컴퓨터 비전에서의 최소화 기법의 고찰과 성능 비교†

박종승° 한준희
포항공과대학교 전자계산학과 컴퓨터비전 연구실

A Survey and Performance Comparison of Minimization Techniques in Computer Vision

J. S. Park J. H. Han
Computer Vision Lab, Dept. of Comp. Sci. & Eng., POSTECH

요 약

컴퓨터 비전에서의 여러 문제를 해결하는데 있어서 최소화 기법이 많이 사용되지만 구체적인 구현 방법이나 성능 평가에 대한 자료가 미흡하다. 본 논문에서는 다른 연구자들이 기법을 선택하는데 참고가 되도록 널리 사용되는 최소화 기법들의 방법, 문제점, 구현, 성능 등에 관하여 기술한다.

1. 서론

컴퓨터 비전에서의 최소화 기법은 매우 중요하다. 최소화 문제는 제약화된 최적화와 무제약 최적화의 두가지 종류로 나눌 수 있다. 제약화된 최적화 문제는 라그랑제 승수를 사용하여 무제약 최적화 방법으로 극값을 구할 수 있다. 본 논문에서는 무제약 최적화 카테고리에 속하는 방법들을 소개한다. 이들 방법을 알아보고 성능을 비교한다. 방법들의 구체적인 내용은 [1]에 기술되어 있다.

2. 선형 최소자승 최소화 방법

선형 최소자승 최적 해를 찾기 위해서 모델은

$$y = g(x,a) = \sum_{k=1}^{m} a_k g_k(x) \tag{1}$$

의 형태로 나타낼 수 있어야 한다. 여기서 g_k는 x에 관한 임의의 함수로 다항식이 아닌 비선형 함수일 수도 있다. 목적함수는

$$E = \sum_{i=1}^{n} f_i^2 = \sum_{i=1}^{n} [y_i - g(x_i;a)]^2$$

로 둘 수 있다. $A_{ij} = g_j(x_i)$ 로 크기가 $n \times m$인 행렬 A를 만들고 $b_i = y_i$ 로 크기가 n인 벡터 b를 만든다. 파라메터들을 벡터형태 $a = (a_1, \cdots, a_m)^T$ 로 표시한다.

최소 해는 모든 a_i에 대한 E의 도함수들이 0이 될 때이므로 $0 = \sum_{i=1}^{n} [y_i - g(x_i,a)]\, g_k(x_i) \quad (k = 1, \cdots, m)$ 의 식을 얻는다. 이 식들을 행렬 형태로 정리하면

$$(A^T A)a = A^T b \tag{2}$$

이다. 식 (2)을 최소자승 문제의 정규식(normal equation)들이라고 한다. LU 분해, Choleksy 분해, Gauss-Jordan 소거 등의 선형대수 방법중 하나를 사용하여 식 (2)을 풀수 있다. 각각의 방법은 약간의 장단점들을 가진다. 에러 공분산 행렬을 구하고 싶을 경우 Gauss-Jordan 소거법을 사용해야 한다. 단지 a만 구하고 싶을 경우에는 역함수의 계산이 필요없는 LU 분해 방법을 사용하여 계산시간을 줄일 수 있다.

정규식을 직접 풀어서 얻은 해는 round-off 에러때문에 상당히 틀릴수가 있다. 이런 경우의 해결책은 QR 분해나 SVD 방법을 사용하는 것이다. 정규식들이 singular에 가까울 경우에는 식을 푸는 과정에서 0에 가까운 수로 나누는 과정이 포함되기 때문에 해는 매우 불안정하게 된다. 정규식들이 singular에 가깝게 되는 경우는 실제적으로 흔히 발생한다. 이는 데이타에 중복된 요소들이 자주 포함되므로 행렬 A가 singular에 가깝게 되기 때문이다. 이런 overdetermined 문제에서 흔히 발생되는 경우를 해결할 수 있는 방법이 SVD이다. overdetermined 시스템에서 SVD는 최소자승 해에 가장 좋은 추정치를 제공한다. 또한 SVD는 underdetermined 시스템에서도 최소자승 해보다 에러가 더 작은 해를 제공한다. 따라서 선형으로 최소자승 해를 구하는 경우에서는 SVD를 사용하는 것이 가장 좋은 방법이다.

