최적화 방법 (Optimization Methods)

Lecture 05

Fall 2025
KAIST MFE
금융수치해석기법

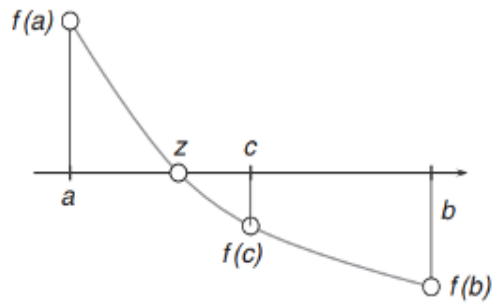
비선형방정식의 해찾기

Finding the roots of $f(x) = 0$

Bisection

Algorithm 30 Bisection.

```
1:  define error tolerance η
2:  if sign f(a) = sign f(b) then stop
3:  while |a − b| > η do
4:      c = a + (b − a)/2
5:      if sign f(a) ≠ sign f(b) then
6:          b = c          # (z left of c)
7:      else
8:          a = c          # (z right of c)
9:      end if
10: end while
```



Fixed point method

- 해를 찾을 때까지 x를 iteration function $g(x)$를 이용해서 업데이트함
- $x_{new} = g(x_{old}) \rightarrow$ generate a sequence x_k for $k = 0, 1, \ldots$

Algorithm 31 Fixed point iteration.

```
1: initialize starting value x^(0)
2: for k = 1,2,... until convergence do
3:     x^(k) = g(x^(k−1))
4: end for
```

Fixed point method 예시

해찾기 예시

$$f(x) = x - x^{7/5} + 1/5 = 0$$

- 다음 2개의 iteration function을 고려

$$g_1(x) = x^{7/5} - 1/5 \qquad g_2(x) = (x + 1/5)^{5/7}$$

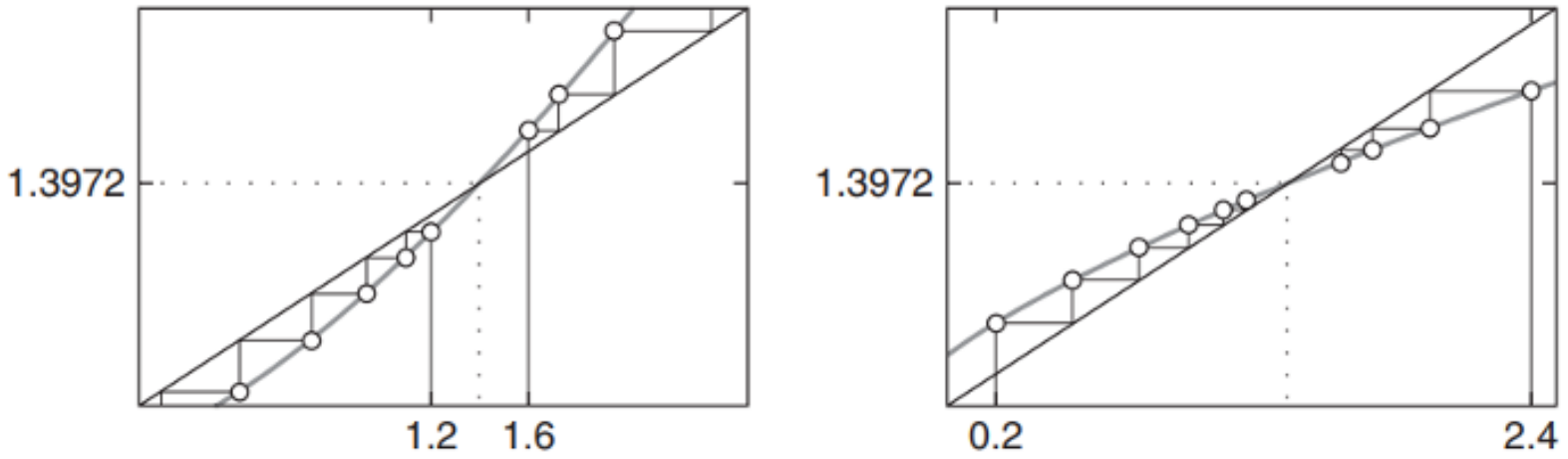


Figure 11.2 Left panel: iteration function g_1. Right panel: iteration function g_2.

Newton's method (one-dimensional)

Taylor expansion:

$$f(x + \Delta x) = f(x) + \Delta x f'(x) + R(x)$$

Step Δx for which we have $f(x + \Delta x) = 0$

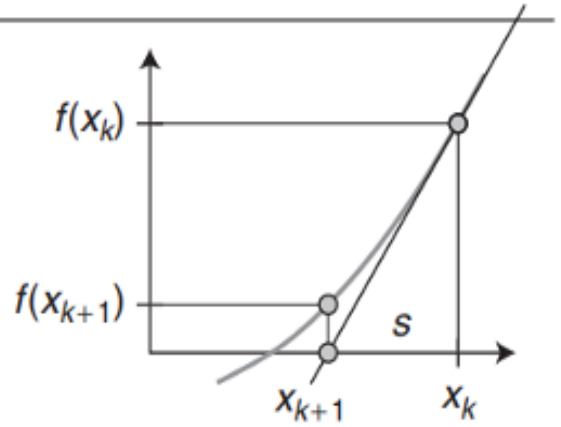
$$\Delta x = -\frac{f(x)}{f'(x)}$$

Step from x_k to x_{k+1}:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Algorithm 32 Newton's method for zero finding.

