산업기술연구(강원대학교 산업기술연구소 논문집), 제20권, 2000.
Journal of Industrial Technology, Kangwon Nat'l Univ., Korea, No. 20, 2000.

최적화 알고리듬들의 객체지향 C++ 라이브러리의 개발

Development of Object-Oriented C++ Library of Optimization Algorithms

현 창 헌* 최 영 일**
Hyun, Chang-Hun Choe, Yeong-Il

Abstract

There are many optimal design packages, but they are big ones and they have only a few kinds of optimal algorithm coded with Fortran and it is sometimes necessary for user to write down some codes into their packages. So it is hard for user to learn how to use and customize them. More over, there are no commercial home-made software for optimum design. So, in this paper, several famous optimum algorithms are coded and modulized with C++ which is known as a suitable computer language in order to build up more algorithms into one computer software. All algorithms developed with C++ here were tested for comparison with optimization tool box of MATLAB and are superior to MATLAB.

키워드 : 최적화 알고리즘, 최적화 라이브러리
Keywords : optimization algorithm, optimization library

1. 서 론

최적화(optimization)란 주어진 환경하에서 최적의 결과를 산출하는 것을 말한다. 대부분의 공학분야에서 시스템의 설계는 최적화 문제로 정식화 시킬 수 있으며 다양한 최적화 기법을 통하여 해결할 수 있다.

최적화 문제를 푸는 방법으로는 최적성 규준(optimality criteria), 탐색법(search method) 등이 있다. 최적성 규준 방법은 최적점에 대한 필요조건 및 충분조건인 최적성 조건(optimality condition)을 풀어 해를 찾는 방법이다. 탐색법은 초기 설계점을 주어 반복과정을 통해 최적해를 찾는 방법이다.

대부분의 최적화 수치법은 상당한 반복계산과정을 포함하고 있기 때문에 적절한 컴퓨터 코드로 바꾸어져 있다. 이러한 컴퓨터 코드는 많은 개발자들에 의해 라이브러리화 되었으며 대표적으로 MATLAB, IMSL, ACM 등에서 제공되고 있다. 이외에 주요 최적화 소프트웨어와 기능을 Table 1에 요약하였다.

지금까지 개발되어 제공되는 최적화 소프트웨어는 대부분이 대형 패키지이어서 사용자가 접근하기가 용이하지 않다. 또한 모두 외국의 것으로서 국내에서는 이러한 최적화 소프트웨어가 개발되어 널리 보급된 것이 없다. 이에 본 연구에서는 최적화 설계문제를 푸는데 있어서 여러 알고리듬들을 쉽게 사용할 수 있도록 자료구조에 대해 체계적으로 정리된 C++ 최적화 모듈을 합성한 라이브러리를 개발하고자 하였다. 교육적이면서도 널리 사용되는 최적화 알고리듬들을 선정하여 수학적인

* 강원대학교 기계메카트로닉스공학부 교수
** 강원대학교 대학원 기계메카트로닉스공학과 석사과정

Table 1. Summary of some optimization packages.

Software	Source(Developer)	Capabilities
UNCMIN	Koontz *et al.* (1985)	Common nonlinear optimization
NPSOL	Gill *et al.* (1986)	Minimizing smooth objective function
MINOS	Murtagh and Saunders (1987)	Linear programming problem
IDESIGN	Arora and Tseng (1987)	Linear and nonlinear programming problem
OSL	IBM (1990)	Quadratic programming problem
TENMIN	Schnabel and Chow (1991)	Unconstrained optimization, quadratic problem
LANCELOT	Conn *et al.* (1992)	Large-scale nonlinear optimization

계층도를 바탕으로 모듈화를 구현하였으며 따라서 다른 모듈의 개발 및 추가를 용이하게 실현시킬 수 있게 하였다.

2. 최적성 판단기준

최적성 판단기준(optimality criteria) 방법은 최적점에 대한 최적성 조건(optimality condition)을 풀어 해를 찾는 방법으로 목적함수가 연속이고 미분가능하다면 매우 유용한 방법이다. 실제에 있어 대부분의 목적함수는 불연속이고 미분 불가능한 경우가 많기 때문에 최적성 규준 방법은 매우 제한적이고 본 연구의 대상은 아니지만 많은 수치해석적 방법들의 저변이론이 되기 때문에 중요하게 다루어지므로 간단히 언급한다.

2.1 최적성 조건

최적성 조건에는 필요조건과 충분조건이 있다. 최적점에서 만족되어야 할 조건들을 필요조건이라 한다. 필요조건을 만족하는 점을 후보최적점(candidate optimum point)이라 한다. 그러나 필요조건을 만족한다고 해서 최적점인 것은 아니다. 따라서 비최적점도 필요조건을 만족시킬 수 있다. 후보최적점이 충분조건을 만족하면 그 점은 최적점이 된다.