3. 비선형 최소화 방법

최적화할 파라메터의 갯수가 m이라고 하고 이를 벡터 $a = (a_1, a_2, \cdots, a_m)^T$ 로 표시하자. 목적함수를 $E(a)$ 로 두면 최적화 문제는 $\min E(a)$ 이다. $E(a)$가 2차 미분가능하면 $\tilde{a}$가 $E(a)$의 국부 최소치가 되기 위해서는 다음 두 조건을 만족해야 한다.

- 1차 도함수들인 gradient $\nabla E(a)$가 $\tilde{a}$에서 0이다.

$$\left.\frac{\partial E}{\partial a_i}\right|_{\tilde{a}} = 0 \quad (i = 1, \cdots, m) \tag{3}$$

- a에 대한 E의 2차 도함수로 $H_{ij} = \frac{\partial}{\partial a_i}\frac{\partial E}{\partial a_j}$ $(i = 1, \cdots, m,\ j = 1, \cdots, m)$ 와 같이 구해지는 Hessian 행렬 H가 positive semi-definite이다.

만약 $E(a)$가 미분가능하고 convex 함수이면 $\nabla E(a) = 0$ 의 해가 국부 최소치가 되는 필요충분 조건이다. $\nabla E(a)$를 수식적으로 구할 수 있는 경우에는 쉽게 해를 찾을 수 있다. 그러나 실제 최적화 문제에서는 목적함수 $E(a)$가 데이터에 따라 달라지므로 $\nabla E(a)$의 명확한 수식을 얻을 수 없다. 그 대신 $\nabla E(a)$를 수치적으로 그리고 반복적으로 얻어야 한다. 대부분의 비선형 최소화 기법들의 방법은 다음과 같은 형태를 가진다.

- 단계 1. 추정할 파라메터의 초기치 a_0를 설정한다.
- 단계 2. $E(a(k+1)) < E(a(k))$ 가 되도록

$$a(k+1) = a(k) + \lambda d(k)$$

로 $a(k+1)$를 구한다.

† 본 연구는 한국과학재단 지정 지능자동화 연구센터로부터 연구비의 일부를 지원받았음

- 단계 3: $a(k)$를 $a(k+1)$로 갱신한 후 위의 작업을 수렴이 될 때까지 반복한다.

여기서 $d(k)$은 현재의 파라메터가 이동할 방향을 나타내고 λ는 이동하는 크기를 나타낸다. 최적화 문제의 핵심은 $d(k)$와 λ를 어떻게 결정하는가에 있다

3.1. Steepest descent 방법

Steepest descent 방법은 매 반복마다 gradient 방향의 반대방향으로 이동하는 것이다. 즉

$$a(k+1) = a(k) - \lambda \nabla E(a(k)) \tag{4}$$

이다 단계 2를 수행한 후 $E(a(k+1))$이 $E(a(k))$보다 크지면 λ를 반으로 줄이고 단계 2를 다시 계산하고, 그렇지 않으면 단계 3으로 진행한다. 이 방법의 단점으로서 만약 초기치가 진짜 해에 충분히 가깝지 않으면 round-off 에러 때문에 진짜 해와 전혀 다른 곳으로 수렴한다

3.2. Gauss-Newton 방법

Gauss-Newton 방법은 함수의 자승들의 합으로 표현되는 목적함수를 최적화하기에 적당하다. 모델이 $y_i = g(x_i, a)$ 이라면 목적함수는

$$E(a) = \sum_{i=1}^{n} f_i^2(a), \text{ where } f_i = y_i - g(x_i, a)$$

의 형태이다 $E(a)$의 1차 도함수들로 $n \times m$ Jacobian 행렬 J를

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial a_1} & \frac{\partial f_1}{\partial a_2} & \cdots & \frac{\partial f_1}{\partial a_m} \\ \frac{\partial f_2}{\partial a_1} & \frac{\partial f_2}{\partial a_2} & \cdot & \frac{\partial f_2}{\partial a_m} \\ & & \cdot & \\ \frac{\partial f_n}{\partial a_1} & \frac{\partial f_n}{\partial a_2} & \cdot\cdot & \frac{\partial f_n}{\partial a_m} \end{bmatrix} \tag{5}$$

로 정의한다 $f = (f_1, f_2, \quad , f_n)^T$ 로 두면

$$\nabla E = 2J^T f$$

이다 2차이상의 도함수들을 무시하면 Hessian 행렬 H는

$$H \approx 2J^T J$$

로 추정할 수 있으므로

$$a(k+1) = a(k) - H^{-1} \nabla E(a(k))$$

로 표시되는 Newton 식은

$$a(k+1) = a(k) - (J_k^T J_k)^{-1} J_k^T f(a(k)) \tag{6}$$

이다 여기서 J_k은 J를 $a(k)$에서 계산한 행렬이다 Gauss-Newton 방법이 Newton 방법과 다른점은 Hessian 행렬이 1차 도함수만으로 계산된다는 점이다