1: initialize starting value x_0
2: **for** $k = 0, 1, 2, \ldots$ until convergence **do**
3: compute $f(x_k)$ and $f'(x_k)$
4: $s = -\frac{f(x_k)}{f'(x_k)}$
5: $x_{k+1} = x_k + s$
6: **end for**



Optimization Problem

일반적인 최적화 문제

$$x^* = \arg\min_{x \in \mathbb{R}^n} f(x)$$

- f: objective function, x^*: solution
- Feasible set이 $\mathbb{R}^n$인 경우 unconstrained optimization
- Feasible set이 다음 X인 경우 constrained optimization

$$X = \{x \in \mathbb{R}^n | c_i(x) = 0 \text{ for } i \in E \text{ and } c_i(x) \geq 0 \text{ for } i \in I\}$$

  - E: equality constraints, I: inequality constraints

최적화 문제의 종류

- **Linear programming**: $f(x)$와 $c_i(x)$가 linear
- **Quadratic programming**: $f(x)$ is quadratic and $c_i(x)$ are linear
- **Convex programming**: $f(x)$ is convex and X is convex
- **Nonlinear Least Squares**: $f(x) = \frac{1}{2}\sum_{j=1}^{n} f_j^2(x)$ and $X = \mathbb{R}^n$
- **Bound-constrained optimization**: $l_i \leq x_i \leq u_i$

Gradient $\nabla f(x)$와 Hessian matrix $\nabla^2 f(x)$

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad \text{and} \quad \nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

Conditions for local minimizer

Taylor expansion

$$\underbrace{f(x^* + z)}_{x} = f(x^*) + \underbrace{\nabla f(x^*)' z}_{\text{C1}} + \frac{1}{2}\underbrace{z' \nabla^2 f(x^* + \xi) z}_{\text{C2}}$$

Sufficient condition for a local minimizer

- **FOC(C1)**: gradient has to be zero

$$\nabla f(x^*) = 0$$

- **SOC(C2)**: Hessian matrix must be positive-definite

$$z' \nabla^2 f(x^*) z \geq 0, \quad \forall z \in \mathbb{R}^n$$

- C1을 만족하나 C2를 만족하지 못하는 경우 x^*는 안장점(saddle point)임

The problem is closely related to solving nonlinear equations $F(x) = 0$, where $F(x)$ corresponds to $\nabla f(x)$.

Classification of methods for solving optimization problem

Gradient-based methods

- Steepest descent
- Newton's method

Direct search methods

- Golden section method (one dimension)
- Simplex method (Nelder–Mead)

Direct search 방법은 다음과 같이 정의됨

- The method uses only objective function evaluations to determine a search direction.
- The method does not model the objective function or its derivatives to derive a search direction or a step size;
- in particular, it does not use finite differences.

1차원 unconstrained optimization

Newton's method

$$f(x+h) \approx f(x) + f'(x)h + \frac{1}{2}f''(x)h^2$$

- FOC는 $\frac{\partial f(x+h)}{\partial h} = 0$ 로 표현할 수 있음
- 위의 식의 양변을 h에 대해서 미분하면, $0 = f'(x) + f''(x)h$

$$h = -\frac{f'(x)}{f''(x)}$$

Iterating scheme:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Algorithm 33 Newton's method for unconstrained optimization.

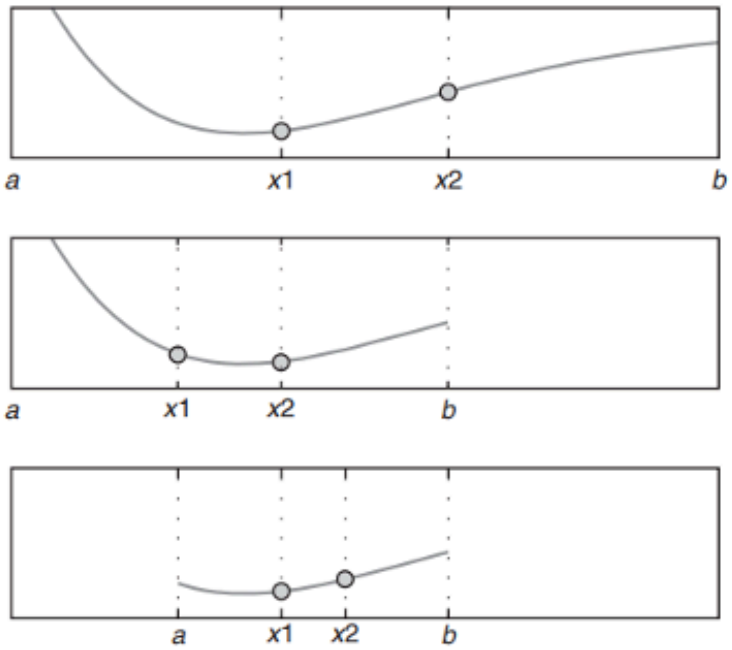
1: initialize $x^{(0)}$ (close to the minimum)
2: **for** $k = 0, 1, 2, \ldots$ `until convergence` **do**
3: compute $f'(x^{(k)})$ and $f''(x^{(k)})$
4: $x^{(k+1)} = x^{(k)} - \frac{f'(x)}{f''(x)}$
5: **end for**

