Nonlinear Least Squares

Stephen Boyd

EE103
Stanford University

December 6, 2016

Outline

Nonlinear equations

- set of m nonlinear equations in n unknowns $x_1, \ldots, x_n$:

$$f_i(x_1, \ldots, x_n) = 0, \quad i = 1, \ldots, m$$

- $f_i(x)$ is the ith equation; $f_i(x)$ is the ith residual
- n-vector of unknowns $x = (x_1, \ldots, x_n)$
- write as $f(x) = 0$ where $f : \mathbf{R}^n \to \mathbf{R}^m$, $f(x) = (f_1(x), \ldots, f_m(x))$
- when f is affine, reduces to set of m linear equations
- over- (under-) determined if $m > n$ ($m < n$); square if $m = n$

Nonlinear least squares

- find $\hat{x}$ that minimizes $\|f(x)\|^2 = f_1(x)^2 + \cdots + f_m(x)^2$
- includes problem of solving equations $f(x) = 0$ as special case
- like (linear) least squares, super useful on its own

Optimality condition

- optimality condition: $\nabla \|f(\hat{x})\|^2 = 0$
- any optimal point satisfies this, but points can satisfy this and not be optimal
- can be expressed as $2Df(\hat{x})^T f(\hat{x}) = 0$
- $Df(\hat{x})$ is the $m \times n$ derivative or Jacobian matrix,

$$Df(\hat{x})_{ij} = \frac{\partial f_i}{\partial x_j}(\hat{x}), \quad i = 1, \ldots, m, \quad j = 1, \ldots, n$$

- optimality condition reduces to normal equations when f is affine

Difficulty of solving nonlinear least squares problem

- solving nonlinear equations or nonlinear least squares problem is (in general) *much harder* than solving linear equations
- even determining if a solution exists is hard
- so we will use *heuristic* algorithms
  - not guaranteed to always work
  - but often work well in practice

  (like k-means)

Outline

Computing equilibrium points

- *Equilibrium prices*: find n-vector of prices p for which $S(p) = D(p)$
  - $S(p)$ is supply of n goods as function of prices
  - $D(p)$ is demand for n goods as function of prices
  - take $f(p) = S(p) - D(p)$

- *Chemical equilibrium*: find n-vector of concentrations c so $C(c) = G(c)$
  - $C(c)$ is consumption of species as function of c
  - $G(c)$ is generation of species as function of c
  - take $f(c) = C(c) - G(c)$

Location from range measurements

- 3-vector x is position in 3-D, which we will estimate
- *range* measurements give (noisy) distance to known locations

$$\rho_i = \|x - a_i\| + v_i, \quad i = 1, \ldots, m$$

  - a_i are known locations
  - v_i are noises
- least squares location estimation: choose $\hat{x}$ that minimizes

$$\sum_{i=1}^{m} \left(\|x - a_i\| - \rho_i\right)^2$$

- GPS works like this

Outline

The basic idea

- at any point z we can form the affine approximation

$$\hat{f}(x;z) = f(z) + Df(z)(x-z)$$

- $\hat{f}(x;z) \approx f(x)$ *provided* x is near z
- we can minimize $\|\hat{f}(x;z)\|^2$, using linear least squares
- we'll iterate, with z the current iterate

Levenberg-Marquardt algorithm

- iterates $x^{(1)}, x^{(2)}, \ldots$
- form affine approximation of f at $x^{(k)}$:

$$\hat{f}(x; x^{(k)}) = f(x^{(k)}) + Df(x^{(k)})(x - x^{(k)})$$

- choose $x^{(k+1)}$ as minimizer of

$$\|\hat{f}(x; x^{(k)})\|^2 + \lambda^{(k)}\|x - x^{(k)}\|^2$$

  ($\lambda^{(k)} > 0$)

- we want $\|\hat{f}(x; x^{(k)})\|^2$ small, but we don't want to move too far from $x^{(k)}$, where $\hat{f}(x; x^{(k)}) \approx f(x)$ no longer holds

Adjusting λ

idea:

- if $\lambda^{(k)}$ is too big, $x^{(k+1)}$ is too close to $x^{(k)}$, and progress is slow
- if $\lambda^{(k)}$ is too small, $x^{(k+1)}$ is too far from $x^{(k)}$, and the linearization approximation is poor

update mechanism:

- if $\|f(x^{(k+1)})\|^2 < \|f(x^{(k)})\|^2$, accept iterate and reduce λ:
  $\lambda^{(k+1)} = 0.8\lambda^{(k)}$
- otherwise, increase λ and do not update x:
  $\lambda^{(k+1)} = 2\lambda^{(k)}$ and $x^{(k+1)} = x^{(k)}$

