LECTURE 19: ROOT-FINDING AND MINIMIZATION

STAT 545: Intro. to Computational Statistics

Vinayak Rao

Purdue University

November 13, 2019

Given some nonlinear function $f : \mathbb{R} \to \mathbb{R}$, 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 $\to$ 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 $\to$ 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).

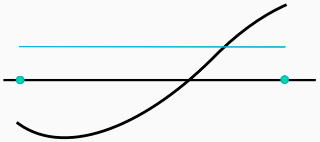
Bisection method

Simplest root-finding algorithm.

Given an initial bracketing, cannot fail.

But is slower than other methods.

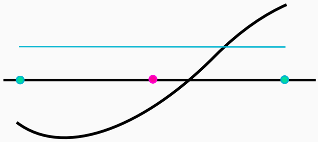
Successively halves the bracketing interval (binary search):



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

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

Successively halves the bracketing interval (binary search):

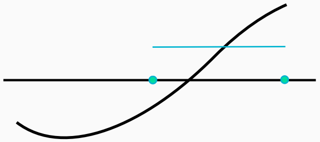


- `Current interval =` (a, b)
- `Set` $c = \frac{a+b}{2}$
- `New interval =` (a, c) `or` (c, b)
  `(whichever is a valid bracketing)`

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)`

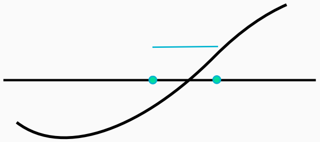
Bisection method

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:

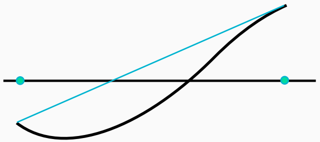
- each iteration reduces error by one significant figure.
- every (fixed) k iterations reduces error by one digit.
- error reduced exponentially with the number of iterations.

Superlinear convergence:

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

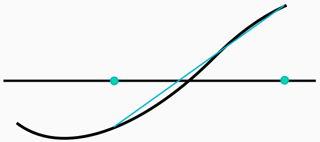
Quadratic convergence:
Number of significant figures *doubles* every iteration.

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: 

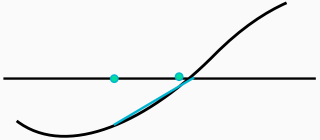
- always keep the newest point
- Superlinear convergence ($m = 1.618$, the golden ratio)

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

- Bracketing (and thus convergence) not guaranteed.

Linearly approximate f to find new approximation to root.

Secant method:



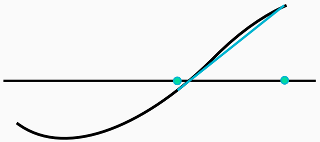
- always keep the newest point
- Superlinear convergence ($m = 1.618$, the golden ratio)

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

- Bracketing (and thus convergence) not guaranteed.

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

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

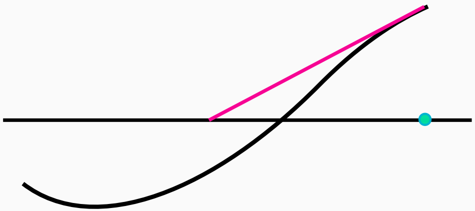
Assume second- and higher-order terms are negligible.
Given x_i, choose $x_{i+1} = x_i + \delta$ so that $f(x_{i+1}) = 0$:

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

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

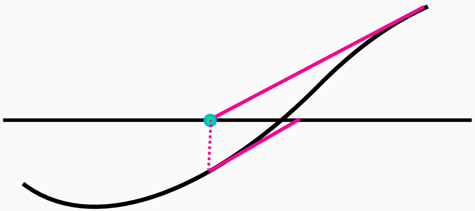
Newton's method (a.k.a. Newton-Raphson)

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



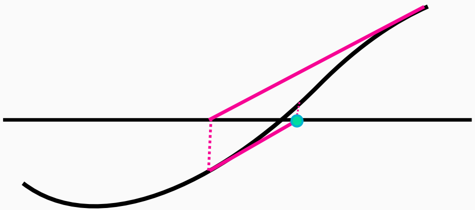
Newton's method (a.k.a. Newton-Raphson)

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



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

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



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 $x_i = x^* + \epsilon_i$,

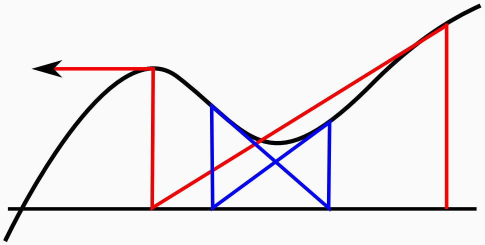
$$f(x_i) \approx f(x^*) + \epsilon_i f'(x^*) + \frac{\epsilon_i^2}{2} f''(x^*)$$

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.