임의의 점 $\boldsymbol{x}^*$에서 Taylor 전개식은 식 (2.1)과 같다.

$$\Delta f = \nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} + \frac{1}{2}\, \boldsymbol{d}^T \boldsymbol{H}(\boldsymbol{x}^*)\, \boldsymbol{d} + R \tag{2.1}$$

여기서

$$\boldsymbol{d} = \boldsymbol{x} - \boldsymbol{x}^*,$$

$$\boldsymbol{H} = \frac{\partial f_{ij}}{\partial x_i \partial x_j},$$

$i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, n$ (n = 변수의 수).

최소점 $\boldsymbol{x}^*$를 찾았다고 가정했을 때, 이 점 근방에서 함수값의 변화가 없거나 증가해야만 한다. 그러기 위해서는 1계 미분항까지 고려했을 때 다음의 식 (2.2)를 만족해야 한다.

$$\nabla f(\boldsymbol{x}^*) = \boldsymbol{0} \tag{2.2}$$

이때 식 (2.2)를 $\boldsymbol{x}^*$에서 $f(\boldsymbol{x})$가 최소이기 위한 1계 필요조건(first order necessary condition)이라 하며 $\boldsymbol{x}^*$를 정류점 또는 상점(stationary point)이라고 한다.

식 (2.1)에 식 (2.2)를 적용하면 식 (2.3)과 같다.

$$\Delta f = \frac{1}{2}\, \boldsymbol{d}^T \boldsymbol{H}(\boldsymbol{x}^*)\, \boldsymbol{d} + R \tag{2.3}$$

여기서 마차가지로 $\boldsymbol{x}^*$의 근방에서 함수값의 변화가 증가해야만 위해서는 2계 미분항까지 고려했을 때 모든 $\boldsymbol{d} \neq \boldsymbol{0}$에 대하여 식 (2.4)를 만족해야 한다.

$$\boldsymbol{d}^T \boldsymbol{H}(\boldsymbol{x}^*)\, \boldsymbol{d} > 0 \tag{2.4}$$

식 (2.4)를 $\boldsymbol{x}^*$에서 $f(\boldsymbol{x})$가 최소이기 위한 2계 충분조건(second order necessary condition)이라 한다.

2.2 라그랑지승수의 정리

전술된 최적성 조건을 사용하여 목적함수 $f(\boldsymbol{x})$의 최소점을 찾을 수 있다. 그러나 제약조건이 있는 경우 최적성 조건을 사용하는데 있어서 목적함수 $f(\boldsymbol{x})$의 성질만이 최소점을 결정하는 데 이용되어질 수 없다.

제약문제의 경우 해석학(calculus)에서 다루는 라그랑지 승수(Lagrange multiplier)에 관한 정리를 도입하여 각각의 제약조건에 대응하는 라그랑지 승수를 두어 라그랑지 함수를 만듬으로써 최적성 조건을 사용할 수 있다.

3. 최적화 수치법

대부분의 공학 설계문제에서 설계변수와 제약조건의 수가 많을 수 있다. 또한 정식화된 목적함수 및 제약함수는 비선형성이 심할 수 있다. 이러한 비선형성은 해석적으로 방정식을 풀기 어렵게 만든다. 따라서 해석적 방법은 적절하지 못하다.

수치적 방법은 최적점으로 초기 추정치의 설계를 선정하고 최적성조건을 만족할 때까지 그것을 반복적으로 변화시키는 과정을 통하여 최적성을 만족하도록 노력한다.

일반적으로 수치법들은 식 (3.1)와 같은 반복과정으로 표현된다.

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \Delta \boldsymbol{x}^{(k)} \qquad (3.1)$$

여기서 $\Delta \boldsymbol{x}^{(k)}$는 현재설계에서의 미소변화를 나타내는데 $\Delta \boldsymbol{x}^{(k)}$는 식 (3.2)와 같이 두 부분으로 분리할 수 있다.

$$\Delta \boldsymbol{x}^{(k)} = \alpha_k \boldsymbol{d}^{(k)} \qquad (3.2)$$

여기서 $\boldsymbol{d}^{(k)}$는 설계공간에서 움직이고자 하는 탐색방향이고 α_k는 양의 스칼라로써 그 방향에서의 이동거리(step size)라고 부른다. 따라서 $\Delta \boldsymbol{x}^{(k)}$의 계산과정은 두 별개의 부문제로 나누어진다.

3.1 비제약문제의 최적화방법

이동거리를 결정하는 문제는 일차원 탐색(one dimensional search) 또는 선탐색(line search)문제라 부른다. 어떤 순간에 설계변화의 바람직한 방향 $\boldsymbol{d}^{(k)}$를 찾은 후 $\boldsymbol{x}^{(k+1)}$를 목적함수에 대입하면 목적함수는 α_k만의 함수가 된다. 이때 α_k만의 목적함수의 최소값은 설계변화 방향 $\boldsymbol{d}^{(k)}$로의 최대 이동거리의 의미를 갖는다.

일차원 탐색문제는 역시 필요조건을 사용한 해석적 방법과 수치적인 방법이 있다. 여기서는 구간탐색법과 다항식 보간법에 관한 수치적인 방법에 대하여 설명하겠다.

