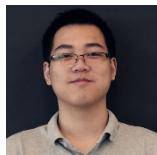




Numerical Optimization in Robotics

Lecture 2: Unconstrained Optimization



主讲人 Zhepei Wang

Ph.D. in Robotics
Zhejiang University





Recommended Reading List

1. Liu DC, Nocedal J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*. 1989 Aug;45(1):503-28.
2. Li DH, Fukushima M. On the global convergence of the BFGS method for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*. 2001;11(4):1054-64.
3. Lewis AS, Overton ML. Nonsmooth optimization via quasi-Newton methods. *Mathematical Programming*. 2013 Oct;141(1):135-63.
4. Nash SG, Nocedal J. A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization. *SIAM Journal on Optimization*. 1991 Aug;1(3):358-72.
5. Shewchuk JR. An introduction to the conjugate gradient method without the agonizing pain.
6. Nash SG. Preconditioning of truncated-Newton methods. *SIAM Journal on Scientific and Statistical Computing*. 1985 Jul;6(3):599-616.



Recommended Reading List

Hessian-Free Optimization

1. Liu DC, Nocedal J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*. 1989 Aug;45(1):503-28.
2. Li DH, Fukushima M. On the global convergence of the BFGS method for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*. 2001;11(4):1054-64.
3. Lewis AS, Overton ML. Nonsmooth optimization via quasi-Newton methods. *Mathematical Programming*. 2013 Oct;141(1):135-63. **BFGS-Type Quasi-Newton Methods**

4. Nash SG, Nocedal J. A numerical study of the limited memory BFGS method and the truncated-Newton method for large scale optimization. *SIAM Journal on Optimization*. 1991 Aug;1(3):358-72.
5. Shewchuk JR. An introduction to the conjugate gradient method without the agonizing pain.
6. Nash SG. Preconditioning of truncated-Newton methods. *SIAM Journal on Scientific and Statistical Computing*. 1985 Jul;6(3):599-616. **Newton-CG Method**

Quasi-Newton Methods



Recall Newton's Method

By second-order Taylor expansion,

$$f(\mathbf{x}) \approx \hat{f}(\mathbf{x}) \triangleq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_k)^T \nabla^2 f(\mathbf{x}_k) (\mathbf{x} - \mathbf{x}_k)$$

Minimizing quadratic approximation

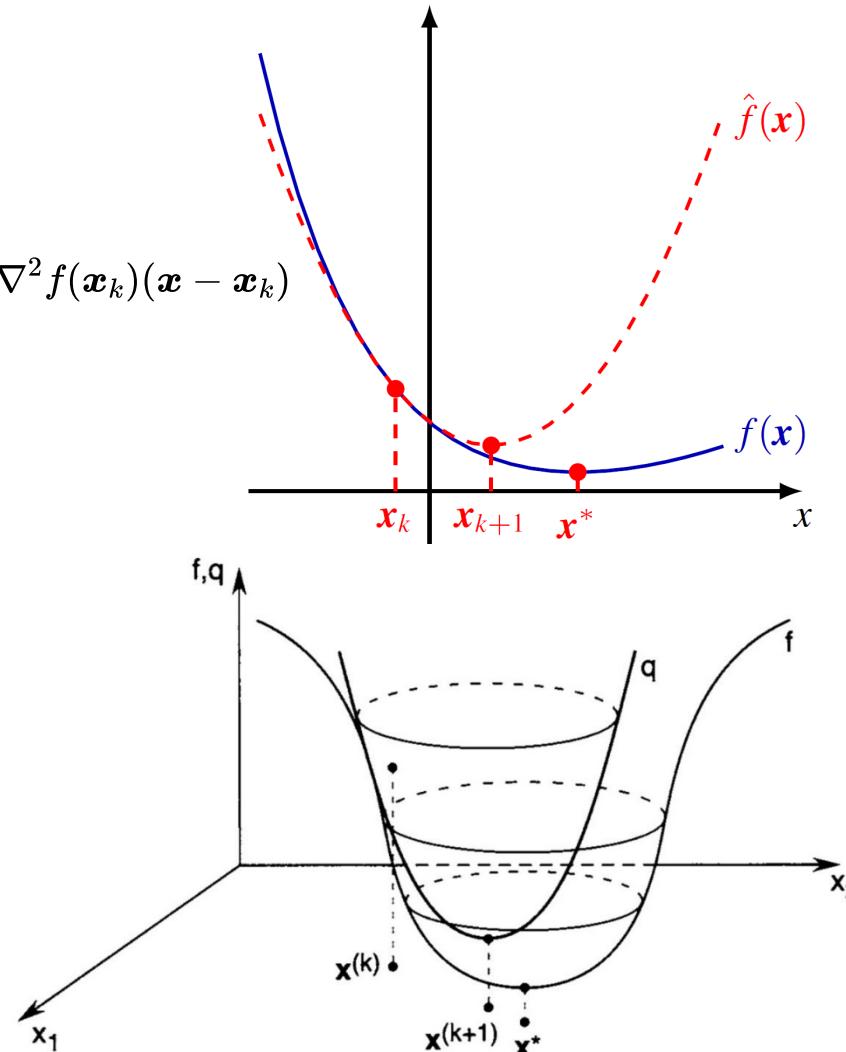
$$\begin{aligned}\nabla \hat{f}(\mathbf{x}) &= \nabla^2 f(\mathbf{x}_k) (\mathbf{x} - \mathbf{x}_k) + \nabla f(\mathbf{x}_k) = \mathbf{0} \\ \implies \mathbf{x} &= \mathbf{x}_k - [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)\end{aligned}$$

provided $\nabla^2 f(\mathbf{x}_k) \succ O$ Positive Definite (PD)

Newton step

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k - [\nabla^2 f(\mathbf{x}_k)]^{-1} \nabla f(\mathbf{x}_k)}$$

Note. If the function is quadratic, then Newton's method gets to the optimum in a single step.

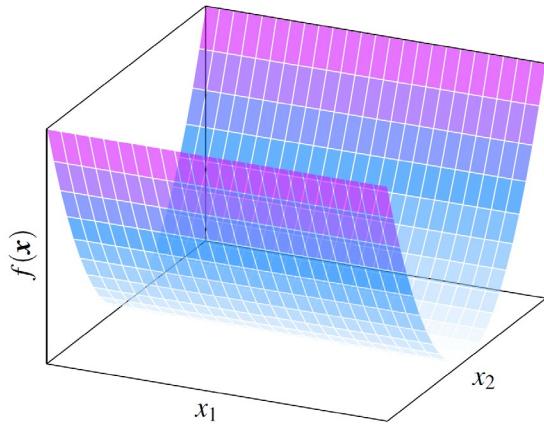




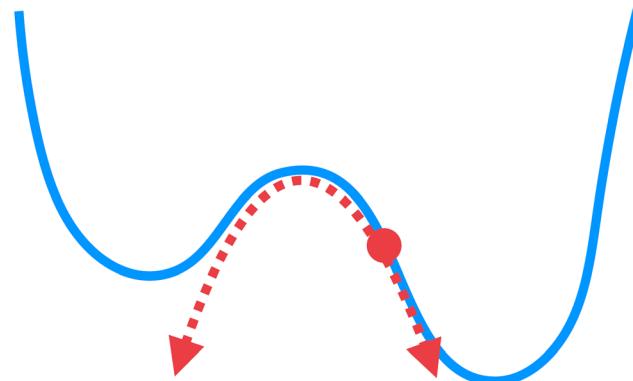
Recall Newton's Method

Drawbacks: In practice Hessian can be singular and indefinite

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$



Singular Hessian!



If start at a point of negative curvature, the Newton goes up!

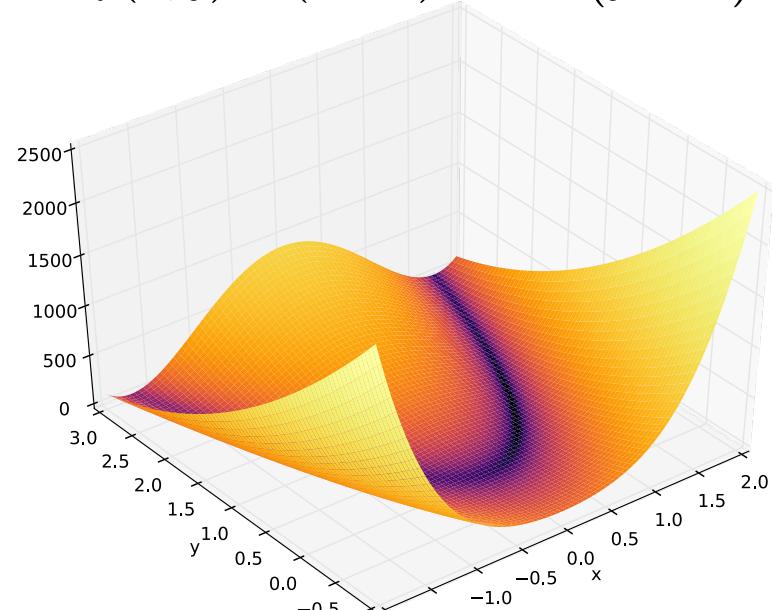
(negative) search direction: must form an acute angle with the gradient



Why Quasi-Newton Methods

- Gradient descent is cheap
 - Compute partials: $O(N)$
- Newton's method is expensive
 - Compute mixed partials: $O(N^2)$
 - Linear solver: $O(N^3)$
- Other problems
 - When quadratic approximation is bad, Newton's is a waste
 - Hessian becomes poorly conditioned
 - Nonconvex problems: indefinite Hessian = ascent step

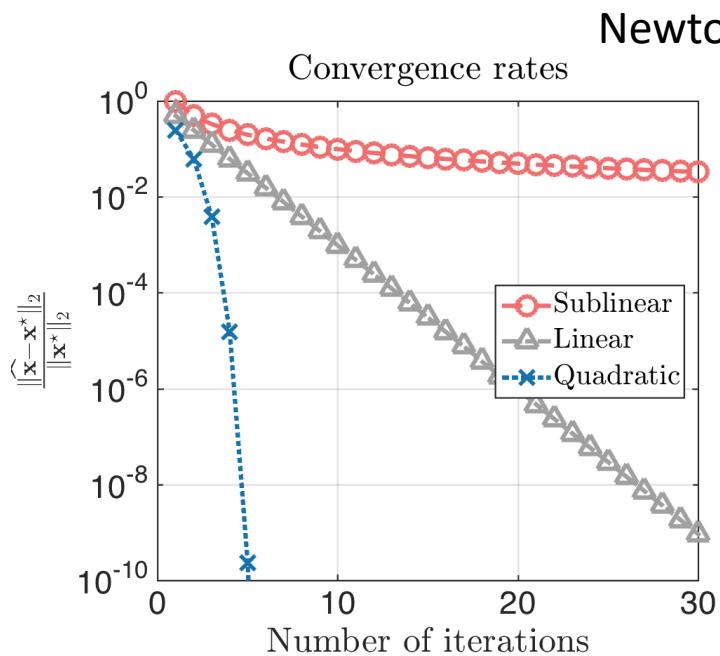
Rosenbrock banana function
 $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$





Why Quasi-Newton Methods

Normally, for **convex smooth** functions, we have



Newton's Method

Gradient Methods

Linear

$$\mathbf{e}^k = \mathbf{x}^k - \mathbf{x}^*$$

Quadratic

$$\|\mathbf{e}^{k+1}\| = C \|\mathbf{e}^k\|$$

Superlinear

$$\|\mathbf{e}^{k+1}\| = C \|\mathbf{e}^k\|^2$$

$$\|\mathbf{e}^{k+1}\| = C \|\mathbf{e}^k\|^p, p > 1$$

As good as quadratic (why?)

