LECTURE 17: OVERVIEW OF OPTIMIZATION

STAT 598z: Introduction to computing for statistics

Vinayak Rao

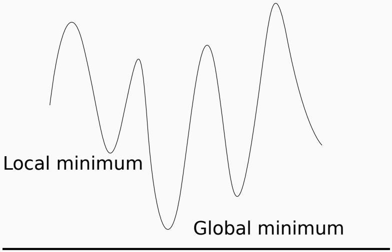
Department of Statistics, Purdue University

March 25, 2019

Global and local minimum

Find minimum of some function $f : \mathbb{R}^D \to \mathbb{R}$.
(maximization is just minimizing $-f$).

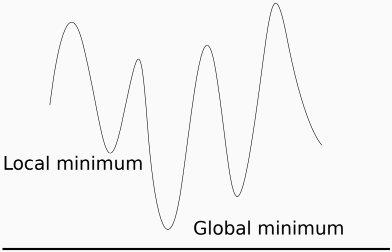
No global information (e.g. only function values, derivatives).



Global and local minimum

Find minimum of some function $f : \mathbb{R}^D \to \mathbb{R}$.
(maximization is just minimizing $-f$).

No global information (e.g. only function values, derivatives).



Finding a global minimum is hard!

We'll settle for a local minimum (maybe with multiple restarts).

Estimating MLE

Consider a set of observations $X = (x_1, \cdots, x_N)$.

Assume $x_i \sim p(x_i|\theta)$

Maximum likelihood:

$$\theta_{MLE} = \text{argmax}\ p(X|\theta) = \text{argmax} \prod_{i=1}^{N} p(x_i|\theta)$$

More convenient to maximize the log-likelihood:

$$\theta_{MLE} = \text{argmax}\ \log p(X|\theta) = \text{argmax} \sum_{i=1}^{N} \log p(x_i|\theta)$$

First-order conditions

The gradient $\nabla f = \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_D} \right]^\top$

First-order conditions

The gradient $\nabla f = \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_D}\right]^\top$

An infinitesimal step along $d\mathbf{x} = [dx_1, \cdots, dx_N]$ gives a change

$$df = \nabla f \cdot d\mathbf{x}$$

First-order conditions

The gradient $\nabla f = \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_D}\right]^\top$

An infinitesimal step along $d\mathbf{x} = [dx_1, \cdots, dx_N]$ gives a change

$$df = \nabla f \cdot d\mathbf{x}$$

∇f: Direction of steepest ascent ($-\nabla f$ is steepest descent).

First-order conditions

The gradient $\nabla f = \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_D}\right]^\top$

An infinitesimal step along $d\mathbf{x} = [dx_1, \cdots, dx_N]$ gives a change

$$df = \nabla f \cdot d\mathbf{x}$$

∇f: Direction of steepest ascent ($-\nabla f$ is steepest descent).

At a local optimum $\nabla f = 0$.

First-order conditions

The gradient $\nabla f = \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_D}\right]^\top$

An infinitesimal step along $d\mathbf{x} = [dx_1, \cdots, dx_N]$ gives a change

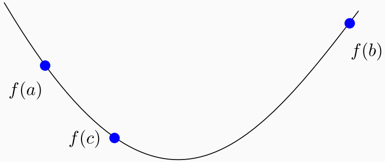
$$df = \nabla f \cdot d\mathbf{x}$$

∇f: Direction of steepest ascent ($-\nabla f$ is steepest descent).

At a local optimum $\nabla f = 0$.

At a local minimum, Hessian $\nabla^2 f \succeq 0$ (positive semidefinite)

$$[\nabla^2 f]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

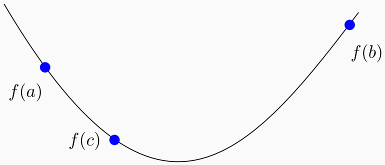


Find a bracket (a, b) with a third point $c \in (a, b)$, with

$$f(a) > f(c) < f(b)$$

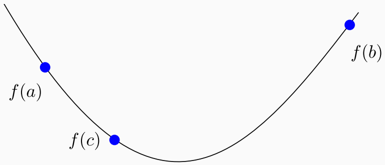
Implies a local minimum lies in (a, b).

