LECTURE 12: ROOT-FINDING AND MINIMIZATION

STAT 545: Intro. to Computational Statistics

Vinayak Rao

Purdue University

October 22, 2018

Root-finding in one-dimension

Given some nonlinear function $f : \mathbb{R} \rightarrow \mathbb{R}$, solve

$$f(x) = 0$$

Invariably need iterative methods.

Assume f is continuous (else things are really messy).

More we know about f (e.g. gradients), better we can do.

Better: faster (asymptotic) convergence.

Root bracketing

$f(a)$ and $f(b)$ have opposite signs $\rightarrow$ root lies in (a, b).

a and b *bracket* the root.

Finding an initial bracketing can be non-trivial.
Typically, start with an initial interval and expand or contract.

Below, we assume we have an initial bracketing.

Root bracketing

$f(a)$ and $f(b)$ have opposite signs → root lies in (a, b).

a and b *bracket* the root.

Finding an initial bracketing can be non-trivial.
Typically, start with an initial interval and expand or contract.

Below, we assume we have an initial bracketing.

Not always possible e.g. $f(x) = (x - a)^2$ (in general, multiple roots/nearby roots lead to trouble).

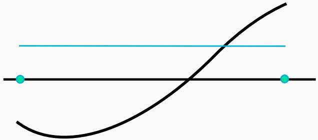
Bisection method

Simplest root-finding algorithm.

Given an initial bracketing, cannot fail.

But is slower than other methods.

Successively halves the bracketing interval (binary search):

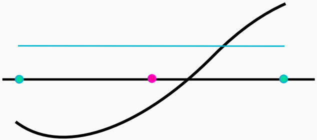


- `Current interval =` (a, b)
- `Set` $c = \frac{a+b}{2}$
- `New interval =` (a, c) `or` (c, b) `(whichever is a valid bracketing)`

Bisection method

Simplest root-finding algorithm.
Given an initial bracketing, cannot fail.
But is slower than other methods.

Successively halves the bracketing interval (binary search):

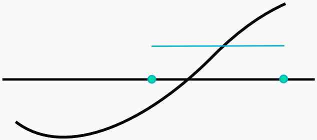


- `Current interval =` (a, b)
- `Set` $c = \frac{a+b}{2}$
- `New interval =` (a, c) `or` (c, b)
  `(whichever is a valid bracketing)`

Bisection method

Simplest root-finding algorithm.
Given an initial bracketing, cannot fail.
But is slower than other methods.

Successively halves the bracketing interval (binary search):



- `Current interval =` (a, b)
- `Set` $c = \frac{a+b}{2}$
- `New interval =` (a, c) `or` (c, b)
  `(whichever is a valid bracketing)`

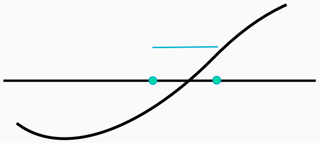
Bisection method

Simplest root-finding algorithm.

Given an initial bracketing, cannot fail.

But is slower than other methods.

Successively halves the bracketing interval (binary search):



- `Current interval =` (a, b)
- `Set` $c = \frac{a+b}{2}$
- `New interval =` (a, c) `or` (c, b)
  `(whichever is a valid bracketing)`

Bisection method (contd)

Let ϵ_n be the interval length at iteration n.

Upperbounds error in root.

$$\epsilon_{n+1} = 0.5\ \epsilon_n \qquad \text{(Linear convergence)}$$

Bisection method (contd)

Let ϵ_n be the interval length at iteration n.
Upperbounds error in root.

$$\epsilon_{n+1} = 0.5\ \epsilon_n \qquad \text{(Linear convergence)}$$

Linear convergence:

- each iteration reduces error by one significant figure.
- every (fixed) k iterations reduces error by one digit.
- error reduced exponentially with the number of iterations.