Attain **super-linear performance** without all the fuss



Quasi-Newton Methods

Newton approximation (PD Hessian H)

$$f(x) - f(x^k) \approx (x - x^k)^T g + \frac{1}{2} (x - x^k)^T H (x - x^k)$$

Solve $H^k d^k = -g^k$

Quasi-Newton approximation

$$f(x) - f(x^k) \approx (x - x^k)^T g^k + \frac{1}{2} (x - x^k)^T M^k (x - x^k)$$

Solve $M^k d^k = -g^k$

What conditions should M satisfy?

- It should not need full 2nd-order derivatives
- The linear equations have closed-form solutions
- It should be lightweight and compact to store
- **It must preserves descent directions**
- **It should contain curvature information (local quadratic apprx.)**



Quasi-Newton Methods

When is descent direction preserved?

$$M^k d^k = -g^k$$

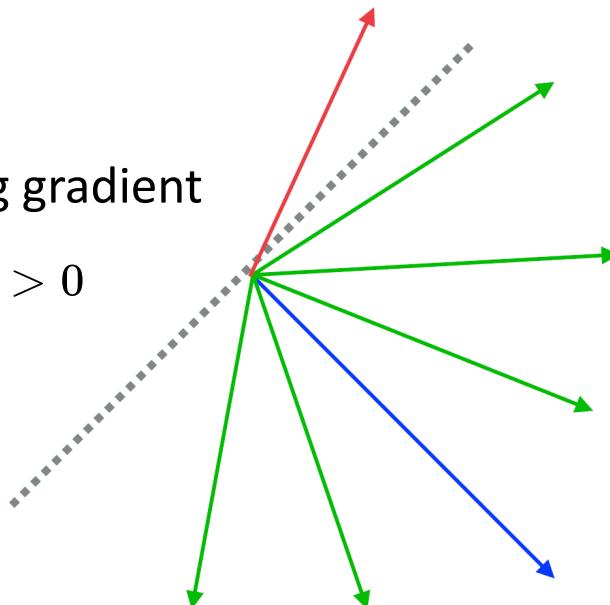
Search direction

Negative gradient step

Search direction makes acute angle with neg gradient

$$\langle -g, -M^{-1}g \rangle = \langle g, M^{-1}g \rangle = g^T M^{-1}g > 0$$

Must be PD (Positive Definite)





Quasi-Newton Methods

Which kind of curvature info is needed?

$$\nabla f(x) - \nabla f(y) \approx H(x - y)$$

$$\Delta x = x^{k+1} - x^k$$

$$\Delta g = \nabla f(x^{k+1}) - \nabla f(x^k)$$

Secant
condition



$$\Delta g \approx M^{k+1} \Delta x$$

or

$$\boxed{\Delta x \approx B^{k+1} \Delta g, \quad M^{k+1} B^{k+1} = I}$$

Construct B directly from gradient/variable diff



Quasi-Newton Methods

Infinitely many B satisfy the secant condition?

$$\Delta x = B^{k+1} \Delta g$$

How to choose the best one?

- Assume the function is convex
- Assume an initial guess is given

$$\min_B \|B - B^k\|^2$$

$$\text{s.t. } B = B^T$$

$$\Delta x = B \Delta g$$



scale invariant

$$\min_B \|H^{\frac{1}{2}}(B - B^k)H^{\frac{1}{2}}\|^2$$

$$\text{s.t. } B = B^T$$

$$\Delta x = B \Delta g$$

$$H = \int_0^1 \nabla^2 f[(1-\tau)x^k + \tau x^{k+1}] d\tau$$



Quasi-Newton Methods

Recover the curvature (approx. inv. Hessian) from gradients

Broyden-Fletcher-Goldfarb-Shanno (BFGS) update

$$B^{k+1} = \left(I - \frac{\Delta x \Delta g^T}{\Delta g^T \Delta x} \right) B^k \left(I - \frac{\Delta g \Delta x^T}{\Delta g^T \Delta x} \right) + \frac{\Delta x \Delta x^T}{\Delta g^T \Delta x}$$

$$B^0 = I, \Delta x = x^{k+1} - x^k, \Delta g = \nabla f(x^{k+1}) - \nabla f(x^k)$$

What conditions should $M = 1/B$ satisfy?

- It should not need full 2nd-order derivatives ✓
- The linear equations have closed-form solutions ✓
- It should be lightweight and compact to store ✓
- It must preserves descent directions ? strict convexity (necessary?) ✓
- It should contain curvature information ✓

BFGS update **preserves PD** if $\Delta g^T \Delta x > 0$



Quasi-Newton Methods

BFGS method for **strictly convex** functions

```
initialize  $x^0, g^0 \leftarrow \nabla f(x^0), B^0 \leftarrow I, k \leftarrow 0$ 
while  $\|g^k\| > \delta$  do
     $d \leftarrow -B^k g^k$ 
     $t \leftarrow$  backtracking line search (Armijo)
     $x^{k+1} \leftarrow x^k + td$ 
     $g^{k+1} \leftarrow \nabla f(x^{k+1})$ 
     $B^{k+1} \leftarrow \text{BFGS}(B^k, g^{k+1} - g^k, x^{k+1} - x^k)$ 
     $k \leftarrow k + 1$ 
end while
return
```

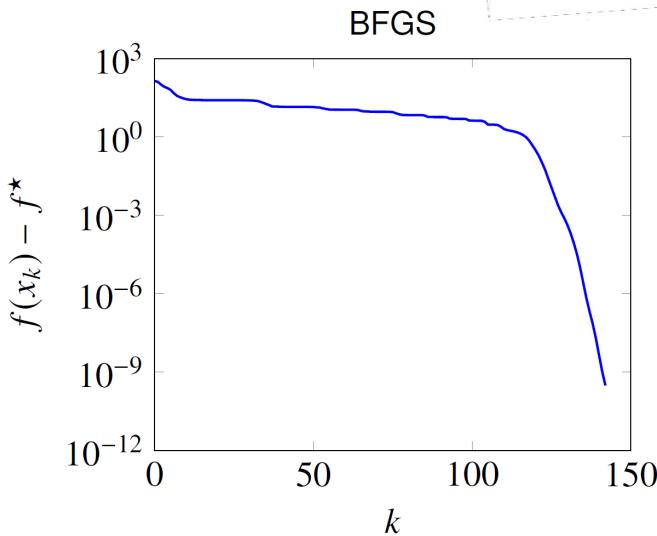
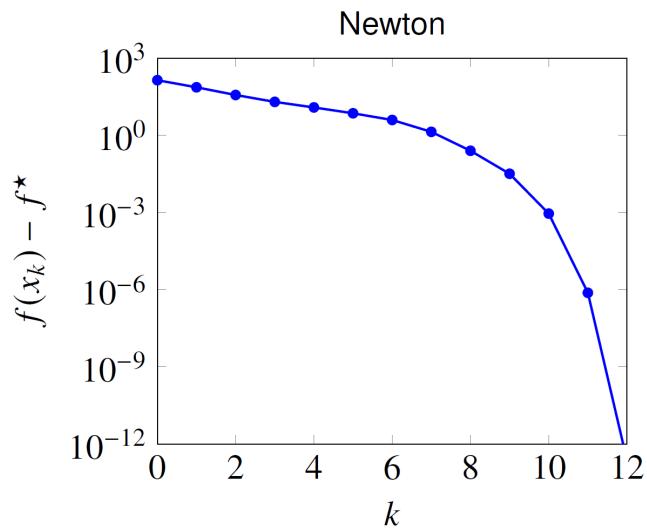
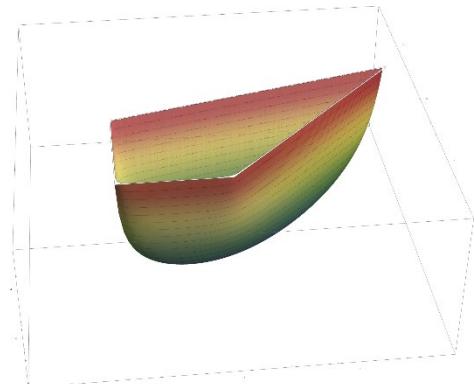


Quasi-Newton Methods

Example

$$f(x) = c^T x - \sum_{i=1}^m \log(b_i - a_i^T x), \quad x \in \mathbb{R}^n$$

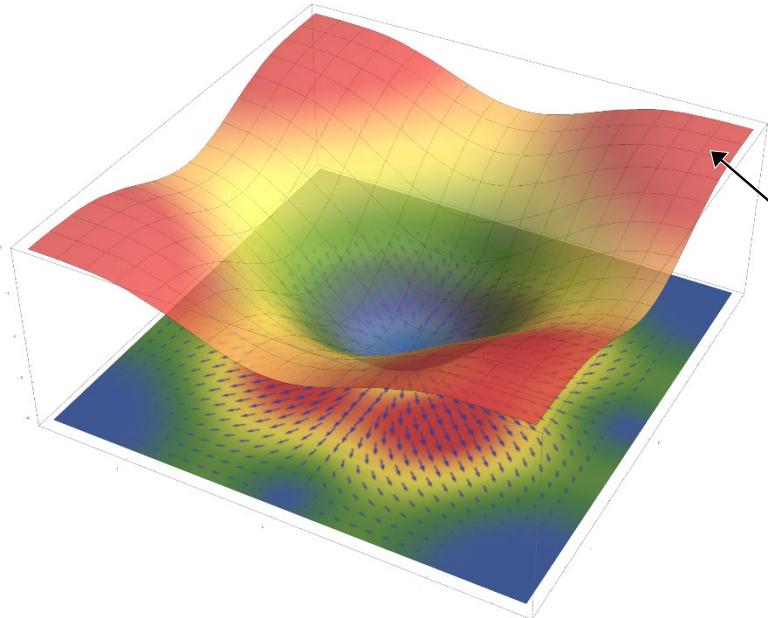
$$n = 100, m = 500$$



- cost per Newton iteration: $O(n^3)$
- cost per BFGS iteration: $O(n^2)$



Quasi-Newton Methods



- negative curvature
- nonconvex
- non-monotonic grad

Restrictions:

- Strict gradient monotonicity does not hold in general
- Curvature info far from optimum is less important
- Iteration cost is quadratic in dim
- Applicability to nonconvex function remains to be verified
- Applicability to nonsmooth function remains to be verified

How to apply to more nonconvex/nonsmooth functions?