Gauss-Newton 방법에도 치명적인 문제점이 있다. 첫째, $E(a(k+1))$이 $E(a(k))$보다 크질 경우가 있으므로 수렴을 보장하지 못한다. 둘째, 행렬 $J^T J$가 거의 singular에 가깝게 되면 식 (6)을 계산할 수 없다 또다른 문제점으로, 식 (6)에 의한 변화량이 너무 커서 파라메터 값들이 수렴가능한 범위를 벗어날 수 있다.

3.3. Levenberg-Marquardt 방법

Levenberg-Marquardt 방법은 $E(a(k+1)) < E(a(k))$ 을 보장하여 Gauss-Newton 방법에서의 문제점들을 해결한 방법이다. 식 (6)의 갱신 식에서 $J_k^T J_k$ 에 추가적인 항을 더해서 singularity 문제를 해결한다 즉 갱신식은

$$a(k+1) = a(k) - (J_k^T J_k + \lambda I)^{-1} J_k^T f(a(k)) \tag{7}$$

이 된다 $\lambda = 0$ 이면 Gauss-Newton 방법과 동일하다 $\lambda \to \infty$ 이면 steepest descent 방법과 유사해지며 이동 크기는 0에 가깝게 된다 따라서 λ를 증가시키면 $E(a(k+1)) < E(a(k))$를 항상 보장할 수 있게 된다

3.4. Levenberg-Marquardt 알고리즘

먼저 목적함수 또는 에러함수

$$E(a) = \sum_{i=1}^{n} f_i^2(a) \tag{8}$$

를 정한다 도함수를 알 수 있으면 도함수를 구한다. 도함수를 알지 못해도 상관없다 비교적 정확한 도함수를

$$\frac{\partial f}{\partial a_i} = \frac{f(a') - f(a)}{\delta}$$

와 같이 수치적으로 계산할 수 있다 여기서 $\delta = \min\left\{|10^{-4} a_i|, 10^{-6}\right\}$ 이고 a'은 $a_i' = a_i + \delta$ and $a_j' = a_i$ for $i \neq j$ 으로 정의된다.

최적화 알고리즘은 다음과 같다

- 단계 1: $\lambda = 10^{-3}$, 초기치 $a(0)$를 설정한다 ;
- 단계 2: $E(a(k))$를 계산한다;
- 단계 3. $a(k)$에서 정의되는 식 (5)의 Jacobian 행렬 J_k를 계산한다;
- 단계 4: 식 (7)을 사용하여 $a(k+1)$을 계산한다;
- 단계 5 $E(a(k+1))$를 계산한다;
- 단계 6: 만약 $E(a(k+1)) > E(a(k))$ 이면 $\lambda = 10 * \lambda$ 로 바꾸고 단계 4로 간다;
- 단계 7 수렴 여부를 다음과 같이 결정한다; 만약 $E(a(k)) < 10^{-4}$ 이면 수렴으로 판정하고 반복을 중지한다, 만약 $E(a(k)) - E(a(k+1)) < 10^{-3}$ 이 연속적으로 4번 이상 일어나면 수렴으로 판정하고 반복을 중지한다;
- 단계 8 $\lambda = 0.1 * \lambda$;
- 단계 9 $a(k) = a(k+1)$,
- 단계 10. 단계 2로 간다,

단계 1에서의 10^{-3}은 대부분의 구현에서 사용되는 경험적인 값이다 단계 7에서의 10^{-4}와 10^{-3}는 사용자가 적당하게 결정할 수 있는 값이다. 위의 알고리즘은 Levenberg-Marquardt 방법이다. Steepest descent 방법과 Gauss-Newton 방법은 단계 4에서 사용되는 식 (7)을 각각 식 (4)과 식 (6)으로 대치하면 된다. λ의 변경식도 적당히 바꿀 수 있다.

