의 위배량을 원래 목적 함수에 더한 수정된 목적 함수(강하 함수, descent function)를 최소화 하여 구한다. 1차원 탐색 방법을 이동 거리를 결정하는 데 이용할 수 있다.
(1차원 탐색이 끝나면 최종 결정된 $x^{(k,,j)}$ 를 $x^{(k,+1)}$ 로 변경한다.)

- 단계 7: 현재의 벌칙 매개 변수를 R_k = R로 저장한다. 반복 횟수를 $k = k+1$로 수정하고 단계 2로 간다.

제약 최적화 문제의 해결을 위한 SQP Class의 구현 예

- Part 1: Simplex 방법
  - Phase I/Phase II

- Part 2: QP(Quadratic Programming)
  - 주어진 문제의 목적 함수식을 2차 형식으로, 제약 조건식들을 1차 형식으로 근사화 하여 탐색 방향을 구함
  - Kuhn-Tucker 필요 조건을 이용하여, 선형 및 비선형 방정식을 구성하여 이를 풀어 탐색 방향을 구함
    - 선형 부정 방정식: Simplex 방법을 이용하여 해를 구함
    - 비선형 부정 방정식: 선형 부정 방정식으로부터 구한 해가 이 비선형 부정 방정식을 만족하는지 확인하여 해를 확정한다.

- Part 3: SQP 방법
  - SQP(Sequential Quadratic Programming): QP를 연속적(Sequential)으로 풀어 현재의 설계점에서의 탐색 방향을 구함
  - 원래의 목적 함수식과 제약 조건식들로부터 강하 함수를 구성하고 이를 최소화 하는 이동 거리를 구함(1차원 탐색 방법을 사용, 예: 황금분할법)

- Simplex 방법을 이용한 2차 계획 문제의 풀이 방법 (1)

Kuhn-Tucker 필요 조건: $\nabla L = \mathbf{0}$

$$\begin{cases} \dfrac{\partial L}{\partial \mathbf{d}_{(n\times1)}} = \mathbf{c}_{(n\times1)} + \mathbf{H}_{(n\times n)}\mathbf{d}_{(n\times1)} + \mathbf{A}_{(n\times m)}\mathbf{u}_{(m\times1)} + \mathbf{N}_{(n\times p)}\mathbf{v}_{(p\times1)} = \mathbf{0} \\ \dfrac{\partial L}{\partial \mathbf{u}_{(m\times1)}} = \mathbf{A}^T{}_{(m\times n)}\mathbf{d}_{(n\times1)} + \mathbf{s}_{(m\times1)} - \mathbf{b}_{(m\times1)} = \mathbf{0} \\ \dfrac{\partial L}{\partial \mathbf{v}_{(p\times1)}} = \mathbf{N}^T{}_{(p\times n)}\mathbf{d}_{(n\times1)} - \mathbf{e}_{(p\times1)} = \mathbf{0} \end{cases}$$

여기서, $u_i, s_i \geq 0;\ i = 1\, to\, m$

$\{ u_i s_i = 0; i = 1\, to\, m$

→ 선형 부정 방정식으로부터 구한 해가 이 비선형 부정 방정식을 만족하는지 확인하여 해를 확정한다.

→ 이 식들은 설계 변수 **d**에 대해 모두 선형이므로,
이 식들로부터 설계 변수 **d**를 구하는 문제는 등호 제약 조건만으로 이루어진 선형 계획 문제임

설계 변수($\mathbf{d}^{(0)}$)가 부호에 제한이 없기 때문에 문제를 풀기 전에 먼저 설계 변수를 아래와 같이 변환해야 함

$$\mathbf{d}_{(n\times1)} = \mathbf{d}_{(n\times1)}{}^{+} - \mathbf{d}_{(n\times1)}{}^{-}$$

여기서, $\mathbf{d}_{(n\times1)}{}^{+},\ \mathbf{d}_{(n\times1)}{}^{-} \geq 0$

등호 제약 조건에 대한 **Lagrange multiplier** $\mathbf{v}_{(p\times1)}$도 부호의 제한이 없으므로 다음과 같이 변환해야 함

$$\mathbf{v}_{(p\times1)} = \mathbf{y}_{(p\times1)} - \mathbf{z}_{(p\times1)}$$

SQP Programming Guide
- Simplex 방법을 이용한 2차 계획 문제의 풀이 방법 (2)