Golden section search

- Unimodal function $f(x)$ in interval $x \in [a, b]$
- searches the minimum by reducing the interval containing the minimum by a constant ratio τ.

Algorithm 34 Golden section search.

1: compute $x_1 = a + (1-\tau)(b-a)$ and $f_1 = f(x_1)$
2: compute $x_2 = a + \tau(b-a)$ and $f_2 = f(x_2)$
3: **while** $(b-a) > \eta$ **do**
4: **if** $f_1 < f_2$ **then**
5: $b = x_2$
6: $x_2 = x_1$
7: $f_2 = f_1$
8: $x_1 = a + (1-\tau)(b-a)$
9: $f_1 = f(x_1)$
10: **else**
11: $a = x_1$
12: $x_1 = x_2$
13: $f_1 = f_2$
14: $x_2 = a + \tau(b-a)$
15: $f_2 = f(x_2)$
16: **end if**
17: **end while**



다차원 unconstrained optimization

Steepest descent method

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

- 여기에서 α는 다음 step에서 $f(x)$의 값을 가장 작게 만드는 최적해임

$$\alpha^* = \arg\min_{\alpha} f(x_k - \alpha \nabla f(x_k))$$

Algorithm 35 Steepest descent method.

1: initialize $x^{(0)}$
2: **for** $k = 0, 1, 2, \ldots$ `until convergence` **do**
3: compute $\nabla f(x^{(k)})$
4: compute $\alpha^* = \text{argmin}_\alpha f\left(x^{(k)} - \alpha \nabla f(x^{(k)})\right)$
5: $x^{(k+1)} = x^{(k)} - \alpha^* \nabla f(x^{(k)})$
6: **end for**