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

Global methods via optimization

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

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

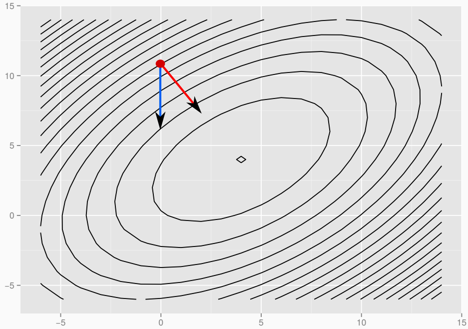
Recall, we want to solve $\mathsf{F}(\mathsf{x}) = 0 \quad (F_i(\mathsf{x}) = 0, \quad i = 1 \cdots N)$.

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.

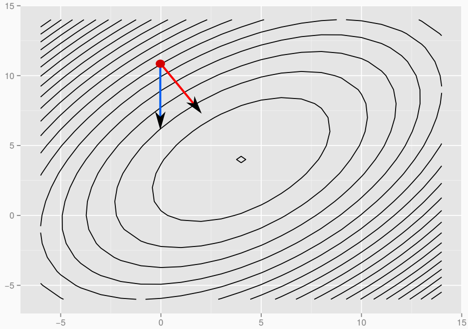
We move along δx instead of $\nabla f = \mathsf{F(x)J(x)}$.

This keeps our global objective in sight.



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.

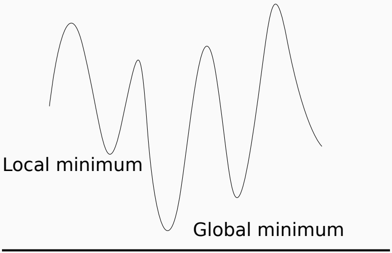


Note: $\nabla f \cdot \delta \mathsf{x} = (\mathsf{F}(\mathsf{x})\mathsf{J}(\mathsf{x})) \cdot (-\mathsf{J}^{-1}(\mathsf{x})\mathsf{F}(\mathsf{x})) = -\mathsf{F}(\mathsf{x})\mathsf{F}(\mathsf{x}) < 0$

Global and local minimum

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

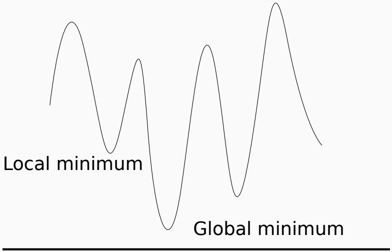
No global information (e.g. only function evaluations, derivatives).



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

The steeper the slope, the bigger the move.

Gradient descent (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)

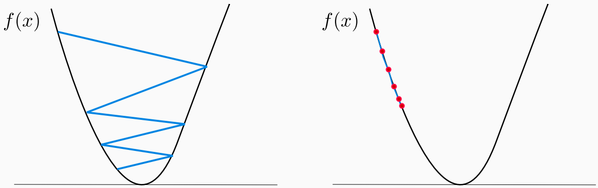
Let x_{old} be our current value.

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)

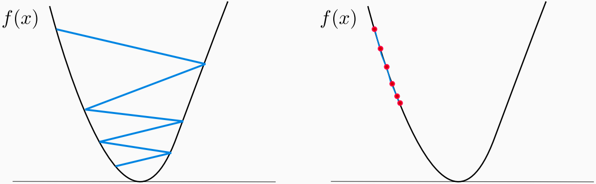
Let x_{old} be our current value.

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

The steeper the slope, the bigger the move.

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

Choosing η is a dark art:



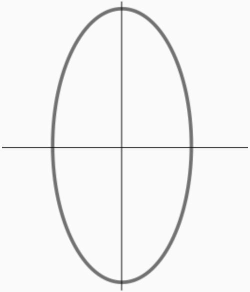
Better methods adapt step-size according to the curvature of f.

Steepest descent also applies to higher dimensions too:

$$x_{new} = x_{old} - \eta \, \nabla f|_{x_{old}}$$

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

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

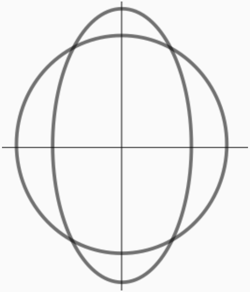


Steepest descent also applies to higher dimensions too:

$$x_{new} = x_{old} - \eta \, \nabla f\big|_{x_{old}}$$

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

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



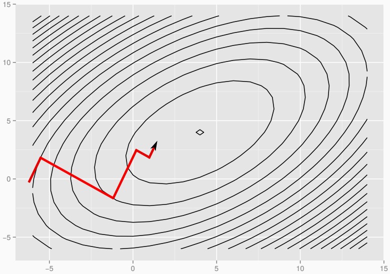
Gradient descent in higher-dimensions

Steepest descent also applies to higher dimensions too:

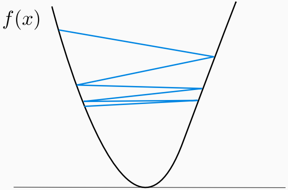
$$x_{new} = x_{old} - \eta \nabla f|_{x_{old}}$$

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

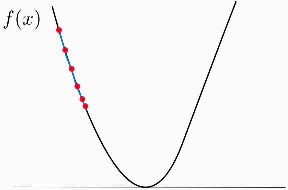
Now, even the optimal step-size η can be inefficient:



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



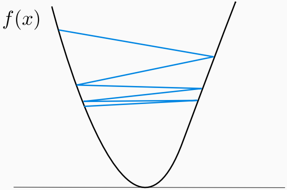
Big steps with little decrease



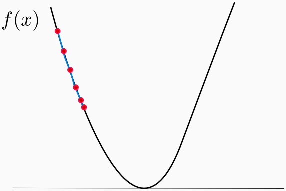
Small steps getting us nowhere

Wolfe conditions

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



Big steps with little decrease



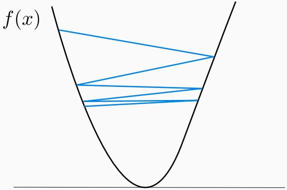
Small steps getting us nowhere

Avg. decrease at least some fraction of initial rate:

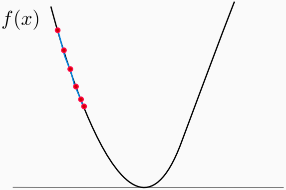
$$f(\mathbf{x} + \lambda \delta \mathbf{x}) \leq f(\mathbf{x}) + c_1 \lambda (\nabla f \cdot \delta \mathbf{x}), \qquad c_1 \in (0, 1) \; e.g. \; 0.9$$

Wolfe conditions

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



Big steps with little decrease



Small steps getting us nowhere

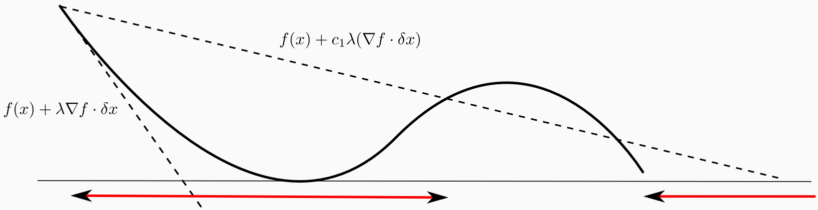
Avg. decrease at least some fraction of initial rate:

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

Final rate is greater than some fraction of initial rate:

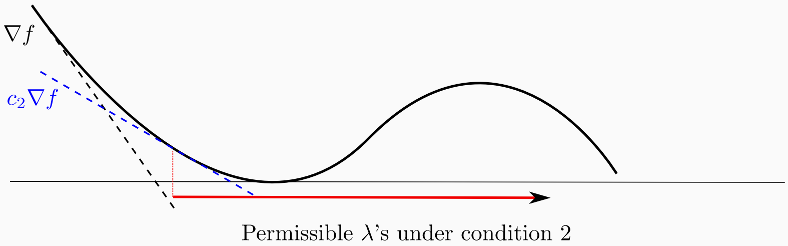
$$\nabla f(\mathbf{x} + \lambda\delta\mathbf{x}) \cdot \delta\mathbf{x} \geq c_2 \nabla f(\mathbf{x})\delta\mathbf{x}, \qquad c_2 \in (0,1) \; e.g. \; 0.1$$

Wolfe conditions



Permissible λ's under condition 1

Wolfe conditions



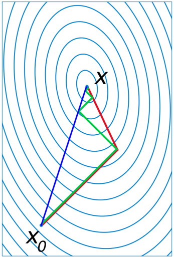
Wolfe conditions

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 $\beta_1 * \beta_2 < 1$

Consider minimizing $\frac{1}{2}x^T Ax - b^T x$:



Steepest descent can take many steps to get to the minimum

Problem: After minimizing along a direction, gradient is perpendicular to previous direction (why)

- Can 'cancel' out earlier gains

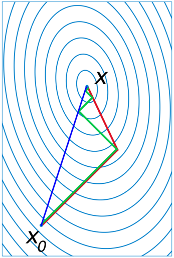
A popular algorithm is conjugate gradient descent

Sequentially updates along directions $p_1, \cdots p_N$:

$$x_{t+1} = x_t + \lambda_{t+1} p_t, \text{ where } \lambda_{t+1} = \text{argmin}_{\lambda} \; f(x_t + \lambda p_t)$$

$$p_{t+1} = \nabla f(x_{t+1}) + \frac{\langle \nabla f(x_{t+1}, \nabla f(x_{t+1}) \rangle}{\langle \nabla f(x_t, \nabla f(x_t) \rangle} p_t$$

Consider minimizing $\frac{1}{2}x^TAx - b^Tx$:



Steepest descent can take many steps to get to the minimum

Problem: After minimizing along a direction, gradient is perpendicular to previous direction (why)

- Can 'cancel' out earlier gains

If $f(x) = \frac{1}{2}x^TAx - b^Tx, \quad x \in \mathbb{R}^d$, CG takes max d steps to converge

Can show the directions satisfy $\langle p_{t+1}, p_t \rangle_A := p_{t+1}^T A p_t = 0$

(this is unlike $p_{t+1}^T p_t = 0$ for steepest descent)