1) 등간격 탐색

일차원 최적화 알고리듬의 기본 개념은 최소점이 존재하는 구간, 즉 불확정구간(interval of uncertainty, I)을 주어진 미소값에 수렴시키는 것이다. 등간격 탐색(equal interval search)은 매우 간단한 방법으로 먼저 시작점으로부터 일정한 간격 δ로 증가하여 함수값을 평가하여 불확정구간을 감소시키는 과정을 반복하는 방법이다.

2) 황금분할 탐색

황금분할 탐색(golden section search)은 피보나치(Fibonacci) 수열로 초기의 I를 결정하여 I를 줄여나가는 방법이다.

3) 다항식 보간법

황금분할 탐색은 초기 최소점의 존재 구간의 결정은 빠른데 비해 초기의 I가 결정된 후 함수값의 평가수가 많다는 단점이 있다. 다항식 보간법은 주어진 구간 I에서의 근사 다항식을 구성하여 다항식의 최소점으로서 정확한 최소점을 추정하는 방법이다. 근사 다항식으로 2차 또는 3차가 많이 사용되어 진다.

지금까지의 세가지 방법은 탐색방향을 알고 있다고 생각하고 이동거리를 정하는 방법에 대하여 논하였다. 그러면 탐색방향 $\boldsymbol{d}^{(k)}$를 정하는 문제들에 대하여 아래에서 살펴 보겠다.

4) 최속강하법

탐색방향 $\boldsymbol{d}^{(k)}$에 필요한 기본요구는 그 방향으로 미소이동을 시키면 목적함수가 감소하여야 한다. 이것을 강하방향이라 부른다. 1847년 코시(Cauchy)는 경사도 벡터(gradient vector)의 성질 중 경사도가 점 $\boldsymbol{x}^*$에서 함수 $f(\boldsymbol{x})$의 최대 증가율 방향을 가리킨다는 것을 도입하여 $\boldsymbol{d}^{(k)}$를 결정하였는데 이러한 방법을 최속강하법(steepest decent method) 또는 경사도법(gradient method)이라 한다[1].

$$\boldsymbol{d}^{(k)} = -\nabla f(\boldsymbol{x}^{(k)}) \qquad (3.3)$$

만일 $\|\nabla f(\boldsymbol{x}^{(k)})\|$가 미소값 ϵ보다 작으면 반복과정을 중단하고 $\boldsymbol{x}^* = \boldsymbol{x}^{(k)}$는 최소점이 된다. 그렇지 않으면 설계를 수정하고 반복과정을 계속한다.

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{d}^{(k)} \qquad (3.4)$$

5) 공액경사도법

Fletcher와 Reeves[2]에 의해 만들어진 공액경사도법(conjugate gradient method)은 최속강하법을 매우 간단하고 효율적으로 수정한 것이다. 최속강하법의 두 연속단계의 최속강하방향은 서로 수직이다. 이것이 최속강하법이 수렴하기는 하나 느리

게 만든다. 공액경사방향은 서로 수직이 아니다. 오히려 이들 방향은 수직의 최속강하방향의 대각선을 통과하게 된다. 따라서 최속강하법의 수렴률을 상당히 향상시킨다.

각 반복과정의 경사방향은 식 (3.5)와 같다. 이때 $\| \nabla f(\boldsymbol{x}^{(k)}) \| < \varepsilon$ 이라면 반복과정을 종료한다.

$$\boldsymbol{d}^{(k)} = - \nabla f(\boldsymbol{x}^{(k)}) + \beta_k \boldsymbol{d}^{(k-1)} \quad (3.5)$$

여기서

$$\beta_k = \begin{cases} 0 & \text{for } k=1 \\ \dfrac{\| \nabla f^{(k)} \|^2}{\| \nabla f^{(k-1)} \|^2} & \text{for } k \geq 2 \end{cases} \quad (3.6)$$

방향이 결정되면 이동거리를 계산하여 설계를 바꾸고 반복과정을 계속한다.

6) 뉴톤의 방법

최속강하법과 공액경사도법은 1계 미분정보만을 이용하여 방향을 결정하였다. 만약 2계의 미분정보의 이용이 가능하면 목적함수의 표면을 보다 정확하게 나타낼 수 있으며, 더 좋은 이동방향을 알 수 있다.

뉴톤의 방법(Newton's method)의 현 설계점에서 함수를 2차항까지 Taylor 전개하여 필요조건 $\partial f / \partial (\Delta \boldsymbol{x}) = \boldsymbol{0}$ 을 적용하는 것인데 설계변화 $\Delta \boldsymbol{x}$는 식 (3.7)과 같다.

$$\Delta \boldsymbol{x} = - \boldsymbol{H}^{-1} \nabla f \quad (3.7)$$

위와 같은 고전적인 뉴톤의 방법은 $\Delta \boldsymbol{x}$ 방향으로 단위길이의 이동거리가 사용된다. 하지만 $\Delta \boldsymbol{x}$를 따라 완전한 길이 이동을 하면 목적함수에 강하단계가 되지 않을 수 있다. 따라서 고전적인 뉴톤의 방법은 수렴을 보장하지 못한다[1]. 이것의 보완으로 수정된 뉴톤의 방법은 식 (3.8)과 같이 이동방향을 결정한다.

$$\boldsymbol{d}^{(k)} = - \boldsymbol{H}^{-1} \nabla f^{(k)} \quad (3.8)$$

방향이 계산되면 일차원 탐색법으로써 이동방향 α_k를 결정하여 설계를 수정한다.