Steepest descent method 예시

$$f(x_1, x_2) = \exp(0.1(x_2 - x_1^2)^2 + 0.05(1 - x_1)^2)$$

$$\text{Starting point} \quad x^0 = [-0.3, 0.8]$$

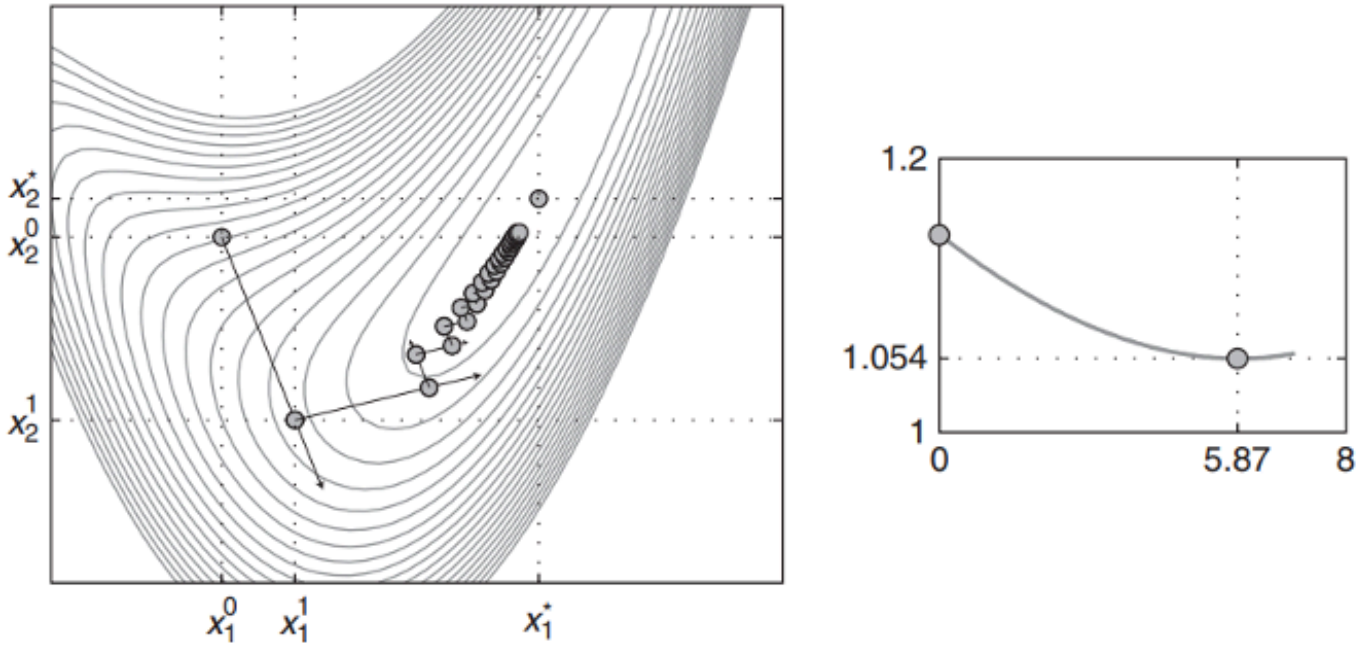


Figure 11.7 Minimization of $f(x_1,x_2) = \exp\left(0.1\,(x_2 - x_1^2)^2 + 0.05\,(1 - x_1)^2\right)$ with the steepest descent method. Right panel: minimization of α for the first step ($\alpha^* = 5.87$).

Newton's method

Quadratic approximation:

$$f(x+h) \approx f(x) + \nabla f(x)h + \frac{1}{2}h'\nabla^2 f(x)h$$

$$\frac{\partial f(x+h)}{\partial h} = \nabla f(x) + \nabla^2 f(x)h = 0$$

$$\nabla^2 f(x)h = -\nabla f(x)$$

Algorithm 36 Newton's method for unconstrained optimization in n dimensions.

1: initialize $x^{(0)}$
2: **for** $k = 0, 1, 2, \ldots$ `until convergence` **do**
3: compute $\nabla f(x^{(k)})$ and $\nabla^2 f(x^{(k)})$
4: solve $\nabla^2 f(x^{(k)})\, s^{(k)} = -\nabla f(x^{(k)})$
5: $x^{(k+1)} = x^{(k)} + s^{(k)}$
6: **end for**

Newton's method 예시

- 앞의 예시에 대한 Newton's method 해법

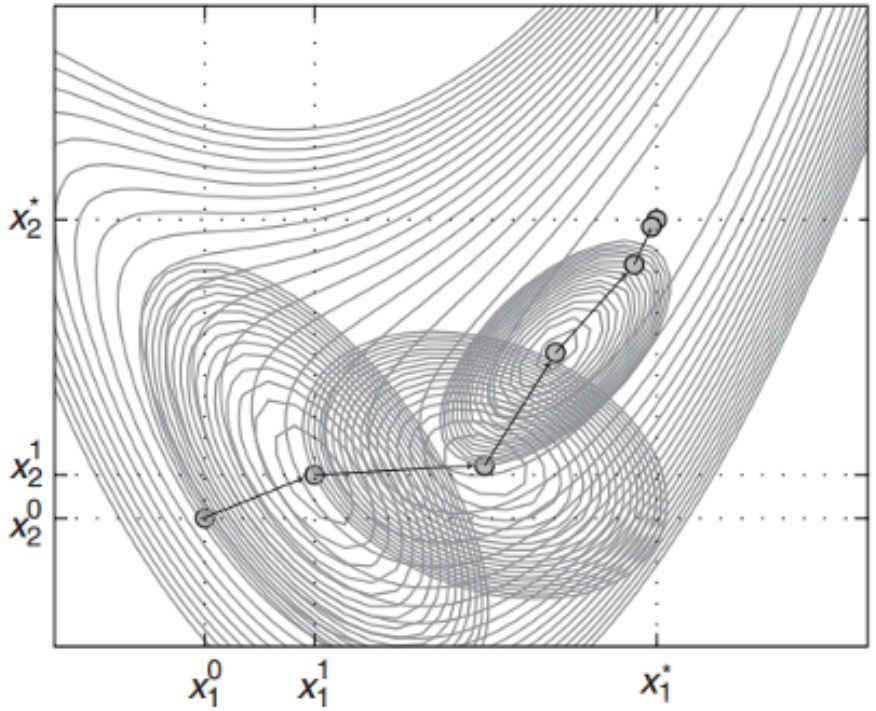


Figure 11.8 Minimization of $f(x_1,x_2) = \exp\left(0.1\,(x_2 - x_1^2)^2 + 0.05\,(1 - x_1)^2\right)$ with Newton's method. Contour plots of the local model for the first three steps.