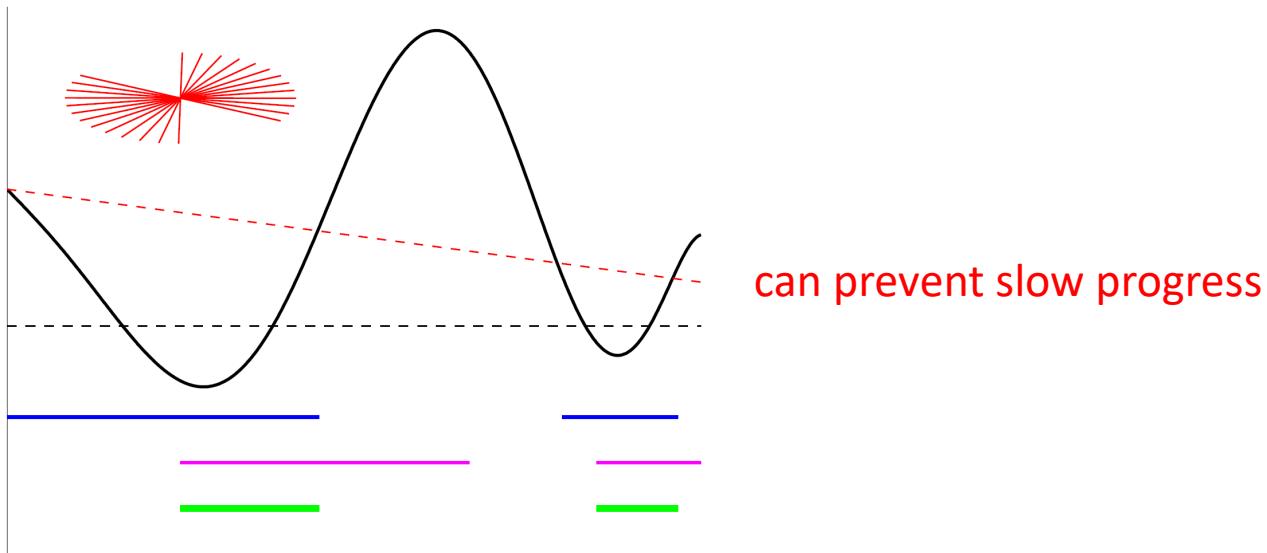


Quasi-Newton Methods

$\Delta g^T \Delta x > 0$? Ensure PD of approximation: Wolfe conditions!

1. weak Wolfe conditions $0 < c_1 < c_2 < 1$ typically $c_1 = 10^{-4}$, $c_2 = 0.9$

$$\begin{aligned} f(x^k) - f(x^k + \alpha d) &\geq -c_1 \cdot \alpha d^T \nabla f(x^k) && \text{sufficient decrease condition} \\ d^T \nabla f(x^k + \alpha d) &\geq c_2 \cdot d^T \nabla f(x^k) && \text{curvature condition} \end{aligned}$$



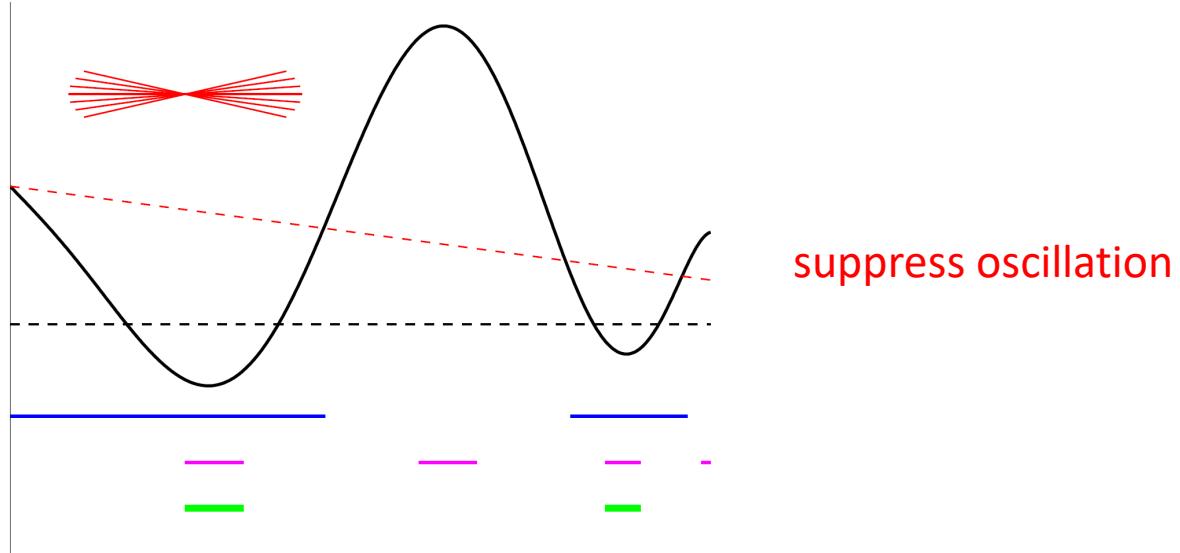


Quasi-Newton Methods

$\Delta g^T \Delta x > 0$? Ensure PD of approximation: Wolfe conditions!

2. strong Wolfe conditions $0 < c_1 < c_2 < 1$ typically $c_1 = 10^{-4}$, $c_2 = 0.9$

$$f(x^k) - f(x^k + \alpha d) \geq -c_1 \cdot \alpha d^T \nabla f(x^k) \quad \text{sufficient decrease condition}$$
$$|d^T \nabla f(x^k + \alpha d)| \leq |c_2 \cdot d^T \nabla f(x^k)| \quad \text{strong curvature condition}$$





Quasi-Newton Methods

strong Wolfe conditions



weak Wolfe conditions



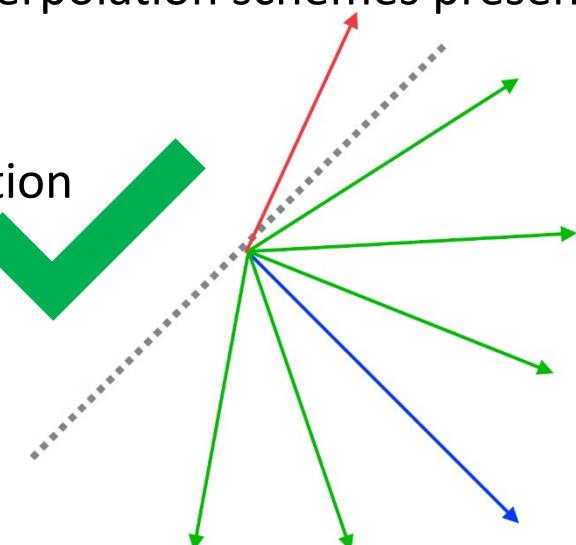
$$\Delta g^T \Delta x > 0$$



BFGS update is PD

Inexact line search suffices (detailed later)
Various interpolation schemes presented

descent direction





Quasi-Newton Methods

Wolfe conditions **cannot guarantee** the convergence of BFGS

The **cautious update** (Li and Fukushima 2001) with mild conditions is needed.

$$B^{k+1} = \begin{cases} \left(I - \frac{\Delta x \Delta g^T}{\Delta g^T \Delta x} \right) B^k \left(I - \frac{\Delta g \Delta x^T}{\Delta g^T \Delta x} \right) + \frac{\Delta x \Delta x^T}{\Delta g^T \Delta x} & \text{if } \Delta g^T \Delta x > \epsilon ||g_k|| \Delta x^T \Delta x, \quad \epsilon = 10^{-6} \\ B^k & \text{otherwise} \end{cases}$$

The cautious update guarantees zero grad to be an accumulation point if

- The func has **bounded sub-level sets**
- The func has **Lipschitz continuous grad**



Quasi-Newton Methods

BFGS method for **possibly nonconvex** functions

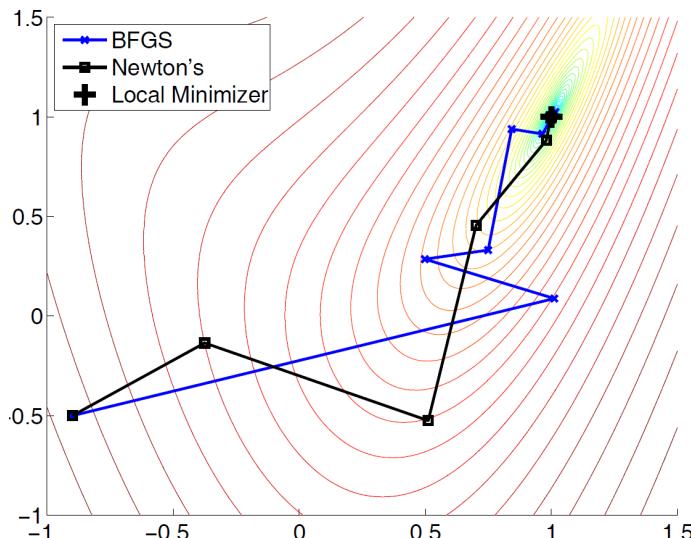
```
initialize  $x^0, g^0 \leftarrow \nabla f(x^0), B^0 \leftarrow I, k \leftarrow 0$ 
while  $\|g^k\| > \delta$  do
     $d \leftarrow -B^k g^k$ 
     $t \leftarrow$  inexact line search (Wolfe)
     $x^{k+1} \leftarrow x^k + td$ 
     $g^{k+1} \leftarrow \nabla f(x^{k+1})$ 
     $B^{k+1} \leftarrow$  Cautious-BFGS $(B^k, g^{k+1} - g^k, x^{k+1} - x^k)$ 
     $k \leftarrow k + 1$ 
end while
return
```



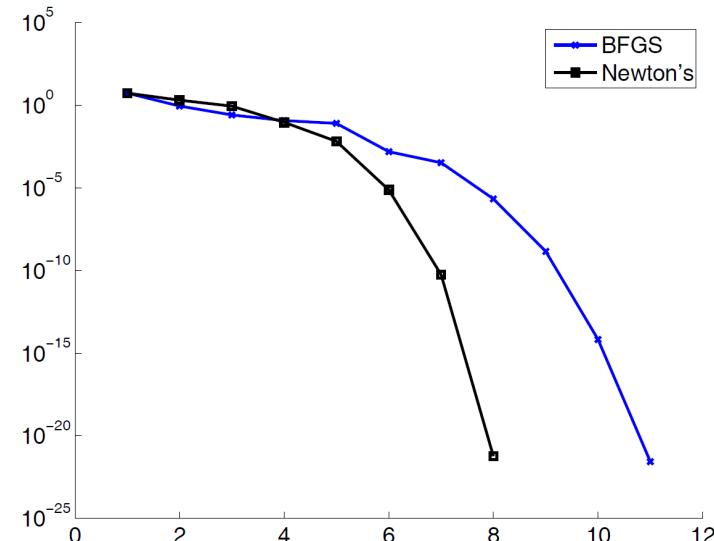
Quasi-Newton Methods

Example

$$f(x) = (1 - x_1)^2 + (x_2 - x_1^2)^2$$



(a) Paths followed.