Kuhn-Tucker 필요 조건: $\nabla L = \mathbf{0}$

$$\frac{\partial L}{\partial \mathbf{d}_{(n\times1)}} = \mathbf{c}_{(n\times1)} + \mathbf{H}_{(n\times n)}\mathbf{d}_{(n\times1)} + \mathbf{A}_{(n\times m)}\mathbf{u}_{(m\times1)} + \mathbf{N}_{(n\times p)}\mathbf{v}_{(p\times1)} = \mathbf{0}$$

$$\frac{\partial L}{\partial \mathbf{u}_{(m\times1)}} = \mathbf{A}^T{}_{(m\times n)}\mathbf{d}_{(n\times1)} + \mathbf{s}_{(m\times1)} - \mathbf{b}_{(m\times1)} = \mathbf{0} \qquad u_i s_i = 0; i = 1\, to\, m$$

$$\frac{\partial L}{\partial \mathbf{v}_{(p\times1)}} = \mathbf{N}^T{}_{(p\times n)}\mathbf{d}_{(n\times1)} - \mathbf{e}_{(p\times1)} = \mathbf{0}$$

여기서, $u_i, s_i \geq 0;\ i = 1\, to\, m$

Kuhn-Tucker 필요 조건(행렬식 표현)

$$\underbrace{\begin{bmatrix} \mathbf{H}_{(n\times n)} & -\mathbf{H}_{(n\times n)} & \mathbf{A}_{(n\times m)} & \mathbf{0}_{(n\times m)} & \mathbf{N}_{(n\times p)} & -\mathbf{N}_{(n\times p)} \\ \mathbf{A}^T{}_{(m\times n)} & -\mathbf{A}^T{}_{(m\times n)} & \mathbf{0}_{(m\times m)} & \mathbf{I}_{(m\times m)} & \mathbf{0}_{(m\times p)} & \mathbf{0}_{(m\times p)} \\ \mathbf{N}^T{}_{(p\times n)} & -\mathbf{N}^T{}_{(p\times n)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times p)} & \mathbf{0}_{(p\times p)} \end{bmatrix}}_{= \mathbf{B}_{((n+m+p)\times(2n+2m+2p))}} \underbrace{\begin{bmatrix} \mathbf{d}^+{}_{(n\times1)} \\ \mathbf{d}^-{}_{(n\times1)} \\ \mathbf{u}_{(m\times1)} \\ \mathbf{s}_{(m\times1)} \\ \mathbf{y}_{(p\times1)} \\ \mathbf{z}_{(p\times1)} \end{bmatrix}}_{= \mathbf{X}_{((2n+2m+2p)\times1)}} = \underbrace{\begin{bmatrix} -\mathbf{c}_{(n\times1)} \\ \mathbf{b}_{(m\times1)} \\ \mathbf{e}_{(p\times1)} \end{bmatrix}}_{= \mathbf{D}_{((n+m+p)\times1)}}$$

$$u_i s_i = 0; i = 1\, to\, m$$

$$\mathbf{B}_{((n+m+p)\times(2n+2m+2p))}\mathbf{X}_{((2n+2m+2p)\times1)} = \mathbf{D}_{((n+m+p)\times1)}$$

SQP Programming Guide
- SQP 방법 요약

① 목적함수는 2차 형식, 제약 조건은 1차형식으로 근사화 (2차 계획 문제로 근사화)

② Lagrange 함수와 쿤-터커 조건을 사용하여 행렬로 표현

$$\begin{bmatrix} \mathbf{H}_{(n\times n)} & -\mathbf{H}_{(n\times n)} & \mathbf{A}_{(n\times m)} & \mathbf{0}_{(n\times m)} & \mathbf{N}_{(n\times p)} & -\mathbf{N}_{(n\times p)} \\ \mathbf{A}^T_{(m\times n)} & -\mathbf{A}^T_{(m\times n)} & \mathbf{0}_{(m\times m)} & \mathbf{I}_{(m\times m)} & \mathbf{0}_{(m\times p)} & \mathbf{0}_{(m\times p)} \\ \mathbf{N}^T_{(p\times n)} & -\mathbf{N}^T_{(p\times n)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times p)} & \mathbf{0}_{(p\times p)} \end{bmatrix} \begin{bmatrix} \mathbf{d}^+_{(n\times 1)} \\ \mathbf{d}^-_{(n\times 1)} \\ \mathbf{u}_{(m\times 1)} \\ \mathbf{s}_{(m\times 1)} \\ \mathbf{y}_{(p\times 1)} \\ \mathbf{z}_{(p\times 1)} \end{bmatrix} = \begin{bmatrix} -\mathbf{c}_{(n\times 1)} \\ \mathbf{b}_{(m\times 1)} \\ \mathbf{e}_{(p\times 1)} \end{bmatrix}$$