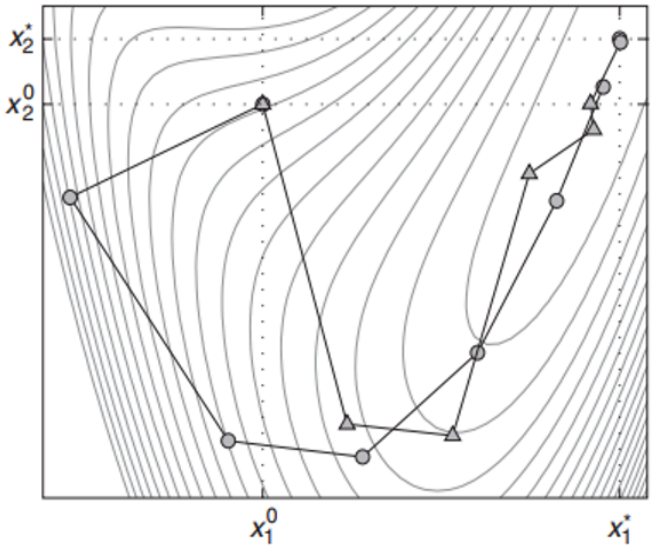
Quasi-Newton method

Approximate gradients and Hessian matrices:

Algorithm 37 Quasi-Newton for unconstrained optimization in n dimensions.

1: initialize $x^{(0)}$ and B_0
2: **for** $k = 0, 1, 2, \ldots$ `until convergence` **do**
3: solve $B_k p^{(k)} = -\nabla f(x^{(k)})$
4: $s^{(k)} = \alpha p^{(k)}$ # (Line search along $p^{(k)}$)
5: $x^{(k+1)} = x^{(k)} + s^{(k)}$
6: $y^{(k)} = \nabla f(x^{(k+1)}) - \nabla f(x^{(k)})$
7: update $B_{k+1} = B_k + U$
8: **end for**

- Comparison of the first seven Newton iterations (circles) with the first five BFGS iterations (triangles)



BFGS(Broyden, Fletcher, Goldfarb, and Shanno)

- Updating matrix

$$U = \frac{y^{(k)} y^{(k)\prime}}{y^{(k)\prime} s^{(k)}} - \frac{(B_k s^{(k)})(B_k s^{(k)})'}{s^{(k)\prime} B_k s^{(k)}}$$

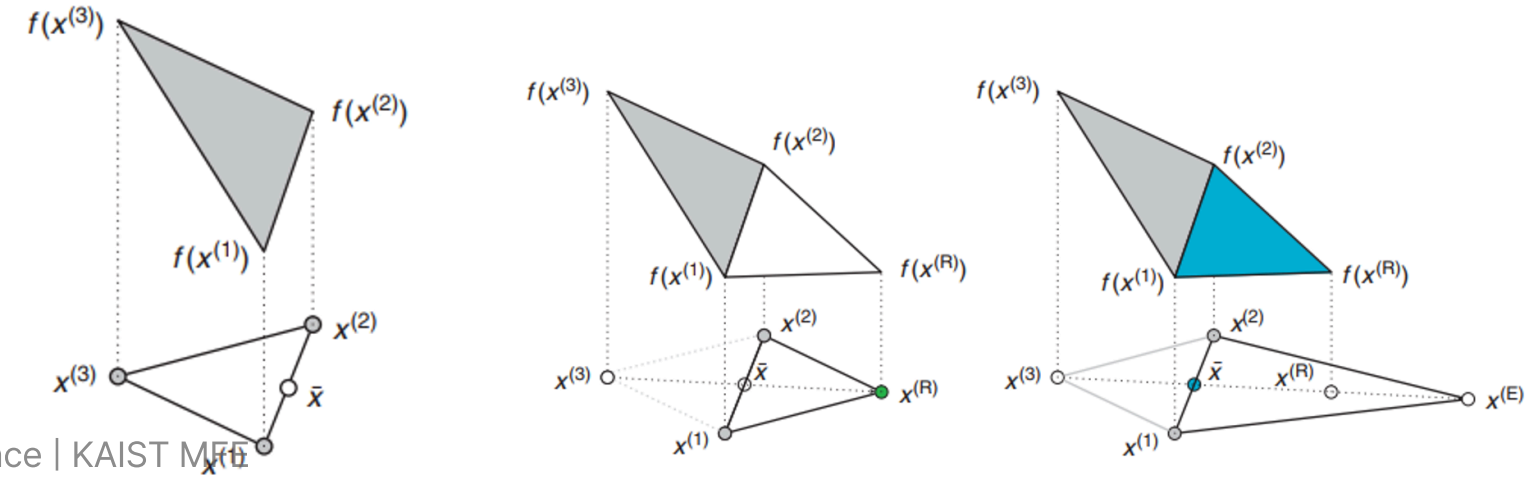
Direct Search Methods

- 최적화 함수가 다음의 문제를 가지는 경우 직접탐색법(direct search methods)이 적합
  - $f(x)$의 도함수가 존재하지 않거나 구할 수 없는 경우
  - $f(x)$의 계산이 매우 비싸며, 이는 많은 양의 시뮬레이션을 통해 얻어지는 경우
  - $f(x)$의 값이 본질적으로 부정확하거나 잡음이 포함된 경우, 예를 들어 몬테카를로 방법의 분산에 영향을 받을 때
  - $f(x)$의 완벽한 최적값보다는 단순한 개선에만 관심이 있는 경우