앞서 설명된 최속경사법은 1계 미분정보만을 사용하므로 수렴률이 나쁘다고 지적되었다. 이 결점은 뉴톤의 방법으로 보정된다.

그러나 뉴톤의 방법은 2계 미분과정에서 소모적인 계산시간과 계산의 불가능성 등이 문제점으로 나타난다. 또한 반복과정에서 이전의 어떠한 정보도 이용하지 않는다. 이것의 대안으로 1계 미분정보와 앞의 반복회에서 얻은 정보를 조작하여 2계 미분행렬을 근사화하는 방법이 있는데 DFP와 BFGS 방법이 대표적이다.

7) DFP 방법

일반적인 함수에 대하여 가장 강력한 알고리듬 중의 하나로서 Davidon[3]이 처음으로 제안하여 Fletcher와 Powell[4]이 수정한 방법이다.

DFP(Davidon, Fletcher and Powell) 알고리듬은 1계 도함수의 정보를 사용하여 식 (3.9)와 같이 Hessian의 역행렬을 근사시킨다.

$$\boldsymbol{H}^{-1(k+1)} = \boldsymbol{H}^{-1(k)} + \frac{\boldsymbol{s}^{(k)} \boldsymbol{s}^{(k)T}}{\boldsymbol{s}^{(k)T} \boldsymbol{y}^{(k)}} - \frac{\boldsymbol{z}^{(k)} \boldsymbol{z}^{(k)T}}{\boldsymbol{y}^{(k)T} \boldsymbol{z}^{(k)}} \quad (3.9)$$

여기서

$$\boldsymbol{s}^{(k)} = \alpha_k \boldsymbol{d}^{(k)} \quad (3.10)$$

$$\boldsymbol{y}^{(k)} = \nabla f^{(k+1)} - \nabla f^{(k)} \quad (3.11)$$

$$\boldsymbol{z}^{(k)} = \boldsymbol{H}^{-1(k)} \boldsymbol{y}^{(k)} \quad (3.12)$$

8) BFGS 방법

DFP 방법이 Hessian의 역행렬을 근사시키는 것에 반해 BFGS(Broyden-Fletcher-Goldfarb-Shanno) 방법[5,6,7,8]은 Hessian을 근사시키는 것으로서 매우 잘 알려지고 효과적인 방법이다.

$$\boldsymbol{H}^{(k+1)} = \boldsymbol{H}^{(k)} + \frac{\boldsymbol{y}^{(k)} \boldsymbol{y}^{(k)T}}{\boldsymbol{y}^{(k)T} \boldsymbol{s}^{(k)}} + \frac{\nabla f^{(k)} \nabla f^{(k)T}}{\nabla f^{(k)T} \boldsymbol{d}^{(k)}} \quad (3.13)$$

여기서

$$\boldsymbol{s}^{(k)} = \alpha_k \boldsymbol{d}^{(k)} \quad (3.14)$$

$$\boldsymbol{y}^{(k)} = \nabla f^{(k+1)} - \nabla f^{(k)} \quad (3.15)$$

지금까지 비제약문제에 대한 최적화 방법으로 전술된 것들은 미분정보를 필요로 하지만 직접탐색법(direct search mehtod)은 미분정보가 필요없다. 직접탐색법은 대부분이 학습(heuristic) 과정이므로 수렴성이 보장되고 난해한 비선형 문제에도

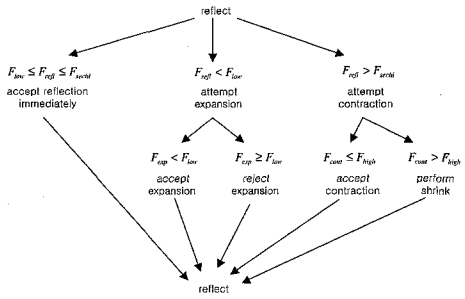

Fig.3 Step transitions for the Nelder-Mead algorithm

적용이 되며 다루기 쉬운 장점이 있다. 직접탐색법 중에서도 심플렉스 방법은 가장 유명한 방법으로서[9] 1965년에 발표된 이후로 최근까지 수정안이 계속해서 발표되고 있다.

9) 심플렉스 방법

Nelder-Mead의 심플렉스(simplex) 방법[10]은 Spendley, Hext, Himsworth [11]의 알고리즘을 수정하여 개발된 방법이다.

Nelder-Mead의 심플렉스 방법은 심플렉스 꼭지점(vertex)들의 함수값들의 크기를 비교하여 가장 큰 함수값을 가지는 꼭지점을 나머지 점들의 중점으로 반전(reflection)시킨다. 이때 함수값들의 크기의 비교에 따라 Fig.3[12]과 같이 보통반전과 확대, 축소, 수축 등의 심플렉스 크기를 조정한다.