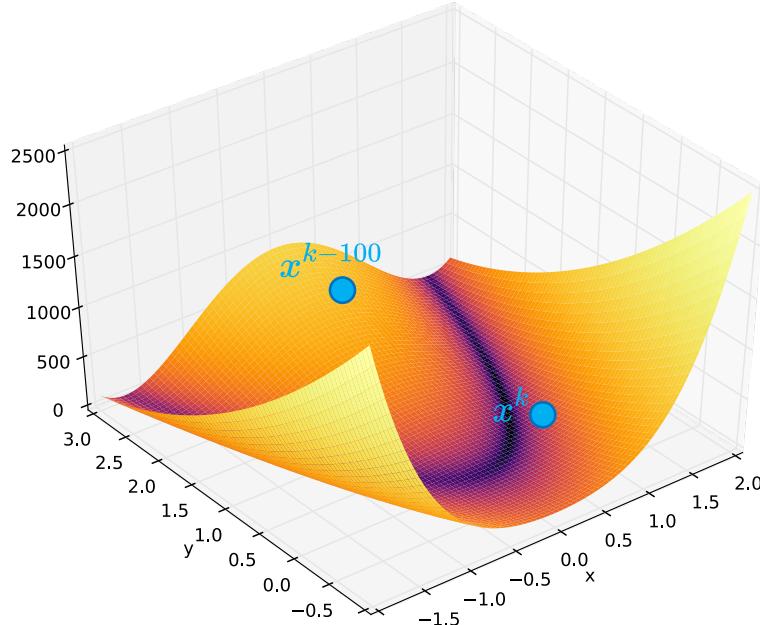


(b) Function value at each iterate.

Satisfactory convergence rate



Quasi-Newton Methods



- Is the curvature info at x^{k-100} useful to iteration at x^k ?
- Is it possible to further reduce the $O(n^2)$ cost per iteration?

Only exploit last $m+1$ pairs of $\{x^k, g^k\}$!

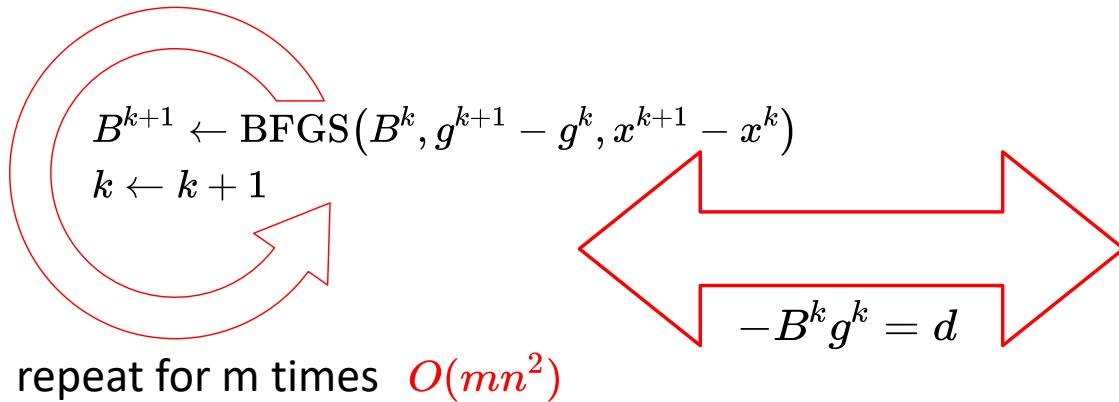


Quasi-Newton Methods

Limited-memory BFGS (**L-BFGS**): do not store B^k explicitly

$$s^k = x^{k+1} - x^k, \quad y^k = g^{k+1} - g^k, \quad \rho^k = 1/\langle s^k, y^k \rangle \text{ where } \langle a, b \rangle := a^T b$$

- Instead we store up to m (e.g., m = 30) values of s^k, y^k, ρ^k

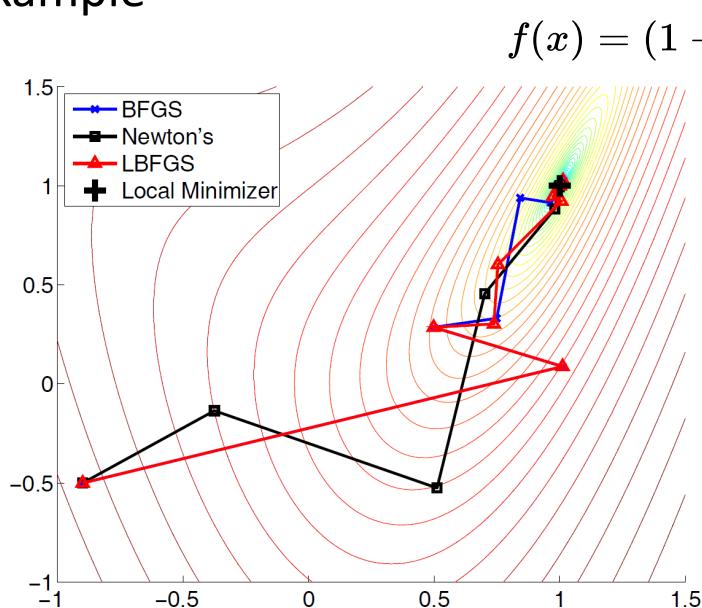


```
d ← gk O(mn)
for i = k - 1, k - 2, ..., k - m
    αi ← ρi⟨si, d⟩
    d ← d - αiyi
end (for)
γ ← ρk-1⟨yk-1, yk-1⟩
d ← d / γ
for i = k - m, k - m + 1, ..., k - 1
    β ← ρi⟨yi, d⟩
    d ← d + si(αi - β)
end (for)
return search direction d
```

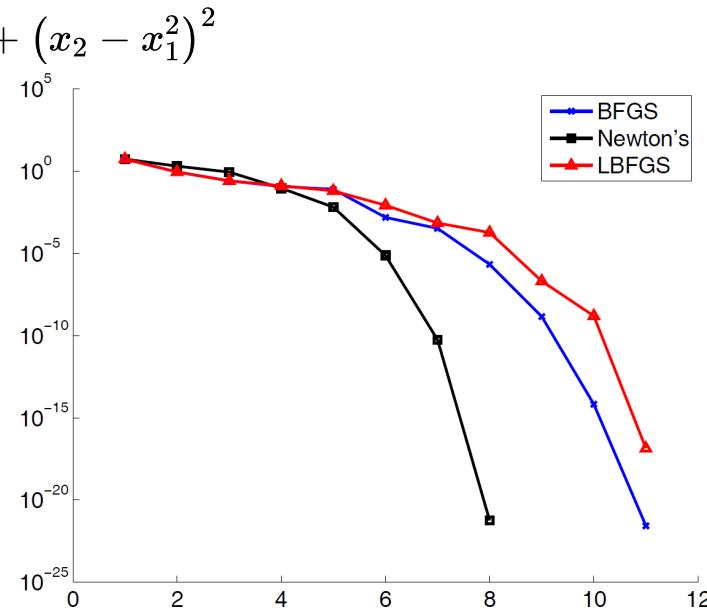


Quasi-Newton Methods

Example



(a) Paths followed.



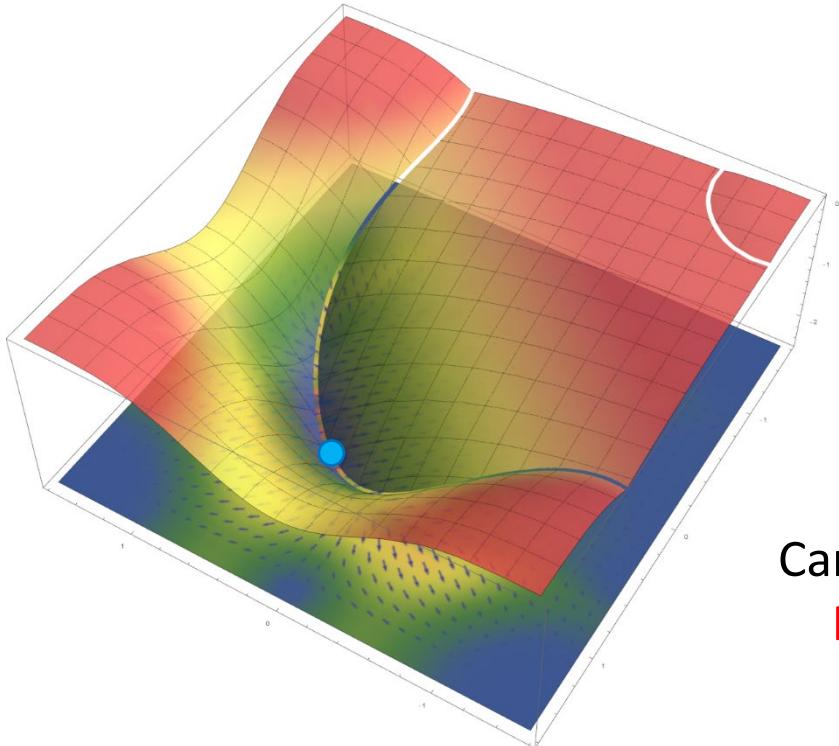
(b) Function value at each iterate.

	Newton's	BFGS	LBFGS
Work per iteration	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(mn)$

L-BFGS is almost the 1st choice for efficient smooth nonconvex opt



Quasi-Newton Methods



What if functions are nonsmooth?