4. 성능 비교

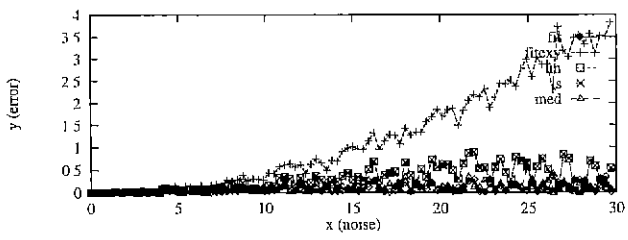

그림 1: 직선의 추정시의 에러

대표적인 몇가지 방법을 구현하여 성능을 비교하였다.

- `ls` : 최소자승 해를 선형으로 추정, Gauss-Jordan 소거법을 이용
- `svd` : 최소자승 해를 선형으로 추정, SVD를 이용

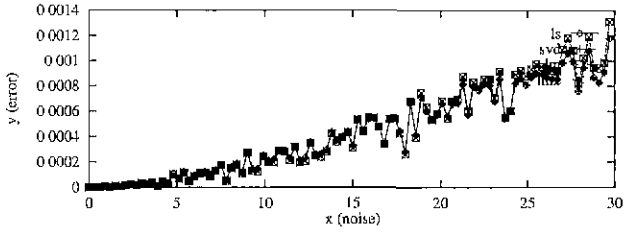

그림 2: 타원의 추정시의 에러

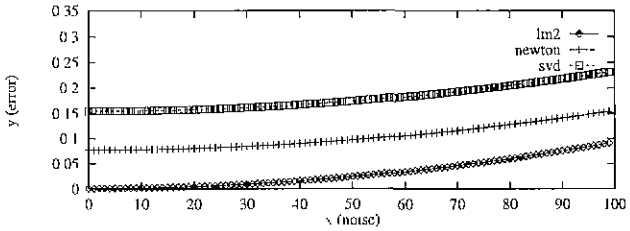

그림 3: 캘리브레이션 파라메터 추정시의 에러

- lm 최소자승 해를 비선형으로 추정; Levenberg-Marquardt 방법; 도함수가 주어짐
- lm2 : 최소자승 해를 비선형으로 추정; Levenberg-Marquardt 방법; 도함수를 수치적으로 계산함

직선 $y = ax + b$ 에 대한 추정으로 성능을 비교해 보았다. 직선은 수식이 매우 간단해서 다른 여러가지 방식들이 적용가능하다. 위의 4가지 방식 외에 [1]에 구현되어 있는 fit, fitexy, med의 3가지를 추가하여 성능을 비교하였다. fit와 fitexy는 에러 분산을 최소화하는 선형식을 푸는 방법으로 fit는 두 좌표축의 에러 분산을 동일하게 가정하였고 fitexy는 두 좌표축의 분산을 다르게 가정하였다. med는 median 연산을 사용한 강건추정이다.

노이즈가 증가함에 따라서 fitexy가 가장 나쁜 성능을 보였고 그 외의 방법은 거의 동일한 성능을 보였으며 그 중에서 med와 fit이 가장 좋은 성능을 보였다. 그림 1는 노이즈에 대한 에러의 그래프를 나타낸다. 이는 fitexy의 수식이 복잡하여 round-off 에러가 fit에 비해 상대적으로 많기 때문이다. ls와 svd는 완전히 동일한 성능을 보였고, lm와 lm2도 완전히 동일한 성능을 보였고, lm와 ls도 거의 동일한 성능을 보였다. 즉 에러는

$$\mathtt{med}, \mathtt{fit} < \mathtt{lm}, \mathtt{lm2} \approx \mathtt{ls}, \mathtt{svd} << \mathtt{fitexy}$$

의 순으로 나타났다.

목적함수가 복잡해지면 상황은 많이 달라진다. 비선형 함수를 최소화하는데 사용가능한 방법은 최소자승 방법, Steepest descent 방법, Gauss-Newton 방법, Levenberg-Marquardt 방법, 강건추정 방법 등이 있다. Steepest descent 방법과 Gauss-Newton 방법은 Levenberg-Marquardt 방법보다 좋을 수 없으므로 최소자승 방법과 Levenberg-Marquardt 방법을 비교한다. 강건추정 방법은 응용에 따라서 그리고 어떤 분포함수를 사용하는가에 따라서 성능의 차이가 나기 때문에 생략한다.