3.2 제약문제의 최적화방법

1) 콤플렉스 방법

Box[13]는 비구속 최적화 문제에 대한 Nelder-Mead의 심플렉스 방법을 구속 최적화 문제에 적합하도록 수정하였는데 이 방법을 콤플렉스 방법(complex method)이라 한다.

변수들의 경계 제약조건(bound constraint)를 만족하는 초기점이 주어지면 $k \geq n+1$개의 꼭지점을 갖는 n차원의 기하 도형, 즉 콤플렉스를 생성한다.

$$x_{i,j} = x_i^{(l)} + r_{i,j}(x_i^{(u)} - x_i^{(l)}),$$
$$i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, k \tag{3.16}$$

여기서 $r_{i,j}$는 0과 1사이의 난수이며, $x_i^{(l)}$와 $x_i^{(u)}$는 각각 x_i의 경계 제약조건이다. Box는 표준적인 k로서 $2n$을 추천하였다. 이때 제약조건에 위배되는 꼭지점은 나머지 점들의 기하학적 중점방향으로 축소시킨다.

Nelder와 Mead의 심플렉스 방법은 각 꼭지점에서의 함수값을 비교하여 가장 큰 함수값을 가지는 꼭지점을 그 꼭지점과 나머지 점들의 기하학적 중점의 연장선상에서 반전, 확장 및 축소 등의 과정을 거치지만 Box의 콤플렉스 방법은 일정한 비율 α로서만 반전을 시도한다. Box는 α의 값으로서 1.3을 사용하였다.

2) 난수 탐색법

난수 탐색법(random search)은 매우 간단한 원리로서 변수들의 경계 제약조건을 만족하는 범위에서 난수를 발생시켜 반복과정을 거치면서 최소점을 갱신한다. 따라서 반복과정의 횟수에 의하여 최적점의 정확도가 결정된다. 난수 탐색법은 전역적인 최소점을 찾는다는 장점 때문에 일찍이 사용되어져 왔다. 비록 비효율적이지만 이러한 장점 때문에 전역적인 최소점이 존재하는 영역을 찾는데 사용되어진다[14]. 일단 그 영역이 찾아지면 정확한 최소점을 찾기위해 조금 더 효율적인 방법이 사용될 수 있다.

4. 최적화 알고리듬 모듈의 설계제작

4.1 자료의 구조

본 연구에서 최적화 알고리듬 모듈을 라이브러리로 제작하는 과정은 모두 객체지향 프로그래밍 언어인 C++로 구현하였고 C++의 객체지향 기술의 핵심인 클래스(class)로써 모듈을 구성하였다.

최적화 알고리듬과 같은 수치해석 알고리듬은 기본적인 수치연산함수를 포함하고 있다. 그리고 이것들은 벡터와 행렬의 연산에 바탕으로 하는 선형대수적 연산함수에 근거한다. 이러한 관계를 정리하여 각 모듈의 계층을 크게 4부분으로 나누어 구성하였다. 모든 수치자료에 대한 데이터형 모듈과 선형대수적 연산 및 수치 알고리듬의 기본적인 함수를 내장하는 모듈, 그리고 그 이하의 최적화

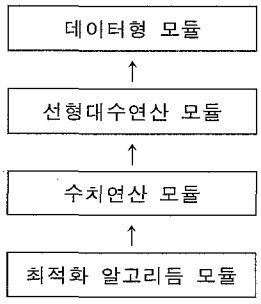

Fig.4 The structure of modules

알고리듬들의 모듈로 이루어져 있다. Fig.4는 모듈의 계층도의 구성을 표현한 것이다.

본 연구에서 라이브러리로 만들어질 최적화 알고리듬은 교육적이면서 널리 사용되는 알고리듬들을 선정하였다. 모든 모듈에 대하여 공통적인 기능들은 모두 상위 모듈로 하였기 때문에 상위 모듈을 이용한 다른 모듈의 개발 및 추가를 용이하게 확장시킬 수 있게 하였다.

4.2 데이터형 모듈

데이터형은 Vector 클래스와 Matrix 클래스로 선언을 하여 Vector연산과 Matrix연산을 자유롭게 사용할 수 있게 하였다. 여기서 벡터 및 행렬연산은 실제 수학적인 표기법으로 사용되어질 수 있게 하였다.

1) Vector 클래스

벡터의 자료형 및 벡터의 기본 연산기능을 포함하고 있다.

2) Matrix 클래스

행렬의 자료형 및 행렬의 기본 연산기능, 역행렬, 전치행렬 연산기능을 포함하고 있다.

4.3 선형대수연산 및 수치알고리듬 모듈

모든 최적화 알고리듬은 LinearAlgebra 클래스를 상속받은 Numeric- Algorithm 클래스를 상속받아 LinearAlgebra 클래스와 NumericAlgorithm 클래스의 함수 및 계수를 사용할 수 있다.

1) LinearAlgebra 클래스

최적화 알고리즘의 가장 상위의 모듈로서 함수의 gradient, 함수의 hessian, 벡터의 norm, 행렬의 norm, 단위행렬 생성 등의 기능을 제공한다.