Troubles with nonsmoothness :

- Gradient may not exist
- Negative sub-grad does not descent
- Curvature can be very large

Can we apply L-BFGS to nonsmooth function?
Practically yes! (Lewis and Overton 2013)



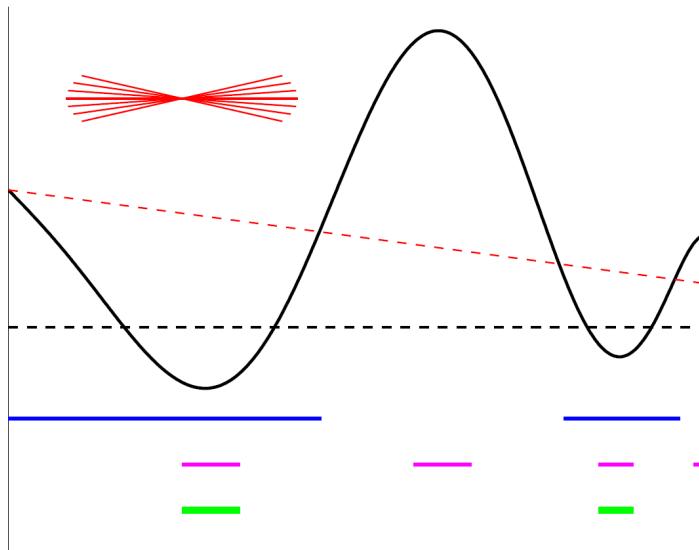
Quasi-Newton Methods

Key issues when applying L-BFGS to nonsmooth functions

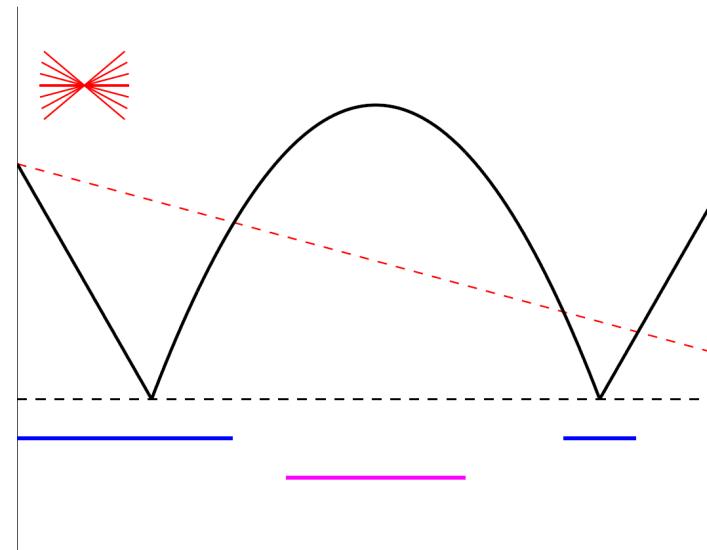
Recall: **strong Wolfe conditions**

$$\begin{aligned} f(x^k) - f(x^k + \alpha d) &\geq -c_1 \cdot \alpha d^T \nabla f(x^k) \\ |d^T \nabla f(x^k + \alpha d)| &\leq |c_2 \cdot d^T \nabla f(x^k)| \end{aligned}$$

sufficient decrease condition
strong curvature condition



(a) $Smooth \phi(\alpha)$.



(b) $nonsmooth \phi(\alpha)$. **conditions fail!**



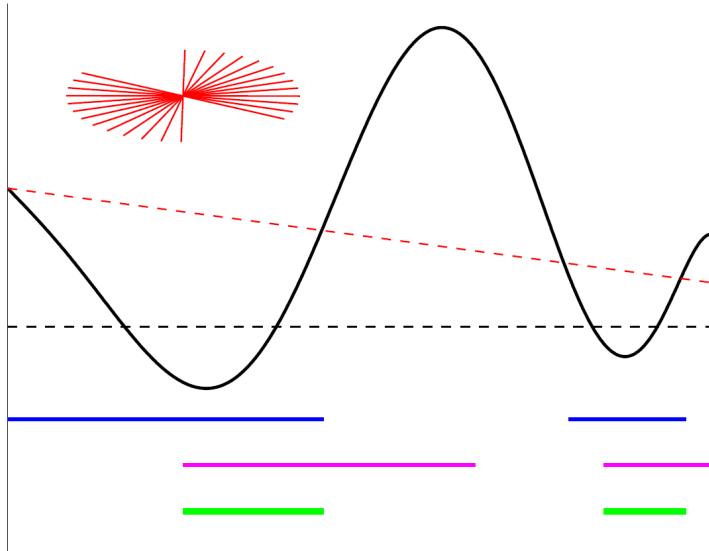
Quasi-Newton Methods

Key issues when applying L-BFGS to nonsmooth functions

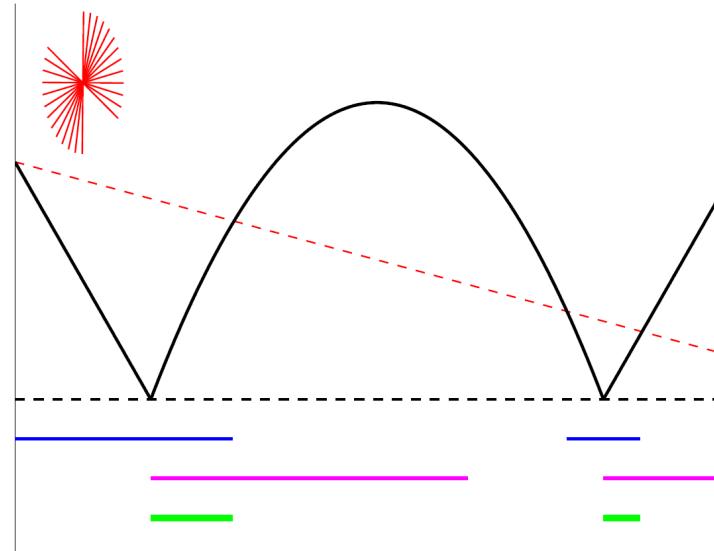
Recall: **weak Wolfe conditions**

$$f(x^k) - f(x^k + \alpha d) \geq -c_1 \cdot \alpha d^T \nabla f(x^k)$$
$$d^T \nabla f(x^k + \alpha d) \geq c_2 \cdot d^T \nabla f(x^k)$$

sufficient decrease condition
curvature condition



(a) *Smooth $\phi(\alpha)$.*



(b) *nonsmooth $\phi(\alpha)$. conditions work!*

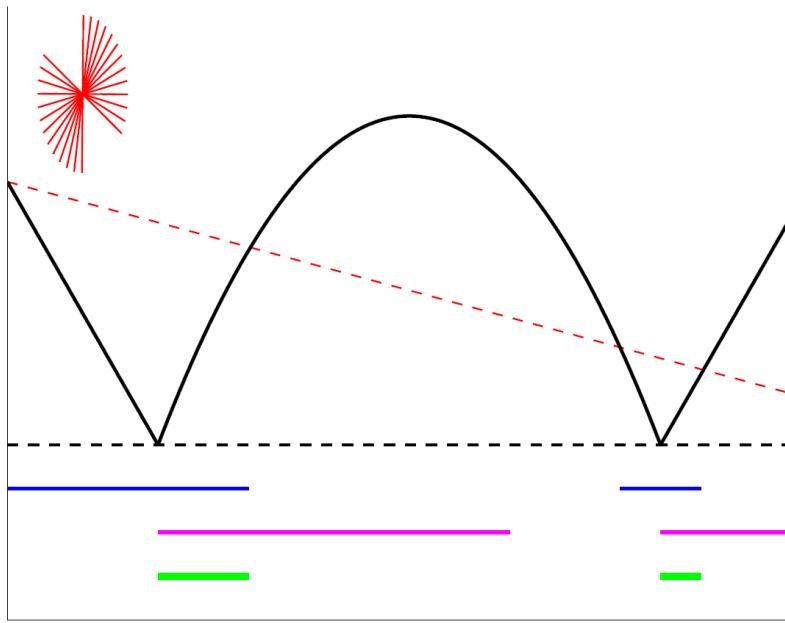


Quasi-Newton Methods

weak Wolfe conditions should be used
for nonsmooth functions

$$S(\alpha) : f(x^k) - f(x^k + \alpha d) \geq -c_1 \cdot \alpha d^T \nabla f(x^k)$$

$$C(\alpha) : d^T \nabla f(x^k + \alpha d) \geq c_2 \cdot d^T \nabla f(x^k)$$



Lewis & Overton line search:

- weak Wolfe conditions
- no interpolation used

$l \leftarrow 0$

$u \leftarrow +\infty$

$\alpha \leftarrow 1$

repeat

if $S(\alpha)$ fails

$u \leftarrow \alpha$

else if $C(\alpha)$ fails

$l \leftarrow \alpha$

else

return α

if $u < +\infty$

$\alpha \leftarrow (l + u)/2$

else

$\alpha \leftarrow 2l$

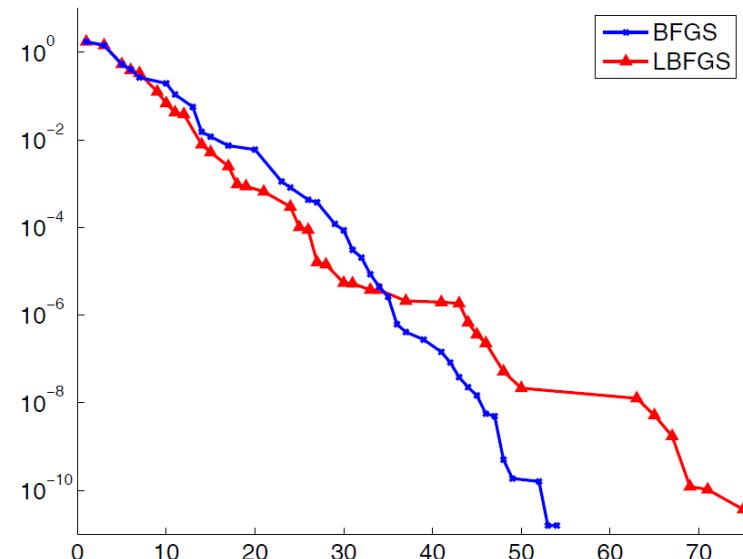
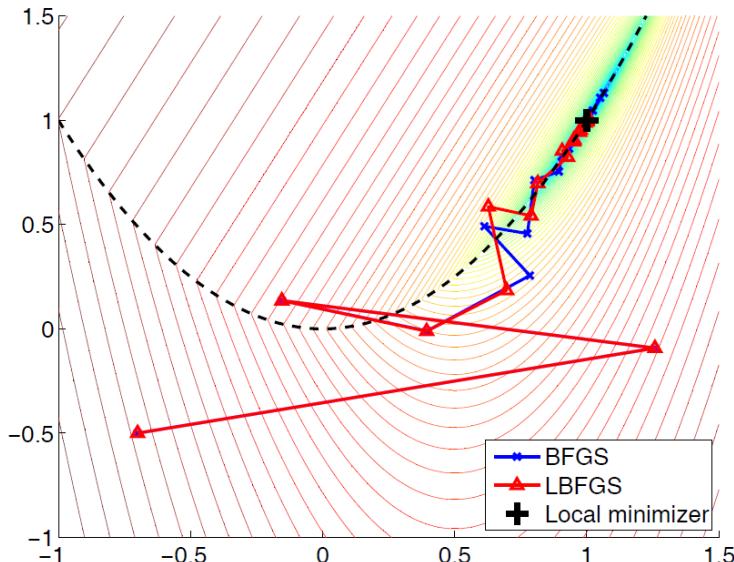
end (repeat)



Quasi-Newton Methods

Example

$$f(x) = (1 - x_1)^2 + |x_2 - x_1^2|$$



(b) Function value at each iterate after the line search.