Minimize $\bar{f} = \mathbf{c}^T{}_{(1\times n)}\mathbf{d}_{(n\times 1)} + \frac{1}{2}\mathbf{d}^T{}_{(1\times n)}\mathbf{d}_{(n\times 1)}$

Subject to $\mathbf{N}^T{}_{(p\times n)}\mathbf{d}_{(n\times 1)} = \mathbf{e}_{(p\times 1)}$

$\mathbf{A}^T{}_{(m\times n)}\mathbf{d}_{(n\times 1)} \le \mathbf{b}_{(m\times 1)}$

③ Simplex Method를 사용하여 탐색방향 찾음

1

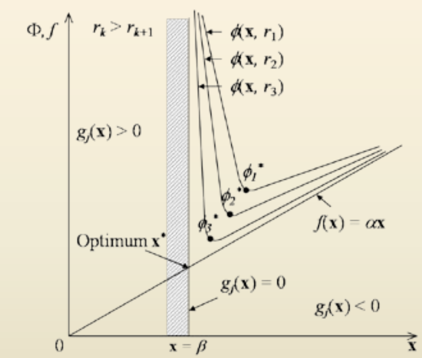
	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y1	Y2	Y3	Y4	Y5	bi	bi/ai
Y1	1	0	-1	0	1/3	-1	0	0	0	0	1	0	0	0	0	1	-
Y2	0	1	0	-1	1/3	0	-1	0	0	0	0	1	0	0	0	1	-
X8	1/3	1/3	-1/3	-1/3	0	0	0	1	0	0	0	0	1	0	0	2/3	-
Y4	-1	0	1	0	0	0	0	0	**1**	0	0	0	0	1	0	1	**1**
Y5	0	-1	0	1	0	0	0	0	0	1	0	0	0	0	1	1	-
A. Obj.	0	0	0	0	-2/3	1	1	0	**-1**	-1	0	0	1	0	0	w-4	-

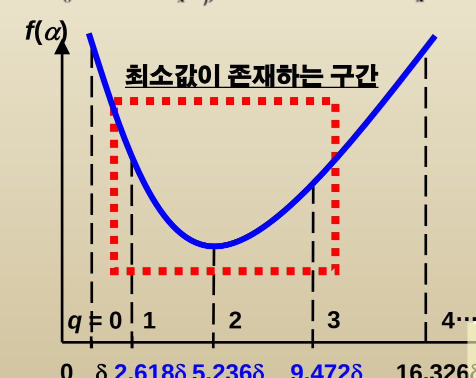
2

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y1	Y2	Y3	Y4	Y5	bi	bi/ai
Y1	1	0	-1	0	1/3	-1	0	0	0	0	1	0	0	0	0	1	-
Y2	0	1	0	-1	1/3	0	-1	0	0	0	0	1	0	0	0	1	-
X8	1/3	1/3	-1/3	-1/3	0	0	0	1	0	0	0	0	1	0	0	2/3	-
X9	-1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1	-
Y5	0	-1	0	1	0	0	0	0	0	**1**	0	0	0	0	1	1	**1**
A. Obj.	-1	0	1	0	-2/3	1	1	0	0	**-1**	0	0	1	1	0	w-3	-