Bisection method (contd)

Let ϵ_n be the interval length at iteration n.
Upperbounds error in root.

$$\epsilon_{n+1} = 0.5\ \epsilon_n \qquad \text{(Linear convergence)}$$

Linear convergence:

- each iteration reduces error by one significant figure.
- every (fixed) k iterations reduces error by one digit.
- error reduced exponentially with the number of iterations.

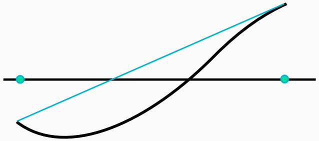
Superlinear convergence:

$$\lim_{n\to\infty} |\epsilon_{n+1}| = C \times |\epsilon_n|^m \qquad (m > 1)$$

Quadratic convergence:
Number of significant figures *doubles* every iteration.

Secant method and bisection method

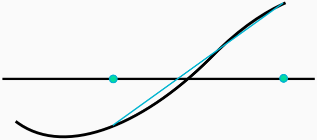
Linearly approximate f to find new approximation to root.



Secant method and bisection method

Linearly approximate *f* to find new approximation to root.

Secant method:



- always keep the newest point
- Superlinear convergence ($m = 1.618$, the golden ratio)

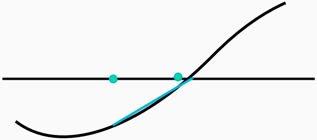
$$\lim_{n \to \infty} |\epsilon_{n+1}| = C \times |\epsilon_n|^{1.618}$$

- Bracketing (and thus convergence) not guaranteed.

Secant method and bisection method

Linearly approximate f to find new approximation to root.

Secant method:



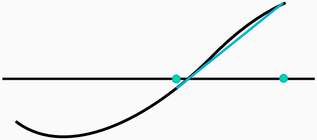
- always keep the newest point
- Superlinear convergence ($m = 1.618$, the golden ratio)

$$\lim_{n\to\infty} |\epsilon_{n+1}| = C \times |\epsilon_n|^{1.618}$$

- Bracketing (and thus convergence) not guaranteed.

Secant method and bisection method

Linearly approximate *f* to find new approximation to root.



Secant method:

- always keep the newest point
- Superlinear convergence ($m = 1.618$, the golden ratio)

$$\lim_{n\to\infty} |\epsilon_{n+1}| = C \times |\epsilon_n|^{1.618}$$

- Bracketing (and thus convergence) not guaranteed.

False position:

- Can choose an old point that guarantees bracketing.
- Convergence analysis is harder.

Practical root-finding

In practice, people use more sophiticated algorithms.

Most popular is Brent's method.

Maintains bracketing by combining bisection method with a quadratic approximation.

Lots of book-keeping.

Newton's method (a.k.a. Newton-Raphson)

At any point uses both function evaluation as well as derivative to form a linear approximation.

Newton's method (a.k.a. Newton-Raphson)

At any point uses both function evaluation as well as derivative to form a linear approximation.

Taylor expansion: $f(x+\delta) = f(x) + \delta f'(x) + \frac{\delta^2}{2} f''(x) + \cdots$

Newton's method (a.k.a. Newton-Raphson)

At any point uses both function evaluation as well as derivative to form a linear approximation.

Taylor expansion: $f(x+\delta) = f(x) + \delta f'(x) + \frac{\delta^2}{2} f''(x) + \cdots$

Assume second- and higher-order terms are negligible.
Given x_i, choose $x_{i+1} = x_i + \delta$ so that $f(x_{i+1}) = 0$:

Newton's method (a.k.a. Newton-Raphson)

At any point uses both function evaluation as well as derivative to form a linear approximation.