기울어진 타원을 추정하는 예를 제시한다. 중심이 원점에 있고 x축 방향으로 중심에서의 길이가 a_1이고 y축 방향으로 중심에서의 길이가 a_2인 타원의 식은 $\frac{x^2}{a_1^2} + \frac{y^2}{a_2^2} - 1 = 0$ 이다. θ를 x-축과 타원의 주축과의 각도라고 하고 (x_0, y_0)를 타원의 중심이라고 하면 원하는 타원의 식은

$$f = \frac{x'^2}{a_1^2} + \frac{y'^2}{a_2^2} - 1 = 0 \qquad (9)$$

으로 표시된다. 여기서 $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix}$ 이다. 목표는 데이터로부터 식 (9)을 최소화하는 5개의 파라메터 $a_1, a_2, x_0, y_0, \theta$ 를 추정하는 것이다. Levenberg-Marquardt 방법은 식 (9)을 직접 최소화할 수 있으나 최소자승 방법은 비선형 함수들의 합의 형태로 바꾸어야 한다. 이를 위해서 식 (9)을

$$f = k_1 x^2 + k_2 y^2 + k_3 xy + k_4 x + k_5 y + k_6 = 0$$

로 고치고 $k_1, \cdots, k_6$ 를 추정한 다음 실제 파라메터들을 계산한다. 이 경우 ls, svd, lm, lm2의 네개의 방법이 거의 동일한 성능을 나타내었다. 즉, 에러는

$$\mathtt{lm} \approx \mathtt{lm2} \approx \mathtt{ls}, \mathtt{svd}$$

의 순으로 나타났다. 실험을 위해 노이즈를 임의로 발생시켰다. 노이즈는 타원상의 점들의 좌표에 임의의 수를 더하는 방법으로 발생시켰다. 임의의 수는 정규분포(분산이 1이고 평균이 0인 가우시안 분포)의 확률로 무작위수를 발생시키고 이 수들에 정수배(노이즈 레벨)를 곱하여 생성된다. 그림 2는 노이즈에 대한 에러의 그래프를 나타낸다.

캘리브레이션 실험을 통한 최적화의 성능을 비교하였다. svd와 lm2 외에 Gauss-Newton 방법인 newton을 추가하여 비교하였다. 캘리브레이션의 구체적인 내용과 목적함수는 [2]을 참조하였다. 추정할 파라메터는 11개로서, 5개의 카메라 내부 파라메터와 6개의 카메라 외부 파라메터(이동을 위한 3개의 파라메터와 회전을 위한 3개의 파라메터)이다. 데이타는 300개의 영상점들이다. 그림 3는 노이즈에 대한 에러를 나타낸다. x축의 의미는 화소단위로 첨가된 노이즈의 전체 합이다. 실제 영상점들로 실험하여서 파라메터의 참값을 알수 없으므로 역투시 에러의 합을 에러로 정의하고 실험하였다. 그래프에서 각 축의 값은 상대적으로만 의미를 가진다고 보아야 한다. 이 경우 에러는

$$\mathtt{lm} \approx \mathtt{lm2} < \mathtt{newton} < \mathtt{svd}$$

의 순으로 나타났다.

5. 결론

특정한 하나의 최소화 방법이 가장 우수하다고 단정할 수는 없다. 목적함수의 형태, 데이타의 특성과 크기, 차원의 크기 등의 요소에 따라서 결과는 달라질 수 있다. 그러나 실험적으로 볼때 다음과 같이 최소화 방법을 선택하는것이 좋을 결과를 낼 확률이 크다.

```
if "목적함수가 선형인가?"
  then svd
  else
      if "함수 형태를 쉽게 식 (1)의 형태로 쉽게 바꿀 수 있는가?"
        then svd
        else
            if "목적함수의 도함수의 계산이 매우 간단한가?"
              then lm
              else lm2
```

문제가 svd 방법이 적용 가능한 형태, 즉 식 (1)의 형태이면 svd 방법을 사용하는것을 권장한다. 대부분의 문제는 lm2에 해당할 것이다. 실제로 Levenberg-Marquardt 방법은 가장 일반적으로 쉽게 적용될 수 있는 방법이다. 도함수를 구하지 않아도 목적함수만으로 최소화가 가능한 lm2가 성능 측면에서도 가장 우수한 방법중의 하나로 판단되었다.

여러가지의 실험 예제를 웹페이지
http://falcon postech ac.kr/ipawb/demo/ 에서 자세히 볼 수 있다.

참고문헌

[1] W H Press, S A. Teukolsky, W. T. Vetterling, and B. P Flannery, *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, second ed., 1992.

[2] O. Faugeras and G. Toscani, "The calibration problem for stereo," in *CVPR86*, pp 15-20, 1986