Simplex 기반의 직접탐색법(Nelder-Mead simplex)

$$x_R = (1+\rho)\bar{x} - \rho x_{n+1}$$

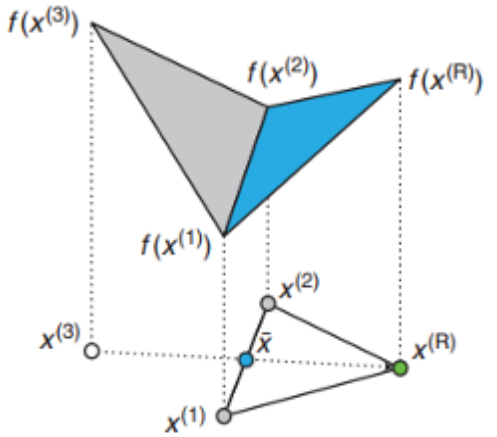
$$x_E = (1+\rho)x_R - \rho\bar{x} \quad \text{if } f(x_R) < f(x_1)$$

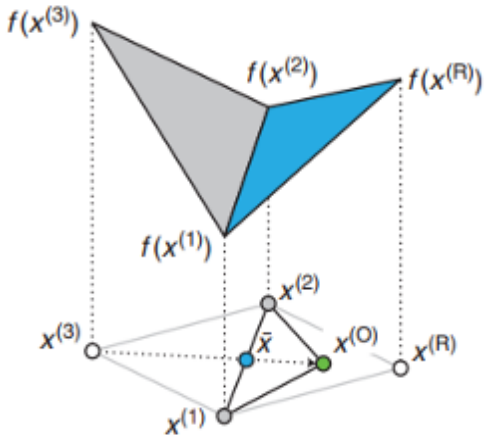


Nelder-Mead simplex

- 또는 만약에 $f(x_n) \leq f(x_R) < f(x_{n+1})$ 인 경우, out-contraction

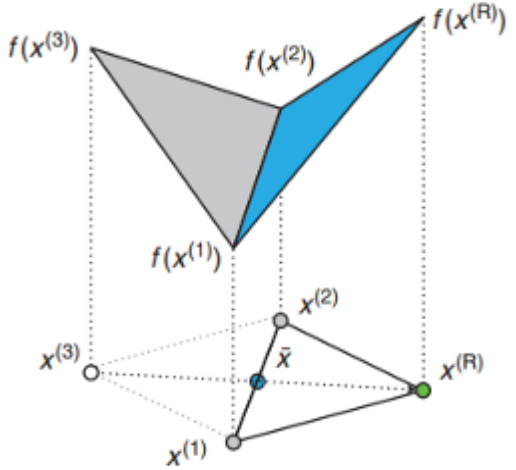
$$x_O = (1 + \psi\rho)\bar{x} - \psi\rho x_{n+1}$$

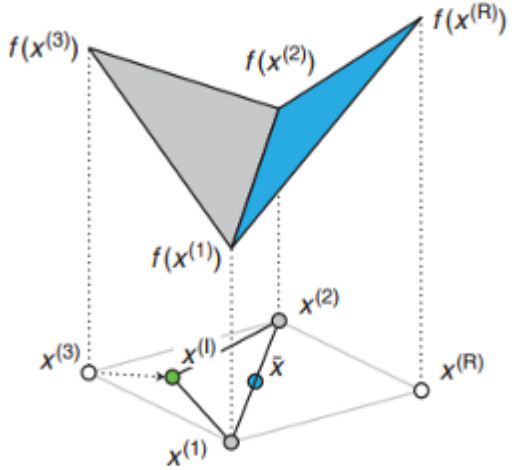




- 또는 만약에 $f(x_R) > f(x_{n+1})$ 인 경우, in-contraction

$$x_I = (1 - \psi\rho)\bar{x} + \psi\rho x_{n+1}$$

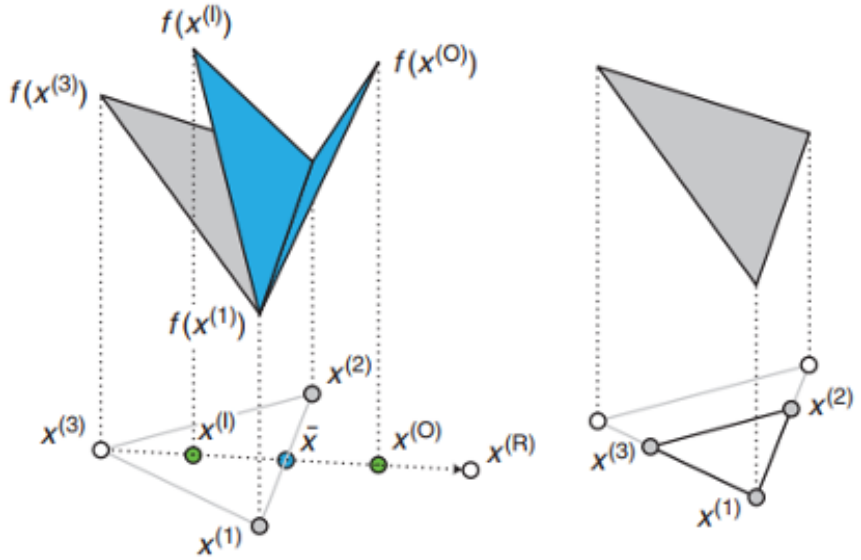




Nelder-Mead simplex

- Finally, if outside or inside contraction results in no improvement over $f(x_{n+1})$, we shrink the simplex:

$$x_i = x^{(1)} - \sigma(x_i - x_1), \quad i = 2, \ldots, n+1$$



- Typical values: $\rho = 1, \psi = 0.5, \sigma = 2$

Nelder-Mead simplex

Algorithm 38 Nelder–Mead simplex direct search.

construct vertices $x^{(1)},\ldots,x^{(n+1)}$ of starting simplex
while stopping criteria not met **do**
    rename vertices such that $f(x^{(1)}) \le \cdots \le f(x^{(n+1)})$
    if $f(x^{(\mathrm{R})}) < f(x^{(1)})$ **then**
        if $f(x^{(\mathrm{E})}) < f(x^{(\mathrm{R})})$ **then** $x^* = x^{(\mathrm{E})}$ **else** $x^* = x^{(\mathrm{R})}$
    else
        if $f(x^{(\mathrm{R})}) < f(x^{(n)})$ **then**
            $x^* = x^{(\mathrm{R})}$
        else
            if $f(x^{(\mathrm{R})}) < f(x^{(n+1)})$ **then**
                if $f(x^{(\mathrm{O})}) < f(x^{(n+1)})$ **then** $x^* = x^{(\mathrm{O})}$ **else** shrink
            else
                if $f(x^{(\mathrm{I})}) < f(x^{(n+1)})$ **then** $x^* = x^{(\mathrm{I})}$ **else** shrink
                end if
            end if
        end if
    end if
    if not shrink **then** $x^{(n+1)} = x^*$ # Replace worst vertex by x^*
end while

Non-linear Least Squares

- **Objective function:**

$$g(x) = \frac{1}{2} r(x)' r(x) = \frac{1}{2} \sum_{i=1}^{m} r_i(x)^2$$

$$r_i(x) = y_i - f(t_i, x)$$

- **The first derivative:**

$$\nabla g(x) = \sum_{i=1}^{m} r_i(x) \cdot \nabla r_i(x) = \nabla r(x)' r(x),$$

$$\text{where} \nabla r(x) = \begin{bmatrix} \frac{\partial r_1(x)}{\partial x_1} & \cdots & \frac{\partial r_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial r_m(x)}{\partial x_1} & \cdots & \frac{\partial r_m(x)}{\partial x_n} \end{bmatrix}$$

- **The second derivative:**

$$\nabla^2 g(x) = \sum_{i=1}^{m} \left(\nabla r_i(x) \cdot \nabla r_i(x)' + r_i(x) \cdot \nabla^2 r_i(x) \right)$$

$$= \nabla r(x)' \nabla r(x) + S(x),$$

$$\text{with } S(x) = \sum_{i=1}^{m} r_i(x) \nabla^2 r_i(x)$$

Newton's method

- **Quadratic approximation of $g(x)$**

$$m_c(x) = g(x_c) + \nabla g(x_c)'(x - x_c) + \frac{1}{2}(x - x_c)'\nabla^2 g(x_c)(x - x_c)$$

- FOC $\nabla m_c(x^+) = 0$을 만족하는 $x^+ = x_c + s_N$ 을 탐색함

$$\nabla m_c(x^+) = \nabla g(x_c) + \nabla^2 g(x_c)(x^+ - x_c) = 0$$
$$\nabla^2 g(x_c) s_N = -\nabla g(x_c)$$

- 즉, Newton's method에서 k번째 iteration은 다음과 같이 정의됨

$$\nabla^2 g(x_k) s_N^k = -\nabla g(x_k)$$
$$x_{k+1} = x_k + s_N^k$$

- 문제는 second derivatives인 $\nabla^2 g(x_k)$ 를 구하는 방법

Gauss-Newton method

Algorithm:

Algorithm 39 Gauss–Newton method.

1: initialize $x^{(0)}$
2: **for** $k = 0, 1, 2, \ldots$ `until convergence` **do**
3: compute $\nabla r(x^{(k)})$
4: Solve $\left(\nabla r(x^{(k)})' \nabla r(x^{(k)})\right) s_{\text{GN}}^{(k)} = -\nabla r(x^{(k)})' r(x^{(k)})$
5: update $x^{(k+1)} = x^{(k)} + s_{\text{GN}}^{(k)}$
6: **end for**

- Gauss-Newton step s_{GN}^k 는 $A'Ax = A'b$의 normal equation system 으로 정의되며, 매 iteration마다 least-square 문제를 풀어야 함
- 이 문제는 다음의 선형방정식 문제를 해결하는 것과 동일하며, QR 분해를 이용해서 보다 안정적으로 풀 수 있음

$$\nabla r(x_k) s_{GN}^k = -r(x_k)$$

Levenberg–Marquardt method

- LM 방법에서는 행렬 $S(x)$를 μI의 대각행렬로 근사함

Algorithm 40 Levenberg–Marquardt method.