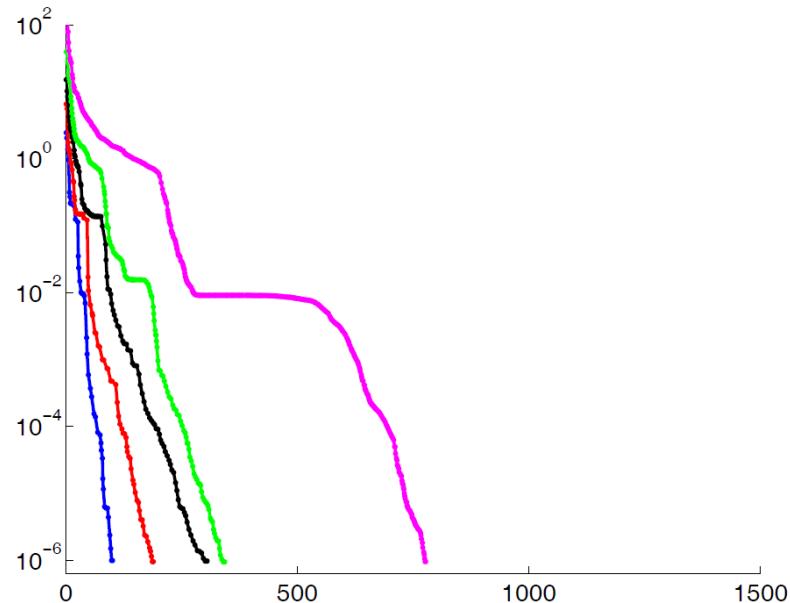


Quasi-Newton Methods

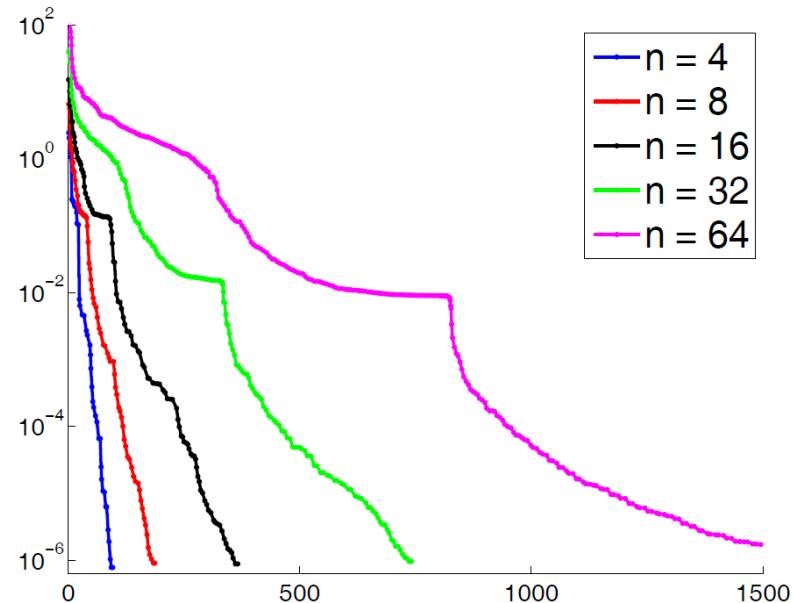
Example

$$f(x) = 4\|Ax\| + 3e_1^T Ax, \kappa(A) = n^2, f^* \equiv 0$$

Ill-conditioned nonsmooth func whose A is PD and randomly generated



LBFGS with $m = 25$



BFGS



Quasi-Newton Methods

L-BFGS method for **possibly nonconvex nonsmooth** functions

```
initialize  $x^0, g^0 \leftarrow \nabla f(x^0), B^0 \leftarrow I, k \leftarrow 0$ 
while  $\|g^k\| > \delta$  do
     $d \leftarrow -B^k g^k$ 
     $t \leftarrow$  Lewis Overton line search
     $x^{k+1} \leftarrow x^k + td$ 
     $g^{k+1} \leftarrow \nabla f(x^{k+1})$ 
     $B^{k+1} \leftarrow$  Cautious-Limited-Memory-BFGS( $g^{k+1} - g^k, x^{k+1} - x^k$ )
     $k \leftarrow k + 1$ 
end while
return
```



Quasi-Newton Methods

Now we have detailed a practically powerful quasi-newton method:

BFGS Update

+

Cautious Update

+

Limited Memory Two-Loop Recursion

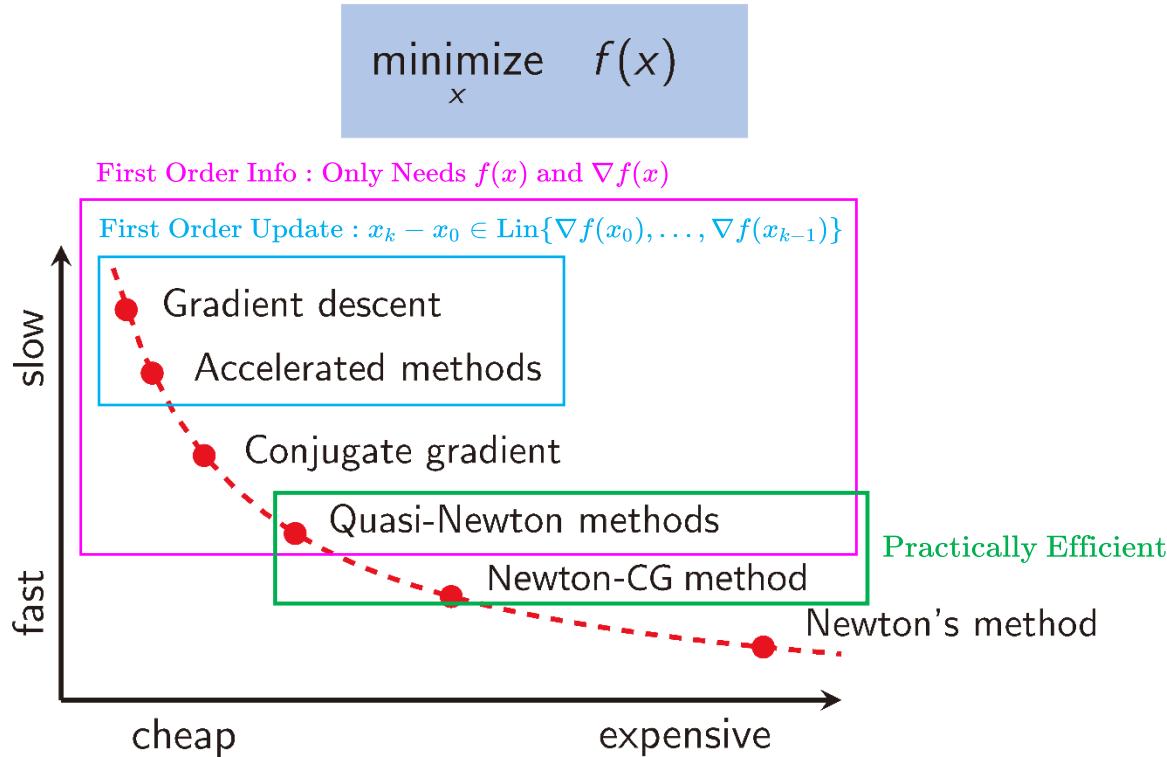
+

Lewis-Overton Line Search



Quasi-Newton Methods

Is quasi-Newton class the only way to avoid Hessian?



Newton-CG Method



Newton-CG Method

Complexity of function/gradient/Hessian

$$f : \mathbb{R}^n \rightarrow \mathbb{R} \quad = \quad \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \quad < \quad \nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \ddots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Function Gradient Hessian

About computational complexity :

- Computing gradient is at least $O(n)$
- Computing gradient is **as easy as** computing the function
- Computing Hessian is at least $O(n^2)$ whose inversion is about $O(n^3)$



Newton-CG Method

Fortunately



Newton-CG Method

Complexity of function/gradient/Hessian-vec product

$$f : \mathbb{R}^n \rightarrow \mathbb{R} \quad = \quad \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \quad = \quad (\nabla^2 f)\xi = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_n \end{pmatrix}$$

Function Gradient Hessian-vec product

Even if exact value is not available, just compute $(\nabla^2 f)\xi \approx \frac{\nabla f(x + \delta\xi) - \nabla f(x)}{\delta}$

However, we need inverse-Hessian-vec product.
What can be exploited via Hessian-vec product?