Taylor expansion: $f(x+\delta) = f(x) + \delta f'(x) + \frac{\delta^2}{2} f''(x) + \cdots$

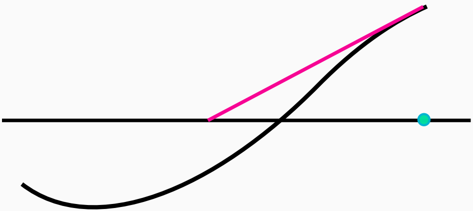
Assume second- and higher-order terms are negligible. Given x_i, choose $x_{i+1} = x_i + \delta$ so that $f(x_{i+1}) = 0$:

$$0 = f(x_i) + \delta f'(x_i)$$

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

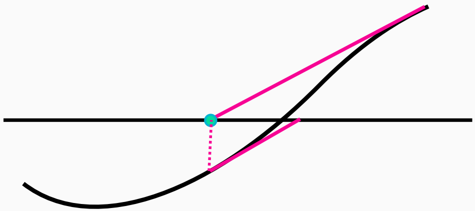
Newton's method (a.k.a. Newton-Raphson)

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



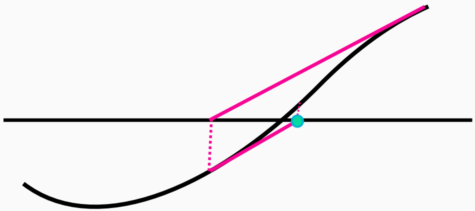
Newton's method (a.k.a. Newton-Raphson)

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



Newton's method (a.k.a. Newton-Raphson)

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



Convergence of Newton's method

Letting x^* be the root, we have

$$x_{i+1} - x^* = x_i - x^* - f(x_i)/f'(x_i)$$
$$\epsilon_{i+1} = \epsilon_i - f(x_i)/f'(x_i)$$

Convergence of Newton's method

Letting x^* be the root, we have

$$x_{i+1} - x^* = x_i - x^* - f(x_i)/f'(x_i)$$
$$\epsilon_{i+1} = \epsilon_i - f(x_i)/f'(x_i)$$

Also since $x_i = x^* + \epsilon_i$,

$$f(x_i) \approx f(x^*) + \epsilon_i f'(x^*) + \frac{\epsilon_i^2}{2} f''(x^*)$$

Convergence of Newton's method

Letting x^* be the root, we have

$$x_{i+1} - x^* = x_i - x^* - f(x_i)/f'(x_i)$$
$$\epsilon_{i+1} = \epsilon_i - f(x_i)/f'(x_i)$$

Also since $x_i = x^* + \epsilon_i$,

$$f(x_i) \approx f(x^*) + \epsilon_i f'(x^*) + \frac{\epsilon_i^2}{2} f''(x^*)$$

This gives

$$\epsilon_{i+1} = -\frac{f'(x_i)}{2f''(x_i)}\epsilon_i^2$$

Convergence of Newton's method

Letting x^* be the root, we have

$$x_{i+1} - x^* = x_i - x^* - f(x_i)/f'(x_i)$$
$$\epsilon_{i+1} = \epsilon_i - f(x_i)/f'(x_i)$$

Also since $x_i = x^* + \epsilon_i$,

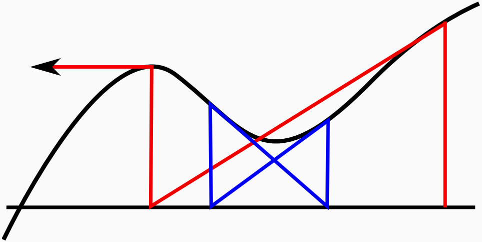
$$f(x_i) \approx f(x^*) + \epsilon_i f'(x^*) + \frac{\epsilon_i^2}{2} f''(x^*)$$

This gives

$$\epsilon_{i+1} = -\frac{f'(x_i)}{2f''(x_i)} \epsilon_i^2$$

Quadratic convergence (assuming $f'(x)$ is non-zero at the root)

Pitfalls of Newton's method



Away from the root the linear approximation can be bad.

Can give crazy results (go off to infinity, cycles etc.)

However, once we have a decent solution can be used to rapidly ‘polish the root’.

Often used in combination with some bracketing method.