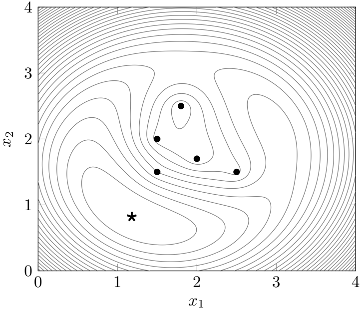
Levenberg-Marquardt iteration

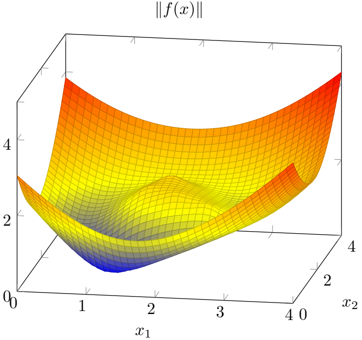
$$x^{(k+1)} = x^{(k)} - \left(Df(x^{(k)})^T Df(x^{(k)}) + \lambda^{(k)} I\right)^{-1} Df(x^{(k)})^T f(x^{(k)})$$

- inverse always exists (since $\lambda^{(k)} > 0$)
- $x^{(k+1)} = x^{(k)}$ only if $Df(x^{(k)})^T f(x^{(k)}) = 0$, *i.e.*, Levenberg-Marquardt stops only when optimality condition holds

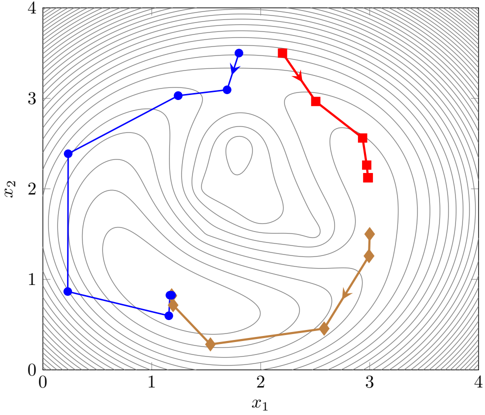
Example: Location from range measurements

range to 5 points (circles)

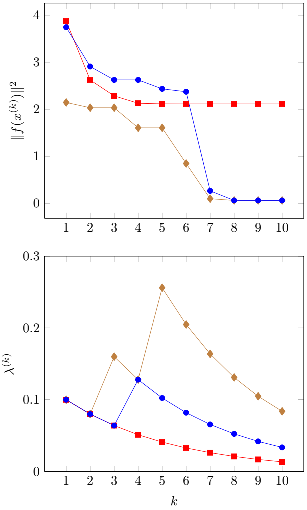




Levenberg-Marquardt from 3 initial points



Levenberg-Marquardt from 3 initial points



Outline

Nonlinear least squares classification

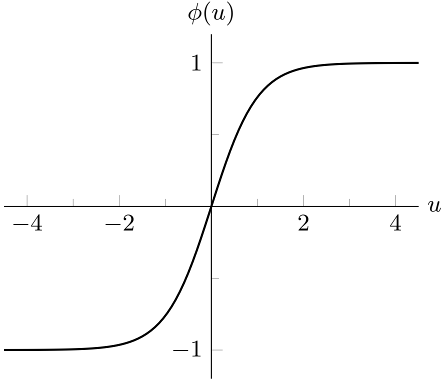
linear least squares classifier:

- $\tilde{f}(x) = \theta_1 f_1(x) + \cdots + \theta_p f_p(x)$
- choose θ to minimize $\sum_{i=1}^{N} (\tilde{f}(x_i) - y_i)^2$ (plus optionally regularization)
- final classifier is $\hat{f}(x) = \mathbf{sign}(\tilde{f}(x))$

nonlinear least squares classifier:

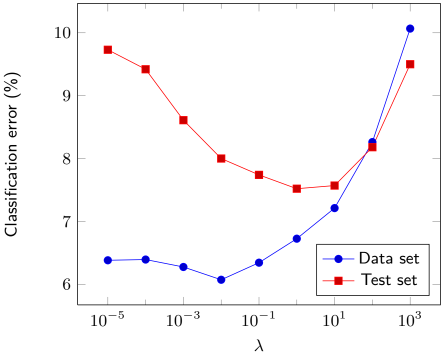
- choose θ to minimize $\sum_{i=1}^{N} (\mathbf{sign}(\tilde{f}(x_i)) - y_i)^2 = 4\times$ number errors
- replace $\mathbf{sign}$ function with smooth approximation ϕ, *e.g.*, sigmoid function $\phi(u) = (e^u - e^{-u})/(e^u + e^{-u})$
- use Levenberg-Marquardt to minimize $\sum_{i=1}^{N} (\phi(\tilde{f}(x_i)) - y_i)^2$

Sigmoid function



Example

- MNIST data set
- linear least squares 10-way classifier: 13.5% test error
- nonlinear least squares 10-way classifier: 7.5% test error



Feature engineering

- add 5000 random features as before
- test set error drops to 2%
- this matches human performance
- with more feature engineering, can substantially beat human performance