Newton-CG Method

The most prominent large-scale linear solver:
(Linear) Conjugate Gradient Method

$$Ax = b$$

$A \in \mathbb{R}^{n \times n}$: a known PD mat

$x \in \mathbb{R}^n$: an unknown vec

$b \in \mathbb{R}^n$: a known vec

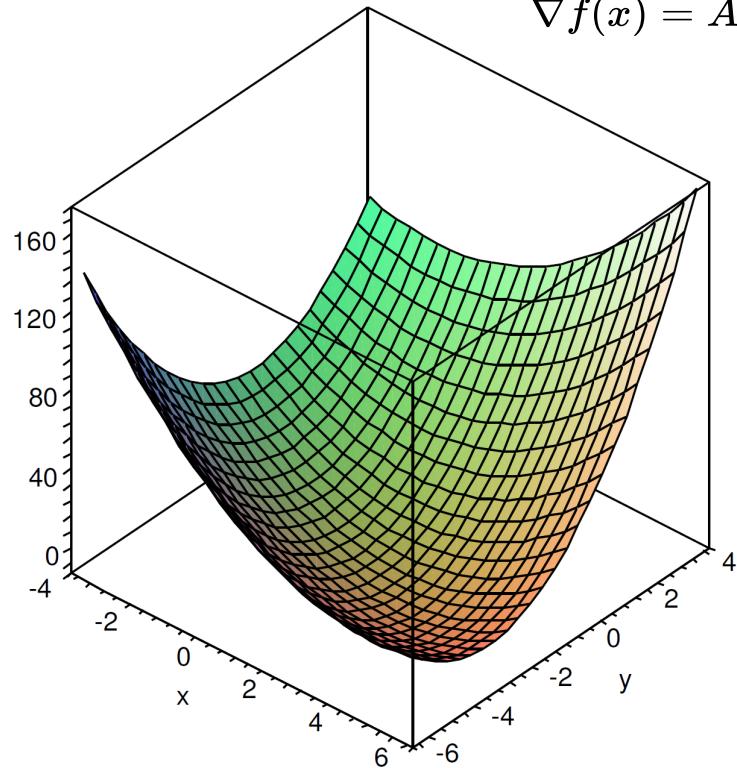
Solve $Ax = b$ via the interface $\gamma(x) := Ax$



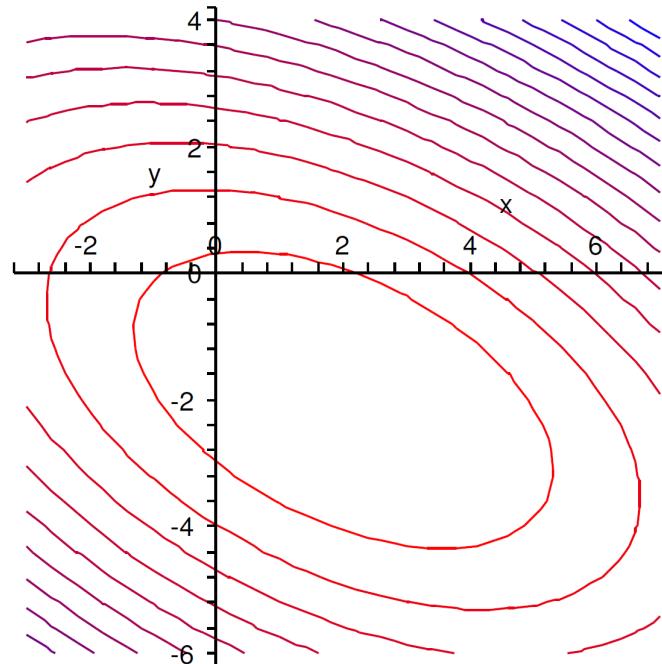
Newton-CG Method

(Linear) Conjugate Gradient Method

$$Ax = b \quad \nabla f(x) = Ax - b = 0$$



$$\arg \min_x f(x) = \frac{1}{2} x^T A x - b^T x$$

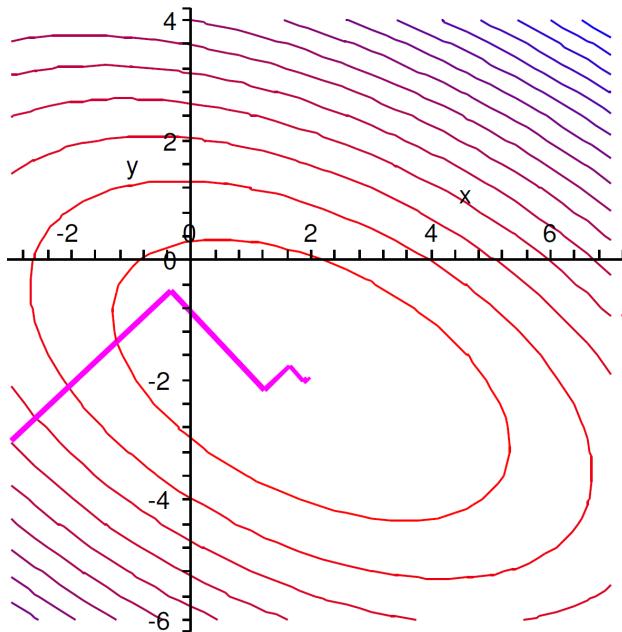




Newton-CG Method

(Linear) Conjugate Gradient Method

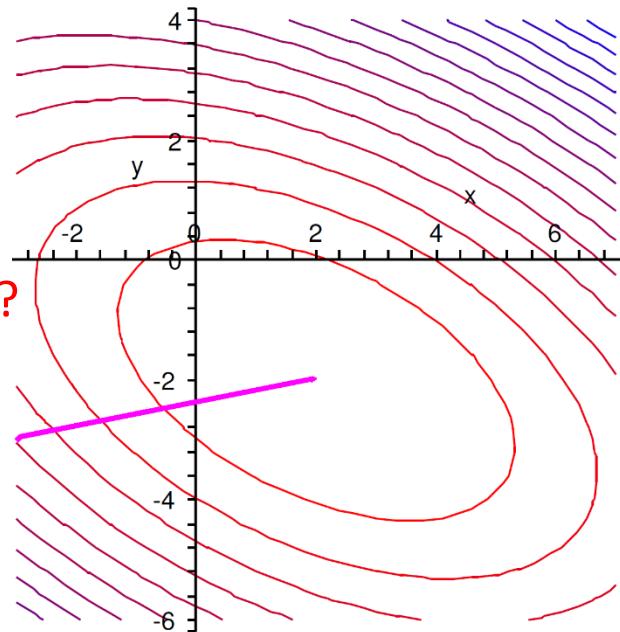
Recall steepest gradient descent



Infinite steps for an exact solution

Any trade-off?

Recall Newton's method

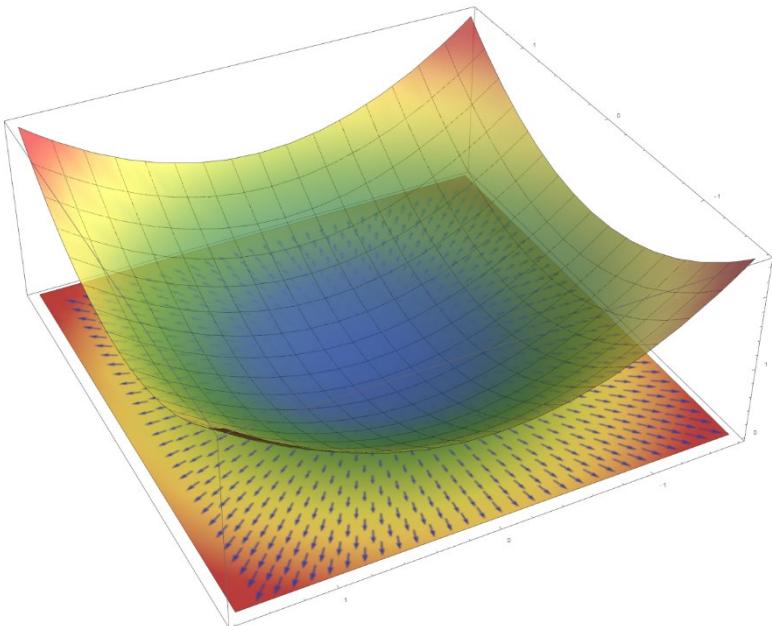


One step for an exact solution but with Hessian-inv

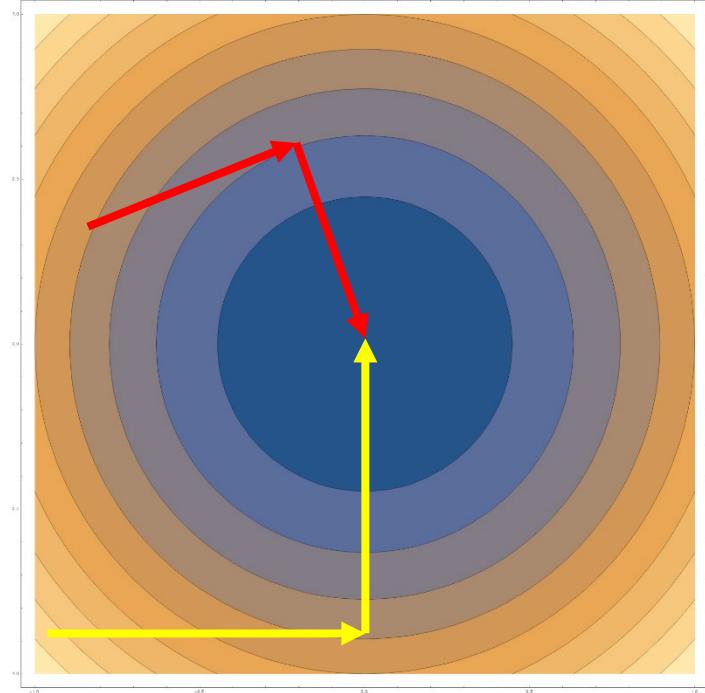


Newton-CG Method

(Linear) Conjugate Gradient Method



A simple case: $f(x) = \frac{1}{2}x^T x - b^T x$

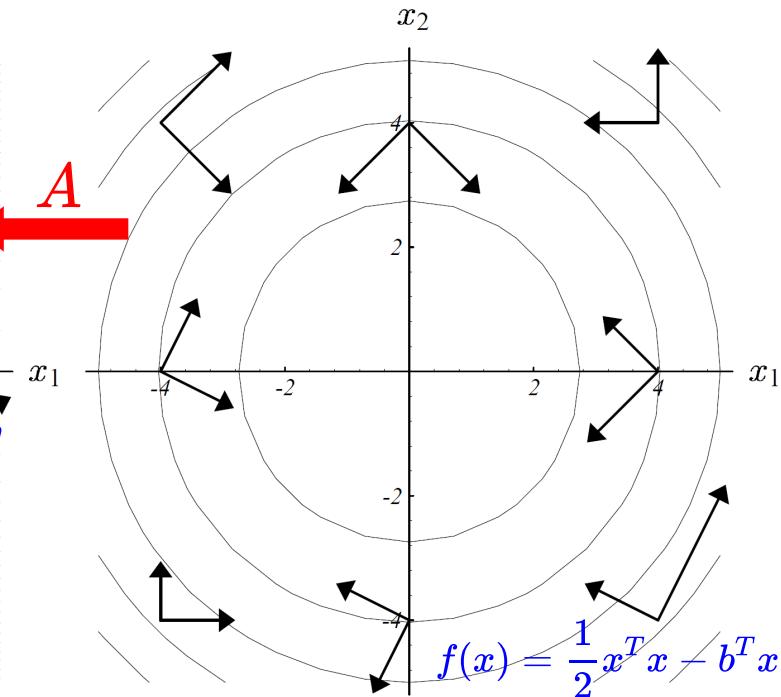
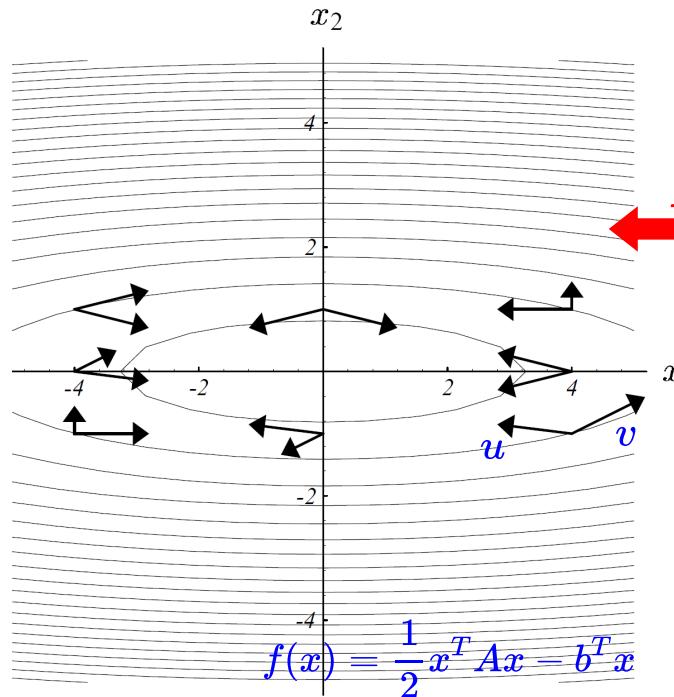