Root-finding for systems of nonlinear equations

Now have N functions $F_1, F_2, \cdots, F_N$ of N variables $x_1, x_2, \cdots, x_N$

Find $(x_1, \cdots, x_N)$ such that:

$$F_i(x_1, \cdots, x_N) = 0 \qquad i = 1 \; to \; N$$

Much harder than the 1-d case.

Much harder than optimization.

Newton's method

Again, consider a Taylor expansion:

$$\mathsf{F}(\mathsf{x} + \delta\mathsf{x}) = \mathsf{F}(\mathsf{x}) + \mathsf{J}(\mathsf{x}) \cdot \delta\mathsf{x} + O(\delta\mathsf{x}^2)$$

Here, $\mathsf{J}(\mathsf{x})$ is the Jacobian matrix at x, with $J_{ij} = \frac{\partial F_i}{\partial x_j}$.

Newton's method

Again, consider a Taylor expansion:

$$\mathsf{F}(\mathsf{x} + \delta\mathsf{x}) = \mathsf{F}(\mathsf{x}) + \mathsf{J}(\mathsf{x}) \cdot \delta\mathsf{x} + O(\delta\mathsf{x}^2)$$

Here, $\mathsf{J}(\mathsf{x})$ is the Jacobian matrix at x, with $J_{ij} = \frac{\partial F_i}{\partial x_j}$.

Again, Newton's method finds $\delta\mathsf{x}$ by solving $\mathsf{F}(\mathsf{x} + \delta\mathsf{x}) = 0$

$$\mathsf{J}(\mathsf{x}) \cdot \delta\mathsf{x} = -\mathsf{F}(\mathsf{x})$$

Solve $\delta\mathsf{x} = -\mathsf{J}(\mathsf{x})^{-1} \cdot \mathsf{F}(\mathsf{x})$ (e.g. by LU decomposition)

Newton's method

Again, consider a Taylor expansion:

$$\mathsf{F}(\mathsf{x} + \delta\mathsf{x}) = \mathsf{F}(\mathsf{x}) + \mathsf{J}(\mathsf{x}) \cdot \delta\mathsf{x} + O(\delta\mathsf{x}^2)$$

Here, $\mathsf{J}(\mathsf{x})$ is the Jacobian matrix at x, with $J_{ij} = \frac{\partial F_i}{\partial x_j}$.

Again, Newton's method finds $\delta\mathsf{x}$ by solving $\mathsf{F}(\mathsf{x} + \delta\mathsf{x}) = 0$

$$\mathsf{J}(\mathsf{x}) \cdot \delta\mathsf{x} = -\mathsf{F}(\mathsf{x})$$

Solve $\delta\mathsf{x} = -\mathsf{J}(\mathsf{x})^{-1} \cdot \mathsf{F}(\mathsf{x})$ (e.g. by LU decomposition)

Iterate $\mathsf{x}_{new} = \mathsf{x}_{old} + \delta\mathsf{x}$ until convergence.

Can wildly careen through space if not careful.

Global methods via optimization

Recall, we want to solve $\mathsf{F}(\mathsf{x}) = 0 \quad (F_i(\mathsf{x}) = 0, \quad i = 1 \cdots N)$.

Global methods via optimization

Recall, we want to solve $\mathsf{F}(\mathsf{x}) = 0 \quad (F_i(\mathsf{x}) = 0, \quad i = 1 \cdots N)$.

Minimize $f(\mathsf{x}) = \frac{1}{2}\sum_{i=1}^{N} |F_i(\mathsf{x})|^2 = \frac{1}{2}|\mathsf{F}(\mathsf{x})|^2 = \frac{1}{2}\mathsf{F}(\mathsf{x}) \cdot \mathsf{F}(\mathsf{x})$.

Global methods via optimization

Recall, we want to solve $\mathsf{F}(\mathsf{x}) = 0 \quad (F_i(\mathsf{x}) = 0, \quad i = 1 \cdots N)$.