One-dimensional minimization



Finding an initial bracketing is not always easy. E.g.

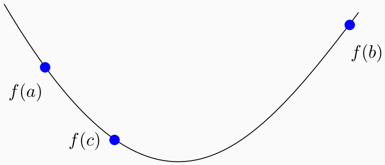
- Pick two points l and r, $l < r$



Finding an initial bracketing is not always easy. E.g.

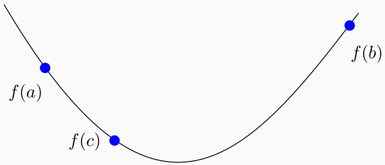
- Pick two points l and r, $l < r$
- If $f(l) < f(r)$, $c = l$ and $b = r$, else $a = l$ and $c = r$.

One-dimensional minimization



Finding an initial bracketing is not always easy. E.g.

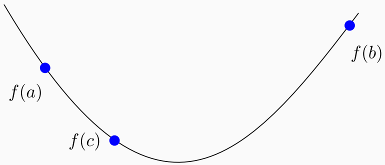
- Pick two points l and r, $l < r$
- If $f(l) < f(r)$, $c = l$ and $b = r$, else $a = l$ and $c = r$.
- In the first case, choose $a < c$, and keep decreasing till $f(a) > f(c)$ (similarly with b for second case)



Having done this, successively refine (a, c) or (c, b).

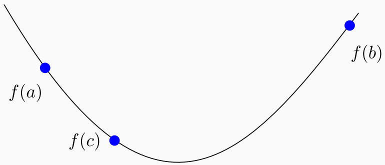
- Pick d in the longer interval (a, c) or (c, b).

One-dimensional minimization



Having done this, successively refine (a, c) or (c, b).

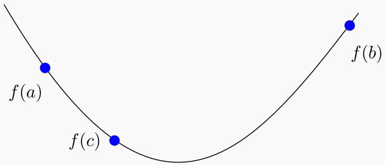
- Pick d in the longer interval (a, c) or (c, b).
- Suppose it is (c, b). Then aither (a, c, d) and (c, d, b) forms a bracket.



Having done this, successively refine (a, c) or (c, b).

- Pick d in the longer interval (a, c) or (c, b).
- Suppose it is (c, b). Then aither (a, c, d) and (c, d, b) forms a bracket.
- Choose and repeat

One-dimensional minimization



Having done this, successively refine (a, c) or (c, b).

- Pick d in the longer interval (a, c) or (c, b).
- Suppose it is (c, b). Then aither (a, c, d) and (c, d, b) forms a bracket.
- Choose and repeat

Doesn't extend easily to higher dimensions.

The simplex algorithm (Nelder & Mead)

Find minimum of some function $f : \mathbb{R}^D \to \mathbb{R}$.

Requires only function evaluations.

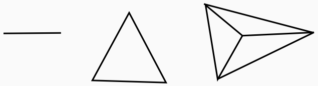
Very general purpose, but not very efficient.

The simplex algorithm (Nelder & Mead)

Find minimum of some function $f : \mathbb{R}^D \to \mathbb{R}$.

Requires only function evaluations.
Very general purpose, but not very efficient.

A 'simplex' in N-dimensions is the convex-hull of $N + 1$ points.
In 1-d: a line segment, 2-d: a triangle, 3-d: a tetrahedron etc.

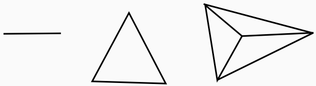


The simplex algorithm (Nelder & Mead)

Find minimum of some function $f : \mathbb{R}^D \to \mathbb{R}$.

Requires only function evaluations.
Very general purpose, but not very efficient.

A 'simplex' in N-dimensions is the convex-hull of $N+1$ points.
In 1-d: a line segment, 2-d: a triangle, 3-d: a tetrahedron etc.



In 1-d we could bracket the minimum.
In higher dims, we must use other heuristics.

The simplex algorithm (Nelder & Mead)

Start with an initial simplex.

Typically, pick an initial point P_0.
Also set $(\mathsf{P}_1, \cdots, \mathsf{P}_{N+1})$ with $\mathsf{P}_i = \mathsf{P}_0 + \lambda_i \mathbf{e}_i$.
Here $\mathbf{e}_i$ is the *i*th coordinate direction, and λ_i is the length-scale in that direction.