2) NumericAlgorithm 클래스

수치적 알고리듬의 공통 계수 및 수렴판정 계수를 포함하고 있으며 수렴 판정조건 함수를 제공하며 알고리듬의 내력 데이터를 저장하는 기능을 가지고 있다.

4.4 최적화 알고리듬 모듈

Vector 클래스와 Matrix 클래스의 자료형을 사용하면서 Numeric- Algorithm 클래스를 상속받은 각각의 최적화 알고리듬 클래스는 기본적인 연산기능과 공통 함수들의 재정의의 필요없이 알고리즘의 구현만 하면 된다. 모든 최적화 알고리듬 클래스는 탐색과정함수와 파라미터 조정함수를 포함하고 있다.

1) EqualInterval 클래스

등간격 탐색법(equal interval search)의 모듈이다.

2) GoldenSection 클래스

황금분할법(golden section seach)의 모듈이다.

3) PInterpolation 클래스

이차 다항식 보간법(quadratic interpolation)의 모듈이다.

4) SteepestDescent 클래스

최속강하법(steepest descent method)의 모듈이다. SteepestDescent 클래스 및 이하 BFGSMethod 클래스까지의 선탐색과정은 황금분할법을 사용하였으며 목적함수의 경사도는 중앙차분법[17]을 사용하였다.

5) ConjugateGradient 클래스

공액경사도법(conjugate gradient method)의 모듈이다.

6) NewtonMethod 클래스

수정된 뉴턴의 방법(modified Newton's method)의 모듈이다.

7) DFPMethod 클래스

DFP 방법의 모듈이다.

8) BFGSMethod 클래스

BFGS 방법의 모듈이다.

9) Simplex 클래스

Nelder-Mead의 심플렉스 방법의 모듈이다.

10) Complex 클래스

Box의 콤플렉스 방법의 모듈이다.

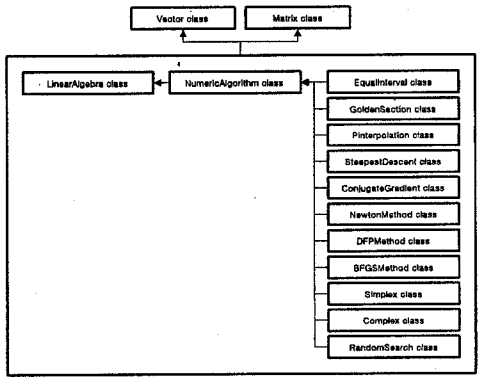

Fig.5 Hierachy chart of class module.

11) RandomSearch 클래스
난수 탐색법의 모듈이다.

Fig. 5는 클래스 모듈들의 계층도를 나타내었다.

5. 최적화 알고리듬 모듈의 검증

5.1 시험함수

More, Garbow, Hillstrom[15]은 최적화 알고리즘 및 다양한 수치해석 알고리즘들의 성능을 평가할 수 있는 시험함수들을 수집하였다. More 등은 이 함수들과 초기 시작점, 최적해 등을 완전히 정리해 놓았다. 이러한 18개 함수들은 개발된 알고리듬의 시험을 위한 비구속 최적화 문제들이며 NETLIB의 MINPACK Collection[16]을 통하여 이용이 가능하다. Table 2에서 시험함수들을 정리하여 나타내었다.

5.2 알고리듬의 검증

모듈화된 알고리듬의 성능을 검증하기 위하여 More 등의 시험함수에 대한 결과를 잘 알려진 공학계산용 소프트웨어 MATLAB의 OPTIMIZATION TOOLBOX[17]에서 제공하는 함수의 결과와 비교하였다. More 등의 시험함수에 대한 수렴된 해의 정확도에 대하여 주요 알고리듬을 비교하였다. 검증은 개발된 알고리듬의 계수 및 조건을 최대한 MATLAB과 일치시켜 Pentium II Processor에서 실시하였다.

실험의 결과 C++ 코드로 개발된 라이브러리는 수렴된 최적해에서 MATLAB과 큰 차이를 보이지

Table 2. MINPACK test function from NETLIB

Test Function	
1	Helical Valley Function
2	Big Exp6 Function
3	Gaussian Function
4	Powell Badly Scaled Function
5	Box 3-Dimensional Function
6	Variably Dimensioned Function
7	Watson Function
8	Penalty 1 Function
9	Penalty 2 Function
10	Brown Badly Scaled Function
11	Brown and Dennis Function
12	Gulf Research and Development Function
13	Trigonometric Function
14	Extended Rosenbrock Function
15	Extended Powell Singular Function
16	Beale Function
17	Wood Function
18	Chebyquad Function