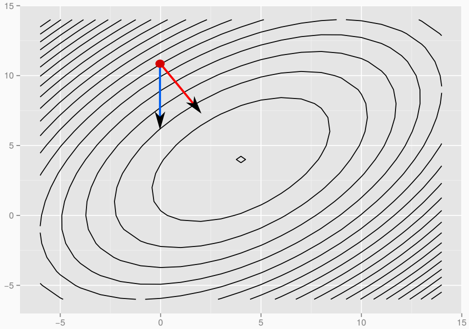
Minimize $f(\mathsf{x}) = \frac{1}{2}\sum_{i=1}^{N} |F_i(\mathsf{x})|^2 = \frac{1}{2}|\mathsf{F}(\mathsf{x})|^2 = \frac{1}{2}\mathsf{F}(\mathsf{x}) \cdot \mathsf{F}(\mathsf{x})$.

Note: It is NOT sufficient to find a local minimum of f.

Global methods via optimization)

We move along $\delta\mathsf{x}$ instead of $\nabla f = \mathsf{F}(\mathsf{x})\mathsf{J}(\mathsf{x})$.

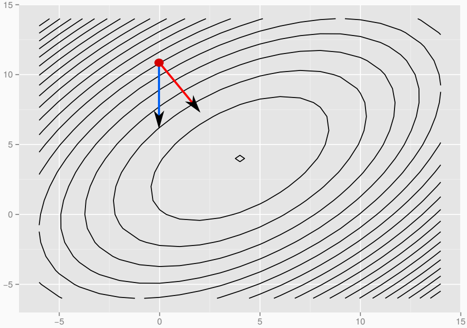
This keeps our global objective in sight.



Global methods via optimization)

We move along $\delta \mathsf{x}$ instead of $\nabla f = \mathsf{F}(\mathsf{x})\mathsf{J}(\mathsf{x})$.

This keeps our global objective in sight.



Note: $\nabla f \cdot \delta \mathsf{x} = (\mathsf{F}(\mathsf{x})\mathsf{J}(\mathsf{x})) \cdot (-\mathsf{J}^{-1}(\mathsf{x})\mathsf{F}(\mathsf{x})) = -\mathsf{F}(\mathsf{x})\mathsf{F}(\mathsf{x}) < 0$

Newton's method with backtracking

A full Newton step sets $\mathsf{x}_{new} = \mathsf{x}_{old} + \delta\mathsf{x}$.

This can cause f to increase i.e. $f(\mathsf{x}_{new} > f(\mathsf{x}_{old})$.

In this case, *backtrack* and set $\mathsf{x}_{new} = \mathsf{x}_{old} + \lambda\delta\mathsf{x}, \quad \lambda \in (0, 1)$.

Since $\delta\mathsf{x}$ is a descent direction, there exists a sufficiently small λ that causes f to decrease.

Newton's method with backtracking

A full Newton step sets $\mathsf{x}_{new} = \mathsf{x}_{old} + \delta\mathsf{x}$.

This can cause f to increase i.e. $f(\mathsf{x}_{new} > f(\mathsf{x}_{old}))$.

In this case, *backtrack* and set $\mathsf{x}_{new} = \mathsf{x}_{old} + \lambda\delta\mathsf{x}, \quad \lambda \in (0, 1)$.

Since $\delta\mathsf{x}$ is a descent direction, there exists a sufficiently small λ that causes f to decrease.

Finding best λ: too much work usually.

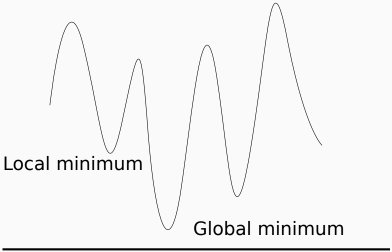
However, just causing f to decrease is not sufficient.