•
•
•

④ 강하 함수를 목적함수로 사용하여 1차원 탐색 방법 (예: 황금 분할법)을 이용하여 탐색 방향으로 이동거리 계산





SQP Programming Guide

- SQP Class 예시

```
#include<vector>
#include "Matrix.h“
#include “Simplex.h“
using namespace std;

typedef double(*FP)(double *x);

class SQP
{
public:
    SQP();              //생성자
    virtual ~SQP();     //소멸자

//Member Variables
    FP m_ObjFn;                   //목적함수
    std::vector<FP> m_EqualFn;    //등호제약조건
    std::vector<FP> m_UnequalFn;  //부등호제약조건

    int NumOfEqual;       //등식의 개수
    int NumOfUnEqual;     //부등식의 개수
    int NumOfVariable;    //미지수의 개수

    double *m_x;              //점의좌표
    double *m_d;              //탐색방향
    double m_MinValue;        //최소값
    Matrix m_SimplexTable;  //쿤-터커 필요조건 행렬
```

```
//Member Function

// 초기 시작 위치와 변수의 개수 입력
    void InitializeSQP(double *_x,int n);
//목적함수, 등호 및 부등호 제약조건 입력
    void AddObjectFunction(double (*f)(double*));
    void AddEqualConstraint(double (*f)(double*));
    void AddUnEqualConstraintFn(double (*f)(double*));

    void Solve();

//쿤-터커 필요조건 행렬식 만듦
    void BuildSimplexTable();
//탐색방향 계산
    void FindSearchDirection();
//End Condition Check
    bool CheckEndCondition();
//탐색 방향으로의 최소값과 최소점 계산
    void FindNextMinPoint();
//강하 함수(Penalty Function)을 정의
    static double PenaltyFunction(double *_x);
};
```

QP문제

1차원 탐색

Penalty function 구성

class 멤버 함수를 함수 포인터로 전달하기 위하여 static으로 선언함

SQP Programming Guide

1) 시작점의 좌표, 목적 함수, 등호/부등호 제약 조건 입력

(Example)

```
double ObjectFn(double *x);
double UnequalConst1(double *x);
double UnequalConst2(double *x);
double EqualConst1(double *x);

int main()
{
    SQP sqp1;

    int n = 2;
    double *x = new double [n];
    x[0]=1;
    x[1]=1;

    sqp1.InitializeSQP(x,n);
    sqp1.AddObjectFunction(ObjectFn);
    sqp1.AddUnequalConstraint(UnequalConst1);
    sqp1.AddUnequalConstraint(UnequalConst2);
    sqp1.AddEqualConstraint(EqualConst1);
    sqp1.Solve();

    delete[] x;
}
```

변수 개수 2개

시작 위치 설정 (1,1)

SQP 초기화

목적함수, 등호 및 부등호 제약조건 입력

SQP Programming Guide

2) Solve 함수의 동작

```
void SQP::Solve()
{
    while(1)
    {
        BuildSimplexTable();

        FindSearchDirection();

        if( CheckEndCondition() )
            break;

        FindNextMinPoint();
    }
}
```

① 목적함수는 2차, 제약 조건은 1차로 근사화 (2차 계획 문제로 표현)

$$\textit{Minimize}\quad \bar{f} = \mathbf{c}^T{}_{(1\times n)}\mathbf{d}_{(n\times 1)} + \frac{1}{2}\mathbf{d}^T{}_{(1\times n)}\mathbf{d}_{(n\times 1)} \quad \textit{Subject to} \quad \begin{aligned} \mathbf{N}^T{}_{(p\times n)}\mathbf{d}_{(n\times 1)} &= \mathbf{e}_{(p\times 1)} \\ \mathbf{A}^T{}_{(m\times n)}\mathbf{d}_{(n\times 1)} &\le \mathbf{b}_{(m\times 1)} \end{aligned}$$