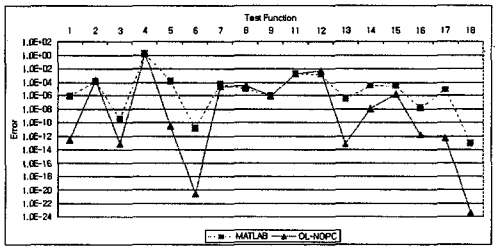

Fig.6 The comparison of true error on testing steepest descent method

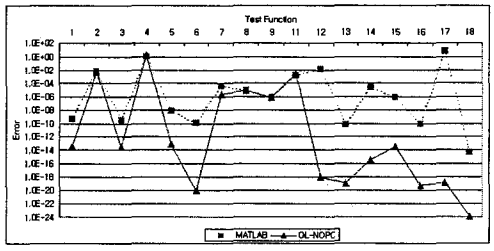

Fig.7 The comparison of true error on testing DFP method

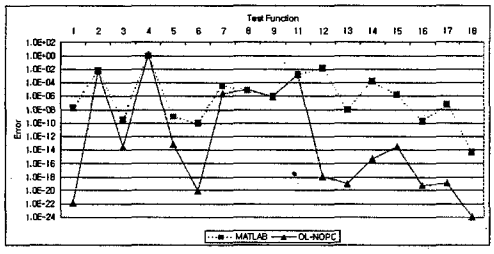

Fig.8 The comparison of true error on testing BFGS method

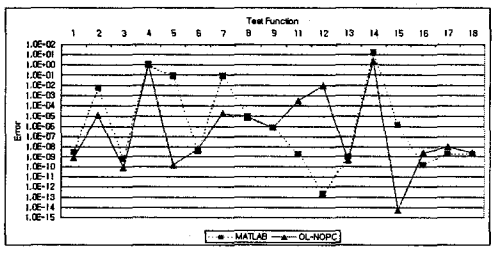

Fig.9 The comparison of true error on testing simplex method

않았으며 오히려 여러 함수에 대하여 정밀한 최적해를 도출하였다. 참해에 대한 MATLAB과 개발한 라이브러리(OL-NOPC)의 오차를 비교해본 결과 개발한 라이브러리는 함수 10번에 대해서만 심플렉스 방법을 제외한 나머지 방법들이 부적절한 해를 도출하였다. 그러나 나머지 17개의 함수에 대해서는 MATLAB과 비교하여 많은 함수에 대해서 우등한 최적해를 산출하거나 근사적이었다(부록의 Table 3, 4, 5, 6 참조). 따라서 개발된 라이브러리는 최적해의 정밀도면에서 신뢰할 수 있다고 판단된다. Fig.6, 7, 8, 9는 각각의 시험함수에 대한 MATLAB과 개발된 알고리듬(OL-NOPC)의 최속경사법, DFP 방법, BFGS 방법, 심플렉스 탐색법을 적용한 수렴된 최적해의 절대오차를 나타내었다.

6. 결 론

최적화는 공학, 특히 설계의 분야에서 자주 접하게 되는 문제임에도 최적화 소프트웨어는 대형 패키지의 일부이거나 Fortran 라이브러리여서 접근하기가 어렵고 현재의 프로그래밍 개발실정에 맞질 않는 문제점들이 있다. 본 연구에서는 이러한 문제점들을 개선하기 위하여 주요 최적화 알고리듬들의 C++ 라이브러리를 개발하였다. 개발한 라이브러리는 NETLIB에서 제공하는 시험함수를 통하여 검증을 하였다. 이상의 연구에 대한 주요 결과 및 결론을 요약하면 다음과 같다.

(1) 주요 최적화 알고리듬들의 수학적인 자료형 구조에 관한 체계적인 계층도를 구성하여 C++ 클래스로서 모듈화하였다.

(2) 모듈화된 알고리듬들에 대하여 시험함수를 가지고서 MATLAB과 비교하였다. 비교한 결과, 대부분의 시험함수에 대해서 MATLAB보다 오차가 적은 최적해를 산출하였다.

(3) 모듈화된 알고리듬들을 최적화 프로그래밍에서 사용되어질 수 있도록 C++ 라이브러리로 개발하였다.

부 록

Table 3. Result of steepest descent method

Test Function	MATLAB	OL-NOPC
1	9.06709716e-07	2.99181342e-13
2	1.44912298e-04	2.15832850e-04
3	1.16177934e-08	1.12793794e-08
4	1.87000038e+00	1.87054705e+00
5	1.38115071e-04	3.20504621e-11
6	1.67990973e-11	2.98467858e-21
7	6.11142837e-05	2.73086481e-05
8	9.08338533e-06	3.12973840e-05
9	8.24811493e-07	8.06639047e-07
10	5.68539425e+04	2.49989639e+11
11	8.58222016e+04	8.58222016e+04
12	1.54665741e-03	5.11504774e-03
13	2.96760244e-07	5.98332675e-14
14	3.22811786e-05	1.29688839e-08
15	2.83121554e-05	1.54328892e-06
16	1.71516124e-08	1.37234480e-12
17	7.79925792e-06	5.06219943e-13
18	7.28204494e-14	4.44687223e-24