Start with an initial simplex.

Typically, pick an initial point P_0.
Also set $(\mathsf{P}_1, \cdots, \mathsf{P}_{N+1})$ with $\mathsf{P}_i = \mathsf{P}_0 + \lambda_i \mathbf{e}_i$.
Here $\mathbf{e}_i$ is the ith coordinate direction, and λ_i is the length-scale in that direction.

Assume $f(\mathsf{P}_0) \leq f(\mathsf{P}_1) \leq \cdots \leq f(\mathsf{P}_{N+1})$.

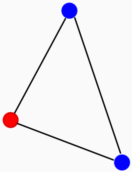
The simplex algorithm (Nelder & Mead)

Start with an initial simplex.

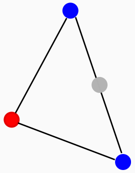
Typically, pick an initial point P_0.
Also set $(\mathsf{P}_1, \cdots, \mathsf{P}_{N+1})$ with $\mathsf{P}_i = \mathsf{P}_0 + \lambda_i \mathbf{e}_i$.
Here $\mathbf{e}_i$ is the ith coordinate direction, and λ_i is the length-scale in that direction.

Assume $f(\mathsf{P}_0) \leq f(\mathsf{P}_1) \leq \cdots \leq f(\mathsf{P}_{N+1})$.

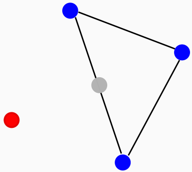
At each step, try to improve the worst point P_{N+1} using one of a sequence of moves.



Get initial simplex

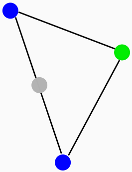


Find worst point, and find centroid of the remaining.

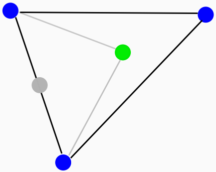


Reflect worst point.

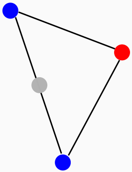
This new point is now either worst, best or in the middle.



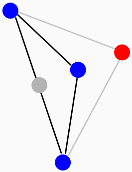
If this is the best point, extend.



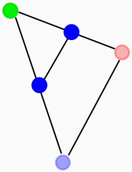
and go back to step one.



If this is the worst point, contract.



If this is the worst point, contract.



Else shrink all points except the best.

Gradient descent

Let x_{old} be our current value

Update x_{new} as $$x_{new} = x_{old} - \eta \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_{old}}$$

The steeper the slope, the bigger the move

Gradient descent

Let x_{old} be our current value

Update x_{new} as $$x_{new} = x_{old} - \eta \left. \frac{\mathrm{d}f}{\mathrm{d}x} \right|_{x_{old}}$$

The steeper the slope, the bigger the move

η: sometimes called the ‘learning rate’
(terminology from the neural network literature)

Gradient descent

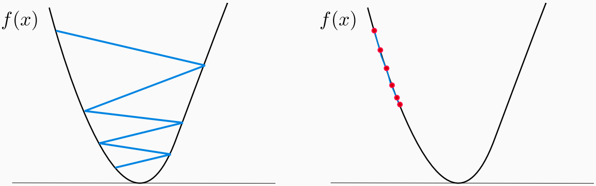
Let x_{old} be our current value

Update x_{new} as $$x_{new} = x_{old} - \eta \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_{old}}$$

The steeper the slope, the bigger the move

η: sometimes called the 'learning rate'
(terminology from the neural network literature)

Choosing η is a dark art:



Gradient descent

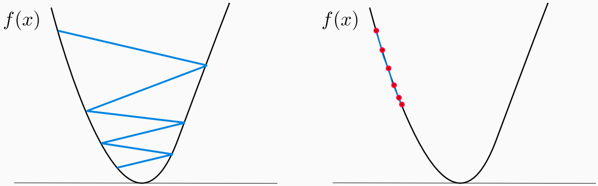
Let x_{old} be our current value

Update x_{new} as $$x_{new} = x_{old} - \eta \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_{old}}$$

The steeper the slope, the bigger the move

η: sometimes called the 'learning rate'
(terminology from the neural network literature)