1: initialize $x^{(0)}$
2: **for** $k = 0, 1, 2, \ldots$ `until convergence` **do**
3: compute $\nabla r(x^{(k)})$ and μ_k
4: solve $\left(\nabla r(x^{(k)})' \nabla r(x^{(k)}) + \mu_k I\right) s_{\text{LM}}^{(k)} = -\nabla r(x)' r(x)$
5: update $x^{(k+1)} = x^{(k)} + s_{\text{LM}}^{(k)}$
6: **end for**

- The step $s_{LM}^{(k)}$ is the solution of a linear Least Squares problem and is computed by solving the overidentified system

$$\begin{bmatrix} \nabla r(x^{(k)}) \\ \mu_k^{1/2} I \end{bmatrix} s_{LM}^{(k)} \approx - \begin{bmatrix} r(x^{(k)}) \\ 0 \end{bmatrix},$$

Solving systems of nonlinear equations $F(x) = 0$

System of n nonlinear equations:

$$F(y) = 0, \quad \begin{cases} f_1(y) = 0 \\ \vdots \\ f_n(y) = 0 \end{cases}$$

- 최소 1개 이상의 f_i 함수가 non-linear일 경우

Jacobian matrix

$$\nabla F(y) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial y_1} & \cdots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}$$

Jacobi, Gauss-Seidel, SOR

Jacobi method

$$y_i^{k+1} = g_i(y_1^k, \ldots, y_{i-1}^k, y_{i+1}^k, \ldots, y_n^k)$$

Gauss-Seidel

$$y_i^{k+1} = g_i(y_1^{k+1}, \ldots, y_{i-1}^{k+1}, y_{i+1}^k, \ldots, y_n^k)$$

Stopping criterion

$$\frac{|y_i^{k+1} - y_i^k|}{|y_i^k| + 1} < \epsilon$$

Algorithm 41 Jacobi, Gauss–Seidel, and SOR for nonlinear systems.

1: initialize $y^{(1)}$, $y^{(0)}$, ε and maximum number of iterations
2: **while** ˜converged $(y^{(0)}, y^{(1)}, \varepsilon)$ **do**
3: $y^{(0)} = y^{(1)}$ # Store precedent iteration in $y^{(0)}$
4: compute $y^{(1)}$ with Jacobi, Gauss–Seidel, or SOR
5: check number of iterations
6: **end while**

Newton's method (Newton-Raphson)

Improving y^{k+1} by approximating $F(y)$

$$F(y) \approx F(y^k) + \nabla F(y^k)(y - y^k) = 0$$

- 위 식을 만족하는 y 값을 찾음

$$y^{k+1} = y^k - [\nabla F(y^k)]^{-1} F(y^k)$$

- y^{k+1}는 $\nabla F(y^k)(y^{k+1} - y^k) = -F(y^k)$ 의 solution임

Algorithm 42 Newton's method for nonlinear systems.

1: initialize $y^{(0)}$
2: **for** $k = 0, 1, 2, \ldots$ `until` convergence **do**
3: compute $b = -F(y^{(k)})$ and $J = \nabla F(y^{(k)})$
4: verify condition of J
5: solve $J s = b$
6: $y^{(k+1)} = y^{(k)} + s$
7: **end for**

Quasi-Newton methods

- At every iteration, the evaluation of the Jacobian matrix is necessary. n^2의 미분계수 계산 필요
- Jacobi 행렬을 정확하게 구하는 대신 쉬운 계산으로 Jacobi 행렬을 업데이트하는 방법

Broyden's method

$$B^{(k+1)} = B^{(k)} + \frac{(dF^{(k)} - B^{(k)}s^{(k)})s^{(k)'}}{s^{(k)'}s^{(k)}},$$

- $dF^{(k)} = F(y^{k+1}) - F(y^k)$이고, $s^{(k)}$는 $B^{(k)}s^{(k)} = -F(y^k)$의 solution임

Algorithm 43 Broyden's method for nonlinear systems.

1: initialize $y^{(0)}$ and $B^{(0)}$ (an approximation for $\nabla F(y^{(0)})$)
2: **for** $k = 0, 1, 2, \ldots$ `until` convergence **do**
3: solve $B^{(k)}s^{(k)} = -F(y^{(k)})$
4: $y^{(k+1)} = y^{(k)} + s^{(k)}$
5: $dF^{(k)} = F(y^{(k+1)}) - F(y^{(k)})$
6: $B^{(k+1)} = B^{(k)} + (dF^{(k)} - B^{(k)}s^{(k)})s^{(k)'}/(s^{(k)'}s^{(k)})$
7: **end for**