Table 4. Result of DFP method

Test Function	MATLAB	OL-NOPC
1	5.22126597e-10	3.62948327e-14
2	5.64875976e-03	5.65564994e-03
3	1.16226280e-08	1.12793294e-08
4	9.99809710e-01	1.87054546e+00
5	7.12589912e-09	7.54725264e-14
6	1.20511318e-10	9.51136708e-21
7	4.09986232e-05	2.35223074e-06
8	9.08259483e-06	8.35778675e-06
9	8.24268405e-07	8.06639004e-07
10	1.94364930e-05	2.49998046e+11
11	8.58222016e+04	8.58222025e+04
12	1.30948929e-02	8.45117885e-19
13	6.43588741e-11	1.21360880e-19
14	2.59579578e-05	3.42637947e-16
15	7.72635685e-07	3.54192757e-14
16	7.13506767e-11	6.19930754e-20
17	6.54764534e+00	1.26220081e-19
18	4.83379686e-15	1.37093416e-24

Table 5. Result of BFGS method

Test Function	MATLAB	OL-NOPC
1	1.75798693e-08	1.44833003e-22
2	5.65598307e-03	5.65564994e-03
3	1.16226234e-08	1.12793294e-08
4	9.99887561e-01	1.87054990e+00
5	8.52684856e-10	7.45488877e-14
6	1.00874628e-10	9.51136708e-21
7	4.04064431e-05	2.35063612e-06
8	9.08259483e-06	8.35778080e-06
9	8.14362151e-07	8.06639004e-07
10	2.33263150e-05	2.49998046e+11
11	8.58222016e+04	8.58222017e+04
12	1.25877426e-02	1.29501432e-18
13	1.03787351e-08	1.09406943e-19
14	1.77855913e-04	5.76890831e-16
15	1.47521216e-06	2.79888662e-14
16	1.81402152e-10	6.23998181e-20
17	4.77808480e-08	1.28032714e-19
18	4.83379689e-15	1.37093416e-24

Table 6. Result of simplex search method

Test Function	MATLAB	OL-NOPC
1	3.26234198e-09	7.94571070e-10
2	5.65564993e-03	1.17243855e-05
3	1.18891931e-08	1.13621906e-08
4	9.99787225e-01	9.99787225e-01
5	7.55887408e-02	1.56796711e-10
6	4.04407787e-09	6.85023120e-09
7	7.72447168e-02	1.75084762e-05
8	8.35778087e-06	8.35778084e-06
9	8.07873770e-07	8.07924166e-07
10	2.00355540e-09	2.80582747e-04
11	8.58222016e+04	8.58222018e+04
12	1.89387430e-13	9.45386164e-03
13	9.58563722e-10	4.53409368e-10
14	1.43161965e+01	2.50802265e+00
15	1.39058605e-06	5.53429385e-15
16	1.39263183e-10	2.70891568e-09
17	1.94483362e-09	1.11479500e-08
18	1.95993693e-09	2.51682443e-09

참 고 문 헌

[1] J. S. Arora, *Introduction to optimum design*, McGraw-Hill, New York, 1989.

[2] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," Vol.7, No.2, pp.149-154, 1964.

[3] W. C. Davidon, "Variable metric methods for minimization," *Technical Report ANL-5990*, Argonne National Labs, 1959.

[4] R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *The Computer Journal*, Vol.6, pp.163-168, 1963.

[5] C. G. Broyden, "The convergence of a class of double-rank minimization algorithms, Part I and II," *Journal of the Institute of Mathematics and Its Applications*, Vol.6, pp.76-90, pp.222-236, 1970.

[6] R. Fletcher, "A new approach to variable metric algorithms," *The Computer Journal*, Vol.13, pp.317-322, 1970.

[7] D. Goldfarb, "A family of variational methods derived by variational means," *Mathematics of Computation*, Vol.24, pp.23-26, 1970.

[8] D. F. Shanno, "Conditioning of quasi-Newton methods for function minimization," *Mathematics of Computation*, Vol.24, pp.647-657, 1970.

[9] R. M. Lewis, V. Torczon, and M. W. Trosset, "Direct search methods: Then and now,"

[10] Nelder, J. A. and R. Mead, "A Simplex Method for Function Minimization," *Computer J.*, Vol.7, pp.308-313, 1965.

[11] Spendley, W., G. R. Hext, and F. R. Himsworth, "Sequential Application of Simplex Designs in Optimisation and Evolutionary Operation," *Technometrics*, Vol.4, pp.441-461, 1962.

[12] R. R. Barton, J. S. Ivey Jr, "Nelder-Mead simplex modifications for simulation optimization," Management Science, Vol.142 issue 7, pp.954, 20p, Jul 1997.

[13] Box, M. J., "A new method of constrained optimization and a comparison with other methods," *The Computer Journal*, Vol.8, pp.42, 1965.

[14] S. S. Rao, *Engineering optimization*, 3rd ed, Wiley, New York, 1996.

[15] More, J. J., B. S. Garbow, and K. E. Hillstrom, "Testing Unconstrained Optimization Software," *ACM Trans. Math. Software*, Vol.7, pp.17-41, 1981.

[16] Dongarra, J. J. and E. Grosse, "Distribution of Mathematical Software via Electronic Mail," *Comm. ACM*, Vol.30, pp.403-407, 1987.

[17] T. Coleman, M. A. Branch, and A. Grace, *Optimization Toolbox user's guide*, The MathWorks Inc., 1999