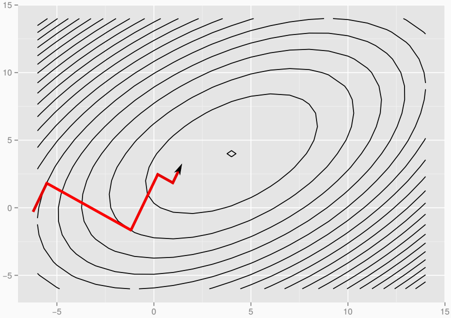
Choosing η is a dark art:



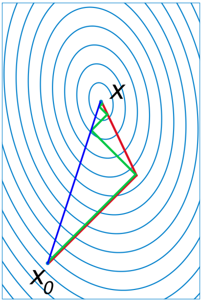
Better methods adapt step-size according to the curvature of f.

Gradient descent applies to higher dimensions too:

$$x_{new} = x_{old} - \eta \left.\nabla f\right|_{x_{old}}$$



Steepest descent



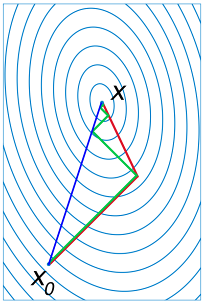
An any iteration, set $\mathbf{p}$ to the direction of steepest descent.

$$\mathbf{p} = \nabla f(x_i)$$

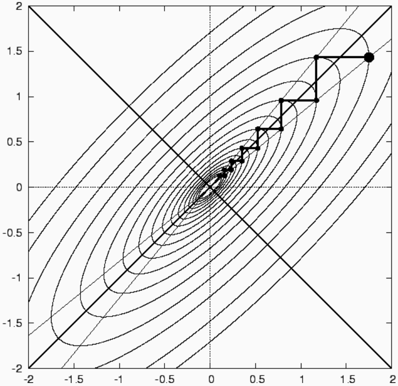
Minimize along that direction: $\lambda_{min} = \text{argmin}_{\lambda} f(\mathbf{x}_i + \lambda \mathbf{p})$

Set $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda_{min}\mathbf{p}$.

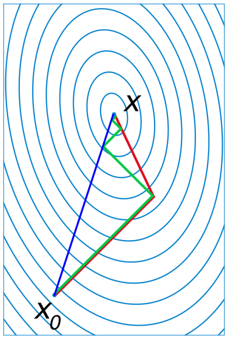
Steepest descent



Can get trapped in long narrow valleys, where successive steps cancel each other.



Can get trapped in long narrow valleys, where successive steps cancel each other.



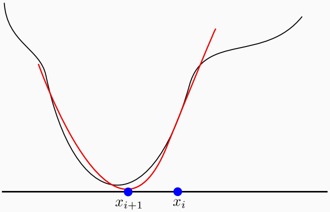
Conjugate gradient avoids moves along the same direction.

For a *D*-dim quadratic loss reaches minimum in *D* steps

Common default method

Newton's method

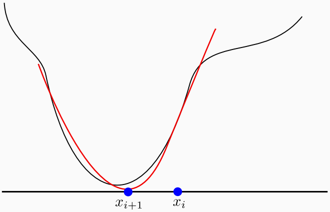
Uses the second derivative (curvature) to decide the step-size η.



At current point x_i, evaluate $f(x_i), f'(x_i)$ and $f''(x_i)$.
Fit a parabola having these values and set x_{i+1} to its minimum.

Newton's method

Uses the second derivative (curvature) to decide the step-size η.



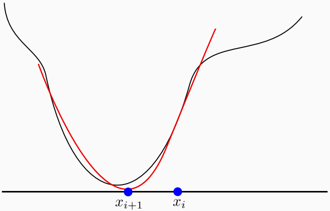
At current point x_i, evaluate $f(x_i), f'(x_i)$ and $f''(x_i)$.
Fit a parabola having these values and set x_{i+1} to its minimum.

Easy to see that (show it!):

$$x_{i+1} = x_i - f'(x_i)/f''(x_i)$$

Newton's method

Uses the second derivative (curvature) to decide the step-size η.



At current point x_i, evaluate $f(x_i), f'(x_i)$ and $f''(x_i)$.
Fit a parabola having these values and set x_{i+1} to its minimum.