Can use Wolfe conditions (later)

Global and local minimum

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

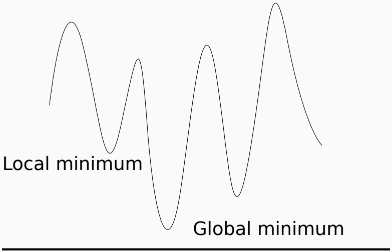
No global information (e.g. only function evaluations, derivatives).



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 evaluations, derivatives).



Finding a global minimum is hard! Usually settle for finding a local minimum (like the EM algorithm).

Conceptually (deceptively?) simpler than EM.

Gradient descent (iterative method)

Let x_{old} be our current value.

Update x_{new} as $$x_{new} = x_{old} - \eta \, \frac{\mathrm{d}f}{\mathrm{d}x}\bigg|_{x_{old}}$$

The steeper the slope, the bigger the move.

Gradient descent (iterative method)

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’
(from neural network literature)

Gradient descent (iterative method)

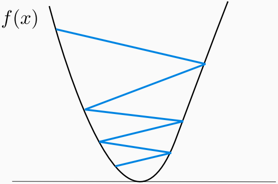
Let x_{old} be our current value.

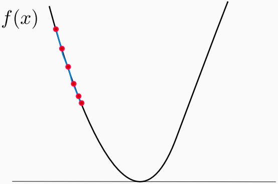
Update x_{new} as $$x_{new} = x_{old} - \eta \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_{old}}$$

The steeper the slope, the bigger the move.

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

Choosing η is a dark art:





Gradient descent (iterative method)

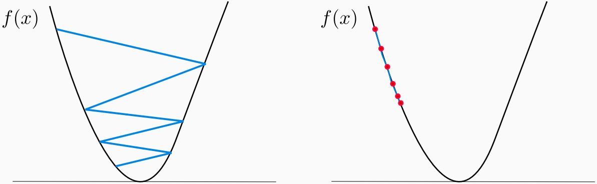
Let x_{old} be our current value.

Update x_{new} as $$x_{new} = x_{old} - \eta\, \frac{\mathrm{d}f}{\mathrm{d}x}\bigg|_{x_{old}}$$

The steeper the slope, the bigger the move.

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

Choosing η is a dark art:



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

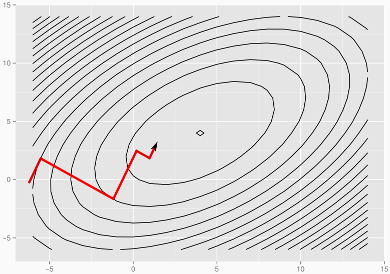
Gradient descent in higher-dimensions

Steepest descent also applies to higher dimensions too:

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

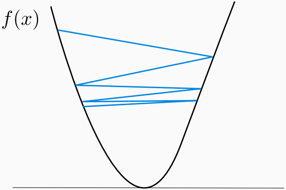
At each step, solve a 1-d problem along the gradient

Now, even the optimal step-size η can be inefficient:

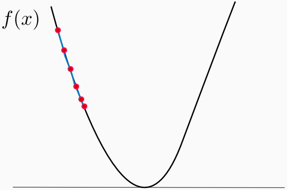


Rather than the best step-size each step, find a decent solution

Wolfe conditions

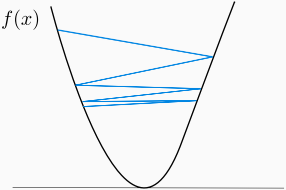


Big steps with little decrease

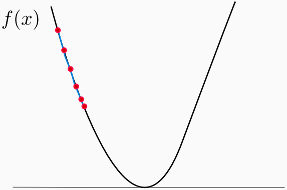


Small steps getting us nowhere

Wolfe conditions



Big steps with little decrease

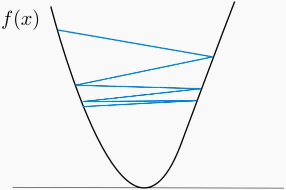


Small steps getting us nowhere

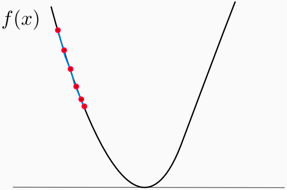
Avg. decrease at least some fraction of initial rate:

$$f(\mathbf{x} + \lambda \delta \mathbf{x}) \leq f(\mathbf{x}) + c_1 \lambda (\nabla f \cdot \delta \mathbf{x}), \qquad c_1 \in (0,1) \; e.g. \; 0.9$$