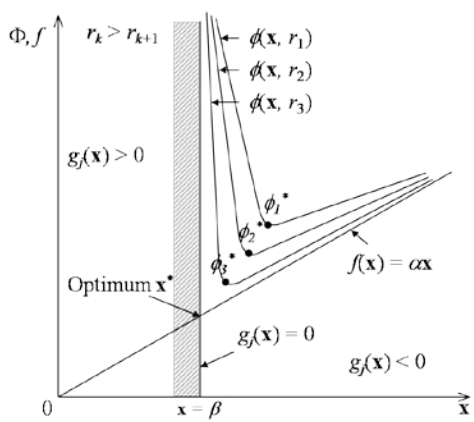
② Lagrange 함수와 쿤-터커 조건을 사용하여 행렬로 표현

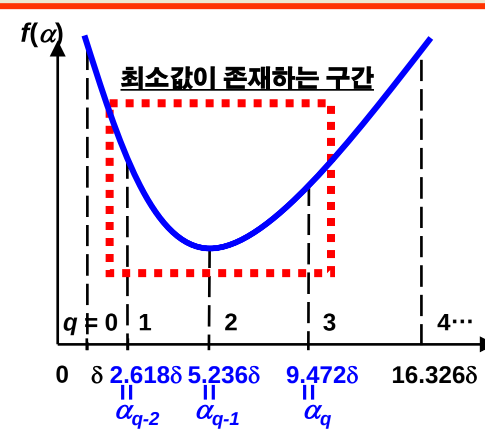
$$\begin{bmatrix} \mathbf{H}_{(n\times n)} & -\mathbf{H}_{(n\times n)} & \mathbf{A}_{(n\times m)} & \mathbf{0}_{(n\times m)} & \mathbf{N}_{(n\times p)} & -\mathbf{N}_{(n\times p)} \\ \mathbf{A}^T{}_{(m\times n)} & -\mathbf{A}^T{}_{(m\times n)} & \mathbf{0}_{(m\times m)} & \mathbf{I}_{(m\times m)} & \mathbf{0}_{(m\times p)} & \mathbf{0}_{(m\times p)} \\ \mathbf{N}^T{}_{(p\times n)} & -\mathbf{N}^T{}_{(p\times n)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times p)} & \mathbf{0}_{(p\times p)} \end{bmatrix} \begin{bmatrix} \mathbf{d}^+{}_{(n\times 1)} \\ \mathbf{d}^-{}_{(n\times 1)} \\ \mathbf{u}_{(m\times 1)} \\ \mathbf{s}_{(m\times 1)} \\ \mathbf{y}_{(p\times 1)} \\ \mathbf{z}_{(p\times 1)} \end{bmatrix} = \begin{bmatrix} -\mathbf{c}_{(n\times 1)} \\ \mathbf{b}_{(m\times 1)} \\ \mathbf{e}_{(p\times 1)} \end{bmatrix}$$

③ Simplex Method를 사용하여 탐색방향 찾음

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	Y1	Y2	Y3	Y4	Y5	bi	bi/ai
Y1	1	0	-1	0	1/3	-1	0	0	0	0	1	0	0	0	0	1	-
Y2	0	1	0	-1	1/3	0	-1	0	0	0	0	1	0	0	0	1	-
X8	1/3	1/3	-1/3	-1/3	0	0	0	1	0	0	0	0	1	0	0	2/3	-
Y4	-1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1
Y5	0	-1	0	1	0	0	0	0	0	1	0	0	0	0	1	1	-
A. Obj.	0	0	0	0	-2/3	1	1	0	-1	-1	0	0	1	0	0	w-4	-

④ 강하 함수를 목적함수로 사용하여 1차원 탐색 방법 (예: 황금 분할법)을 이용하여 탐색 방향으로 이동거리 계산





n : 변수의 개수(m_NumOfVariable)

m : 부등호 제약조건 식의 개수(m_NumOfUnequal)

p : 등호 제약조건 식의 개수(m_NumOfEqual)

Simplex Table 구성: BuildSimplexTable 함수 구현

주어진 목적함수와 제약 조건 식으로부터 다음 매트릭스를 구성해야 함

$$\begin{bmatrix} \mathbf{H}_{(n\times n)} & -\mathbf{H}_{(n\times n)} & \mathbf{A}_{(n\times m)} & \mathbf{0}_{(n\times m)} & \mathbf{N}_{(n\times p)} & -\mathbf{N}_{(n\times p)} \\ \mathbf{A}^T_{(m\times n)} & -\mathbf{A}^T_{(m\times n)} & \mathbf{0}_{(m\times m)} & \mathbf{I}_{(m\times m)} & \mathbf{0}_{(m\times p)} & \mathbf{0}_{(m\times p)} \\ \mathbf{N}^T_{(p\times n)} & -\mathbf{N}^T_{(p\times n)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times p)} & \mathbf{0}_{(p\times p)} \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix}\mathbf{d}^+_{(n\times1)} \\ \mathbf{d}^-_{(n\times1)}\end{matrix}} \\ \mathbf{u}_{(m\times1)} \\ \mathbf{s}_{(m\times1)} \\ \mathbf{y}_{(p\times1)} \\ \mathbf{z}_{(p\times1)} \end{bmatrix} = \begin{bmatrix} -\mathbf{c}_{(n\times1)} \\ \mathbf{b}_{(m\times1)} \\ \mathbf{e}_{(p\times1)} \end{bmatrix}$$

탐색 방향 (부호 제약이 없으므로 두 개로 분리)

Lagrange multipliers

완화 변수

$$= \mathbf{B}_{((n+m+p)\times(2n+2m+2p))}$$

문제로부터 구할 수 있는 것

$$= \mathbf{X}_{((2n+2m+2p)\times1)}$$

구하려고 하는 것

$$= \mathbf{D}_{((n+m+p)\times1)}$$

문제로부터 구할 수 있는 것

$$\mathbf{B}_{((n+m+p)\times(2n+2m+2p))}\mathbf{X}_{((2n+2m+2p)\times1)} = \mathbf{D}_{((n+m+p)\times1)}$$