Easy to see that (show it!):

$$x_{i+1} = x_i - f'(x_i)/f''(x_i)$$

If f'' is large, we're uncertain about f', so take a small step.

Newton's method in higher dimensions

Update rule:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - [\nabla^2 f(\mathbf{x}_i)]^{-1} \nabla f(\mathbf{x}_i)$$

Newton's method in higher dimensions

Update rule:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - [\nabla^2 f(\mathbf{x}_i)]^{-1} \nabla f(\mathbf{x}_i)$$

Need to calculate the Hessian $\nabla^2 f$: N^2 elements.

Need to invert the Hessian: N^3 operations.

Newton's method in higher dimensions

Update rule:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - [\nabla^2 f(\mathbf{x}_i)]^{-1} \nabla f(\mathbf{x}_i)$$

Need to calculate the Hessian $\nabla^2 f$: N^2 elements.

Need to invert the Hessian: N^3 operations.

Each iteration can be expensive.
Quasi-Newton methods try to alleviate this issue.

Update rule:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - [\nabla^2 f(\mathbf{x}_i)]^{-1} \nabla f(\mathbf{x}_i)$$

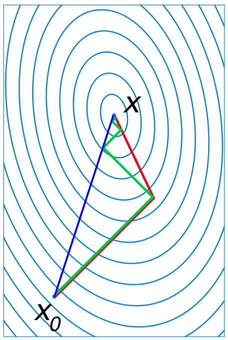
Need to calculate the Hessian $\nabla^2 f$: N^2 elements.

Need to invert the Hessian: N^3 operations.

Each iteration can be expensive.
Quasi-Newton methods try to alleviate this issue.

We have to be wary about taking wild steps.

Newton's descent



Set **p** to the minimum of the local quadratic approximation.

$$\mathbf{p} = [\nabla^2 f(\mathbf{x})]^{-1} \nabla f(x_i)$$

Reaches minimum of quadratic loss in 1 step

Quasi-Newton methods

Newton's method: $\mathbf{p} = [\nabla^2 f(\mathbf{x})]^{-1} \nabla f(\mathbf{x}_i)$

Steepest's descept: $\mathbf{p} = I \, \nabla f(\mathbf{x}_i)$

Quasi-Newton methods

Newton's method: $\mathbf{p} = [\nabla^2 f(\mathbf{x})]^{-1} \nabla f(\mathbf{x}_i)$

Steepest's descept: $\mathbf{p} = I \, \nabla f(\mathbf{x}_i)$

Quasi-Newton methods use other matrices B:

$$\mathbf{p} = \mathrm{B} \nabla f(x_i)$$

Quasi-Newton methods

Newton's method: $\mathbf{p} = [\nabla^2 f(\mathbf{x})]^{-1} \nabla f(\mathbf{x}_i)$

Steepest's descept: $\mathbf{p} = I \, \nabla f(\mathbf{x}_i)$

Quasi-Newton methods use other matrices B:

$$\mathbf{p} = \mathbf{B} \nabla f(x_i)$$

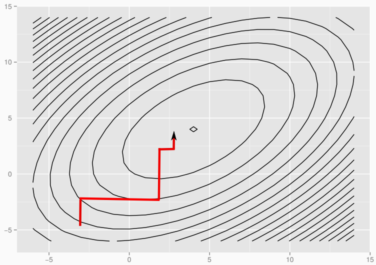
Usually, $\mathbf{B}$ is allowed to vary from iteration to iteration, with

$$\mathbf{B}_i \to [\nabla^2 f(\mathbf{x})]^{-1}$$

Get benefits of Newton's method, without $O(N^3)$ computations.

E.g. BFGS

Co-ordinate descent



Saw this last lecture

Simple, clean and inexpensive.
Often the 1-d problems can be solved exactly.

Convergence can be slow.
Exception: axis aligned ellipses need just D steps.

Use the `optim` function

Syntax:

```
optim(par, fn, gr = NULL, ...,
  method = c('Nelder-Mead', 'BFGS', 'CG', 'L-BFGS-B', 'SANN',
             'Brent'),
  lower = -Inf, upper = Inf,
  control = list(), hessian = FALSE)
```

`fn`: function to be optimized

`gr`: gradient function (calculate numerically if `NULL`)

`par`: initial value of parameter to be optimized (should be first argument of `fn`)