Wolfe conditions



Big steps with little decrease



Small steps getting us nowhere

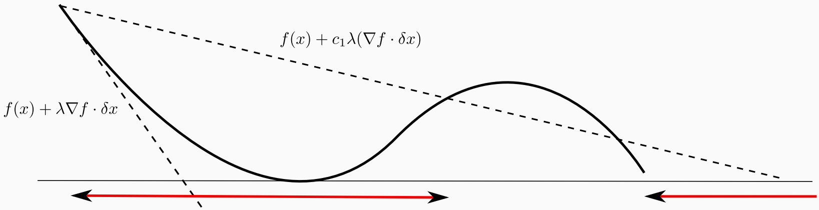
Avg. decrease at least some fraction of initial rate:

$$f(\mathsf{x} + \lambda\delta\mathsf{x}) \leq f(\mathsf{x}) + c_1\lambda(\nabla f \cdot \delta\mathsf{x}), \qquad c_1 \in (0, 1)\ e.g.\ 0.9$$

Final rate is greater than some fraction of initial rate:

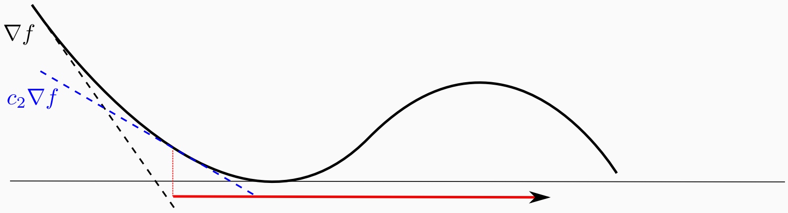
$$\nabla f(\mathsf{x} + \lambda\delta\mathsf{x}) \cdot \delta\mathsf{x} \geq c_2\nabla f(\mathsf{x})\delta\mathsf{x}, \qquad c_2 \in (0, 1)\ e.g.\ 0.1$$

Wolfe conditions


$f(x) + c_1\lambda(\nabla f \cdot \delta x)$
$f(x) + \lambda\nabla f \cdot \delta x$

Permissible λ's under condition 1

Wolfe conditions



Permissible λ's under condition 2

Wolfe conditions

A simple way to satisfy Wolfe conditions:

Set $\delta x = -\nabla f, c_1 = c_2 = .5$

Start with $\lambda = 1$, and while condition i is not satisfied, set $\lambda = \beta_i t$ (for $\beta_1 \in (0, 1), \beta_2 > 1$ and $\beta_1 * \beta_2 < 1$

Estimating MLE

Consider a set of observations $Y = (y_1, \cdots, y_N)$.

Assume $y_i \sim p(y|\theta)$

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

Estimating MLE

Consider a set of observations $Y = (y_1, \cdots, y_N)$.

Assume $y_i \sim p(y|\theta)$

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

The gradient of the log-likelihood is $\nabla \ell(\theta) = \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$
The average of the gradients of each datapoint.

Estimating MLE

Consider a set of observations $Y = (y_1, \cdots, y_N)$.

Assume $y_i \sim p(y|\theta)$

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

The gradient of the log-likelihood is $\nabla \ell(\theta) = \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$
The average of the gradients of each datapoint.

Starting with an initial θ_0, iterate:

$$\theta_{i+1} = \theta_i + \eta_i \nabla \ell(\theta_i)$$

Gradient descent (contd.)

$$\nabla \ell(\theta) = \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$$

Cons:

- Calculating the gradient is $O(N)$.
  (Each iteration must cycle through all datapoints.)
- *Lots* of redundancy, esp. for large N.