목적 함수의 Gradient Vector

$$\mathbf{H}_{(n\times n)} = \mathbf{I}_{(n\times n)},\quad \mathbf{c}_{(n\times1)} = \begin{bmatrix} \partial f(\mathbf{x})/\partial x_1 \\ \vdots \\ \partial f(\mathbf{x})/\partial x_n \end{bmatrix},\quad \mathbf{b}_{(m\times1)} = \begin{bmatrix} -g_1(\mathbf{x}) \\ \vdots \\ -g_m(\mathbf{x}) \end{bmatrix},\quad \mathbf{e}_{(p\times1)} = \begin{bmatrix} -h_1(\mathbf{x}) \\ \vdots \\ -h_p(\mathbf{x}) \end{bmatrix}$$

부등호 제약 조건의 Gradient Vector

$$\mathbf{A}_{(n\times m)} = \begin{bmatrix} \partial g_1(\mathbf{x})/\partial x_1 & \cdots & \partial g_m(\mathbf{x})/\partial x_1 \\ \vdots & \vdots & \vdots \\ \partial g_1(\mathbf{x})/\partial x_n & \cdots & \partial g_m(\mathbf{x})/\partial x_n \end{bmatrix},$$

등호 제약 조건의 Gradient Vector

$$\mathbf{N}_{(n\times p)} = \begin{bmatrix} \partial h_1(\mathbf{x})/\partial x_1 & \cdots & \partial h_p(\mathbf{x})/\partial x_1 \\ \vdots & \vdots & \vdots \\ \partial h_1(\mathbf{x})/\partial x_n & \cdots & \partial h_p(\mathbf{x})/\partial x_n \end{bmatrix}$$

3) Simplex Table 구성

Simplex Table 구성: BuildSimplexTable 함수 구현

2변수 함수 $f(\mathbf{x}) = f(x_1, x_2)$ 의 Gradient Vector($\nabla f(\mathbf{x})$)는 다음과 같은 Central Difference Method을 이용하여 수치적으로 계산할 수 있음

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial f(x_1, x_2)}{\partial x_1} \\ \dfrac{\partial f(x_1, x_2)}{\partial x_2} \end{bmatrix}$$

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = \frac{f(x_1 + \Delta x_1, x_2) - f(x_1 - \Delta x_1, x_2)}{(x_1 + \Delta x_1) - (x_1 - \Delta x_1)} = \frac{f(x_1 + \Delta x_1, x_2) - f(x_1 - \Delta x_1, x_2)}{2\Delta x_1}$$

$$\frac{\partial f(x_1, x_2)}{\partial x_2} = \frac{f(x_1, x_2 + \Delta x_2) - f(x_1, x_2 - \Delta x_2)}{(x_2 + \Delta x_2) - (x_2 - \Delta x_2)} = \frac{f(x_1, x_2 + \Delta x_2) - f(x_1, x_2 - \Delta x_2)}{2\Delta x_2}$$

여기서, $\Delta x_1, \Delta x_2$ 은 아주 작은 양수(예, 10^{-6})

n : 변수의 개수(m_NumOfVariable)

m : 부등호 제약조건 식의 개수(m_NumOfUnequal)

p : 등호 제약조건 식의 개수(m_NumOfEqual)

Simplex Table 구성: BuildSimplexTable 함수 구현

주어진 목적함수와 제약 조건 식으로부터 다음 매트릭스를 구성해야 함

$$\begin{bmatrix} \mathbf{H}_{(n\times n)} & -\mathbf{H}_{(n\times n)} & \mathbf{A}_{(n\times m)} & \mathbf{0}_{(n\times m)} & \mathbf{N}_{(n\times p)} & -\mathbf{N}_{(n\times p)} \\ \mathbf{A}^T{}_{(m\times n)} & -\mathbf{A}^T{}_{(m\times n)} & \mathbf{0}_{(m\times m)} & \mathbf{I}_{(m\times m)} & \mathbf{0}_{(m\times p)} & \mathbf{0}_{(m\times p)} \\ \mathbf{N}^T{}_{(p\times n)} & -\mathbf{N}^T{}_{(p\times n)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times p)} & \mathbf{0}_{(p\times p)} \end{bmatrix} \begin{bmatrix} \mathbf{d}^+{}_{(n\times 1)} \\ \mathbf{d}^-{}_{(n\times 1)} \\ \mathbf{u}_{(m\times 1)} \\ \mathbf{s}_{(m\times 1)} \\ \mathbf{y}_{(p\times 1)} \\ \mathbf{z}_{(p\times 1)} \end{bmatrix} = \begin{bmatrix} -\mathbf{c}_{(n\times 1)} \\ \mathbf{b}_{(m\times 1)} \\ \mathbf{e}_{(p\times 1)} \end{bmatrix}$$