Gradient descent (contd.)

$$\nabla \ell(\theta) = \sum_{i=1}^{N} \nabla \log p(x_i|\theta)$$

Cons:

- Calculating the gradient is $O(N)$.
  (Each iteration must cycle through all datapoints.)
- *Lots* of redundancy, esp. for large N.

Pros:

- Convergence is better understood.
- Accelerated methods are available (e.g. Newton's method, conjugate gradient)

Stochastic gradient descent

Use a noisy gradient $\widehat{\nabla\ell}$.

Typically split data into N/B batches of size B.

Each iteration, calculate gradient on one of the batches B_i:

$$\widehat{\nabla\ell}(\theta) = \sum_{j \in B_i} \nabla \log p(x_j|\theta)$$

Stochastic gradient descent

Use a noisy gradient $\widehat{\nabla \ell}$.

Typically split data into N/B batches of size B.
Each iteration, calculate gradient on one of the batches B_i:

$$\widehat{\nabla \ell}(\theta) = \sum_{j \in B_i} \nabla \log p(x_j|\theta)$$

Pros:

- Calculating the gradient is $O(B)$.
  (Often, each batch is just a single datapoint)
- Much faster convergence (just one sweep through the data can get you a decent solution).
- Often, you get better solutions.
- Useful for online systems, tracking θ that varies over time .

Stochastic gradient descent

Use a noisy gradient $\widehat{\nabla \ell}$.

Typically split data into N/B batches of size B.
Each iteration, calculate gradient on one of the batches B_i:

$$\widehat{\nabla \ell}(\theta) = \sum_{j \in B_i} \nabla \log p(x_j | \theta)$$

Cons:

- Convergence analysis is harder.
- Noisy gradients mean the algorithm will never converge.
  Typically need to reduce the step size every iteration.
  We want

$$\eta_i \to 0, \quad \sum_{i=1}^{\infty} \eta_i = \infty$$

E.g. $\eta_i = \frac{a}{b+i}$

Stochastic gradient descent with momentum

One way to accelarate convergence is to include a momentum term:

$$\theta_{i+1} = \theta_i + \eta_i \widehat{\nabla \ell}(\theta_i) + \beta_i \underbrace{(\theta_i - \theta_{i-1})}_{\text{momentum}}$$

Stochastic gradient descent with momentum

One way to accelarate convergence is to include a momentum term:

$$\theta_{i+1} = \theta_i + \eta_i \widehat{\nabla \ell}(\theta_i) + \beta_i \underbrace{(\theta_i - \theta_{i-1})}_{\text{momentum}}$$

More generally,

$$\theta_{i+1} = \theta_i + \eta_i \widehat{\nabla \ell}(\theta_i + \gamma(\theta_i - \theta_{i-1})) + \beta_i(\theta_i - \theta_{i-1})$$

Include many popular algorithms:

- Polyak's heavy ball method (HB): $\gamma = 0$
- Nesterov's accelerated gradient (NAG): $\gamma_i = \beta_i$

Adaptive methods

Adaptive methods accelerate convergence by using the entire history of iterates to determine step-sizes.

Adaptive methods

Adaptive methods accelerate convergence by using the entire history of iterates to determine step-sizes.

Often take the general form

$$\theta_{i+1} = \theta_i + \eta_i H_i^{-1} \widehat{\nabla} \ell(\theta_i + \gamma(\theta_i - \theta_{i-1})) + \beta_i H_i^{-1} H_{i-1} (\theta_i - \theta_{i-1})$$

where H_i is some combination of all previous gradients. E.g.

$$H_i = \operatorname{diag}\left(\sum_{j=1}^{i} g_j \circ g_j \right),$$

with $g_j = \widehat{\nabla} \ell(\theta_j + \gamma(\theta_j - \theta_{j-1}))$, and $\circ$ element-wise product.

Examples are AdaGrad, Adam etc.