$$\mathbf{B}_{((n+m+p)\times(2n+2m+2p))}\mathbf{X}_{((2n+2m+2p)\times 1)} = \mathbf{D}_{((n+m+p)\times 1)}$$

(n+m+p)개의 등식으로만 구성된 연립 일차 방정식 → 미지수의 개수 > 식의 개수

$$\underset{(2n+2m+2p)}{} \qquad \underset{(n+m+p)}{}$$

$(2n+2m+2p)$ $(n+m+p)$

⇩

$$\mathbf{B}_{((n+m+p)\times(2n+2m+2p))}\mathbf{X}_{((2n+2m+2p)\times 1)} + \boxed{\mathbf{Y}_{(n+m+p)}} = \mathbf{D}_{((n+m+p)\times 1)}$$

식의 개수만큼 인위 변수(artificial variable) 추가

⇩

인위 목적 함수(artificial object function) 추가

3) Simplex Table 구성

n : 변수의 개수(m_NumOfVariable)

m : 부등호 제약조건 식의 개수(m_NumOfUnequal)

p : 등호 제약조건 식의 개수(m_NumOfEqual)

Simplex Table 구성: BuildSimplexTable 함수 구현

$$\mathbf{B}_{((n+m+p)\times(2n+2m+2p))}\mathbf{X}_{((2n+2m+2p)\times 1)} + \mathbf{Y}_{(n+m+p)} = \mathbf{D}_{((n+m+p)\times 1)}$$

$$\left[\begin{array}{cccccc|ccc|c}
\mathbf{H}_{(n\times n)} & -\mathbf{H}_{(n\times n)} & \mathbf{A}_{(n\times m)} & \mathbf{0}_{(n\times m)} & \mathbf{N}_{(n\times p)} & -\mathbf{N}_{(n\times p)} & \mathbf{I}_{(n\times n)} & \mathbf{0}_{(n\times m)} & \mathbf{0}_{(n\times p)} & -\mathbf{c}_{(n\times 1)} \\
\mathbf{A}^T{}_{(m\times n)} & -\mathbf{A}^T{}_{(m\times n)} & \mathbf{0}_{(m\times m)} & \mathbf{I}_{(m\times m)} & \mathbf{0}_{(m\times p)} & \mathbf{0}_{(m\times p)} & \mathbf{0}_{(m\times n)} & \mathbf{I}_{(m\times m)} & \mathbf{0}_{(m\times p)} & \mathbf{b}_{(m\times 1)} \\
\mathbf{N}^T{}_{(p\times n)} & -\mathbf{N}^T{}_{(p\times n)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times m)} & \mathbf{0}_{(p\times p)} & \mathbf{0}_{(p\times p)} & \mathbf{0}_{(p\times n)} & \mathbf{0}_{(p\times m)} & \mathbf{I}_{(p\times p)} & \mathbf{e}_{(p\times 1)}
\end{array}\right]$$

Artificial Object Function
(모든 열의 합에 (-)를 붙여 구함)

Matrix
m_SimplexTable;

행의 개수

$(n+m+p)+1$

인위목적함수

등호의 우변

열의 개수 $(3n+3m+3p)+1 = 3(n+m+p)+1$

SQP Programming Guide
3) Simplex Table 구성

Simplex Table 구성: BuildSimplexTable 함수 구현

- 함수 구현 시 주의 사항

✓ **인위 목적 함수는 각 열을 모두 더한 다음 (-)를 붙여주면 된다.**

✓ **인위 목적 함수를 만들기 전에 반드시 가장 마지막 열의 부호가 모두 양수 또는 0인지 확인하고, 음수일 경우 행 전체에 (-)를 곱해 준다.**

✓ **m_SimplexTable을 구성할 때, Matrix class의 SetPartOfMatrix 함수를 사용하면 편리하다.**

4) 탐색 방향을 찾음

탐색 방향을 찾음: FindSearchDirection 함수 구현

variableTable과 biaiTable을 선언

↓

Simplex class의 Solve 함수 실행

↓

모든 $u_i s_i = 0$ 인가?

Yes → Simplex 종료

No → 저장된 Simplex를 불러와 다시 실행

Matrix VariableTable;

Matrix biaiTable;

Matrix m_SimplexTable;

행의 개수 $(n+m+p)+1$

열의 개수 $3(n+m+p)+1$

※ 주의 사항

Pivot 열 또는 행을 찾을 때, 선택할 수 있는 열 또는 행이 여러 개일 수 있다.

이 경우, 선택하지 않은 열 또는 행 정보를 가지는 Simplex를 저장하고 있어야 한다.

4) 탐색 방향을 찾음

탐색 방향을 찾음: FindSearchDirection 함수 구현

(090930)Simplex Programming Guide참고

```
void SQP::FindSearchDirection()
{
    ……
    Simplex mySimplex;
    mySimplex.Initialize(…);

    while(1)
    {
        //인위목적함수가 0이되는 쌍이
        //있으면 true를 return
        //없으면 false를 return
        bool isFound = mySimplex.Solve();

        if(isFound)
        {
            uisi =0 인지 확인후 참이면 종료
        }

        //저장된 Simplex를 하나 불러옴
        mySimplex = …
    }
}
```

```
bool Simplex::Solve()
{
    while(1)
    {
        if (m_pivotColumn == -1)
            FindPivotColumn();
        if (m_pivotColumn == -1)
        {
            return false;
        }
        if (m_pivotRow == -1)
            FindPivotRow();
        if (m_pivotRow == -1)
        {
            return false;
        }
        Pivot();
        if(CheckEndCondition())
            return true;
    }
    return true;
}
```

최소값이 음수가 아님
(모든 계수가 양수)

bi/ai 비율이
최소인 양수가 없음

SQP Programming Guide

4) 탐색 방향을 찾음

탐색 방향을 찾음: FindSearchDirection 함수 구현

- ✓ 반드시 Roll-Back 할 수 있도록 구현되어야 함
- ✓ Roll-back 기능은 각자 방법대로 구현 가능. 위의 예제는 참고 사항
- ✓ 탐색 방향은 $\mathbf{d}_{(n\times 1)} = \mathbf{d}^{+}{}_{(n\times 1)} - \mathbf{d}^{-}{}_{(n\times 1)}$ 로 구해짐

SQP Programming Guide

5) 강하 함수 구현

✓ 강하 함수의 정의

$$\Phi(\mathbf{x}) = f(\mathbf{x}) + R \cdot V(\mathbf{x})$$

여기서, $R = \max\{R_0, \sum_{i=1}^{n} |\mathbf{v}_i| + \sum_{j=1}^{m} \mathbf{u}_j\}$: 벌칙 매개 변수로서 모든 Lagrange multiplier의 합과 R_0중 큰값

$V(\mathbf{x}) = \max\{0; |h_1|, |h_2|, |h_3|, \ldots; g_1, g_2, g_3, \ldots\}$: 현재의 설계점에서의 최대 제약 조건 위배 값과 0 중 큰값
제약조건을 위배하지 않을 경우 0

```
double SQP::PenaltyFunction(double *_x)
{
    // QP문제에서 구한 v,u 를 더함: R
    // _x값을 대입하여 등호 및 부등호 제약 조건의 함수값을 더함: V(x)
    return m_ObjFn(_x) + R * V;
}
```


SQP Programming Guide

6) 1차원 탐색 방법(예: 황금분할법)을 이용하여 강하 함수(Penalty Function)를 최소로 하는 이동거리를 검출

FindNextMinPoint 함수 구현

```
void SQP::FindNextMinPoint()
{
    double** section;
    double* section = new double* [3];
    for(int i=0;i<3;i++)
    {
        section[i] = new double [m_NumOfVariable];
    }

    findMinValueExistSection(m_X, m_d, section, PenaltyFunction);
    m_MinValue = GoldenSectionSearch(section, PenaltyFunction, m_X);

    for(int i=0;i<3;i++)
    {
        delete[] section[i];
    }
    delete []section;
}
```

QP에서 구해진 방향에 따라
1차원 탐색 방법(예: 황금분할법)을 이용하여
강하 함수(Penalty Function)를
최소로 하는 이동거리를 검출 함

Programming Assignment #1에서 자신이 작성한 황금분할법 프로그램 이용