At most n orthogonal steps with exact line search



Newton-CG Method

(Linear) Conjugate Gradient Method

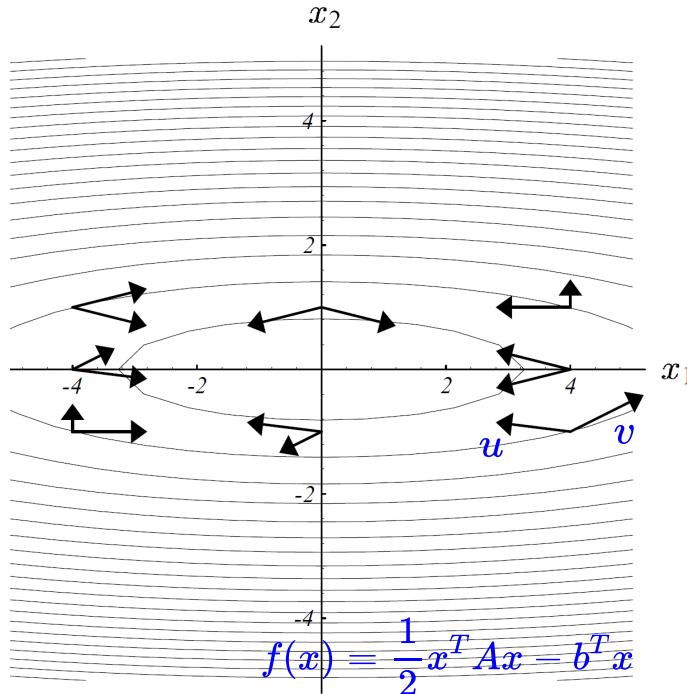


u and v is conjugate \iff $u^T A v = 0$ \iff u and v is A -orthogonal



Newton-CG Method

(Linear) Conjugate Gradient Method



Given n mutually conjugate directions

$$u^1, u^2, \dots, u^n$$

Successively conduct **closed-form exact line search**

$$\alpha \leftarrow \frac{b^T u^k - (x^k)^T [Au^k]}{(u^k)^T [Au^k]}$$

interface $\gamma(u^k)$

$$x^{k+1} \leftarrow x^k + \alpha u^k$$

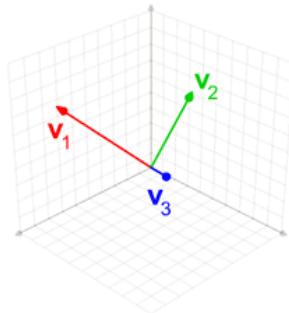
Obtain an exact solution in n steps

How to find conjugate directions?



Newton-CG Method

(Linear) Conjugate Gradient Method



$$\mathbf{u}^1 = \mathbf{v}^1,$$

$$\mathbf{u}^2 = \mathbf{v}^2 - \text{proj}_{\mathbf{u}^1}(\mathbf{v}^2),$$

$$\mathbf{u}^3 = \mathbf{v}^3 - \text{proj}_{\mathbf{u}^1}(\mathbf{v}^3) - \text{proj}_{\mathbf{u}^2}(\mathbf{v}^3),$$

$$\mathbf{u}^4 = \mathbf{v}^4 - \text{proj}_{\mathbf{u}^1}(\mathbf{v}^4) - \text{proj}_{\mathbf{u}^2}(\mathbf{v}^4) - \text{proj}_{\mathbf{u}^3}(\mathbf{v}^4),$$

⋮

$$\mathbf{u}^k = \mathbf{v}^k - \sum_{j=1}^{k-1} \text{proj}_{\mathbf{u}^j}(\mathbf{v}^k),$$

$$\mathbf{e}^1 = \frac{\mathbf{u}^1}{\|\mathbf{u}^1\|}$$

$$\mathbf{e}^2 = \frac{\mathbf{u}^2}{\|\mathbf{u}^2\|}$$

$$\mathbf{e}^3 = \frac{\mathbf{u}^3}{\|\mathbf{u}^3\|}$$

$$\mathbf{e}^4 = \frac{\mathbf{u}^4}{\|\mathbf{u}^4\|}$$

⋮

$$\mathbf{e}^k = \frac{\mathbf{u}^k}{\|\mathbf{u}^k\|}.$$

conjugate Gram–Schmidt process

Gram–Schmidt process

$$\text{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{u}, \mathbf{v} \rangle_A}{\langle \mathbf{u}, \mathbf{u} \rangle_A} \mathbf{u}, \quad \langle \mathbf{u}, \mathbf{v} \rangle_A = \mathbf{u}^T A \mathbf{v}$$



$$\text{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u}$$

construct conjugate directions via (conjugate) Gram–Schmidt process



Newton-CG Method

(Linear) Conjugate Gradient Method

linearly independent v_1, v_2, \dots, v_k



conjugate Gram–Schmidt process



mutually conjugate u_1, u_2, \dots, u_k

$$u^k = v^k - \sum_{j=1}^{k-1} \text{proj}_{u^j}(v^k)$$

but the recurrence is too long!



Newton-CG Method

(Linear) Conjugate Gradient Method Chooses v_1, v_2, \dots, v_k Such That

- All linearly independent vectors are generated **on the fly**
- The conjugate Gram–Schmidt enjoys a **short recurrence**



Newton-CG Method

(Linear) Conjugate Gradient Method

If the linearly independent vector is chosen as

$$\mathbf{v}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k$$

The history-dependent coefficients in conjugate G-S process becomes

$$\text{proj}_{\mathbf{u}^j}(\mathbf{v}^k) = 0, \forall j \leq k-2$$

Thus we get a short recurrence

$$\mathbf{u}^k = \mathbf{v}^k - \sum_{j=1}^{k-1} \text{proj}_{\mathbf{u}^j}(\mathbf{v}^k) = \mathbf{v}^k + \frac{\|\mathbf{v}^k\|^2}{\|\mathbf{v}^{k-1}\|^2} \mathbf{u}^{k-1}$$



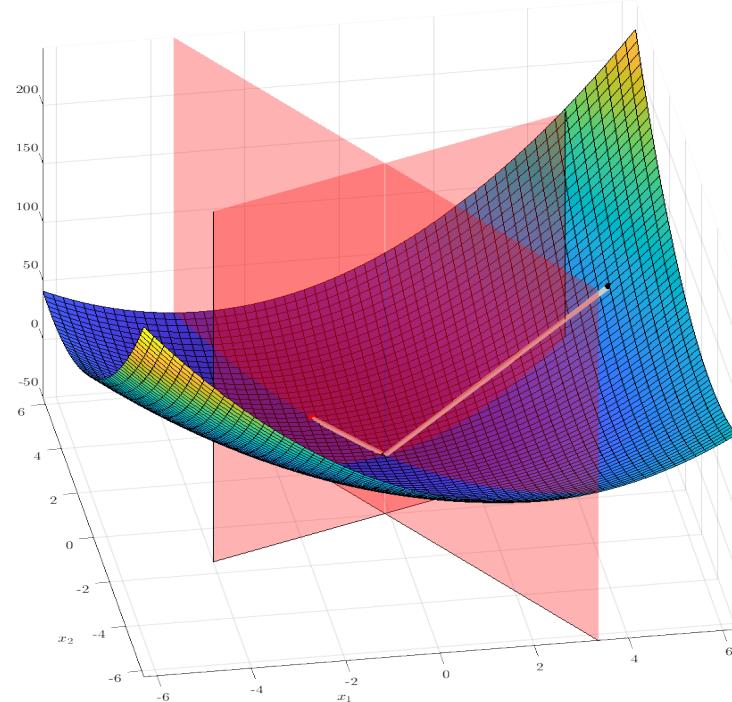
Newton-CG Method

(Linear) Conjugate Gradient Method

```
x1 ← xini
v1 ← b - Ax1
u1 ← v1
k ← 1
while vk ≠ 0
    α ← ||vk||2 / (uk)T Auk
    xk+1 ← xk + αuk
    vk+1 ← vk - αAuk
    β ← ||vk+1||2 / ||vk||2
    uk+1 ← vk+1 + βuk
    k ← k + 1
end (while)
return xk
```

float-point precision:
 $\|v^k\| \geq \epsilon$

interface $\gamma(u^k)$



If exact arithmetic is used, CG always finds an exact solution in n steps



Newton-CG Method

(Linear) Conjugate Gradient Method

```
x1 ← xini
v1 ← b - Ax1
u1 ← v1
k ← 1
while vk ≠ 0
    α ← \|vk\|^2 / (uk)T Auk
    xk+1 ← xk + αuk
    vk+1 ← vk - αAuk
    β ← \|vk+1\|^2 / \|vk\|^2
    uk+1 ← vk+1 + βuk
    k ← k + 1
end (while)
return xk
```

float-point precision:
 $\|v^k\| \geq \epsilon$

Features:

- CG can either be interpreted as a direct or iterative solver
- CG is often used with preconditioner
- CG only requires mat-vec product
- CG directly exploits mat sparsity without producing fill-in
- CG normally converges fast

$A \in \mathbb{R}^{555 \times 555}$, $\kappa(A) > 10^{10}$		
Method	Number of Iterations	
Gauss Seidel	208,000	
Block SOR methods	765	
Preconditioned conjugate gradient	25	



Newton-CG Method

Newton-CG Method

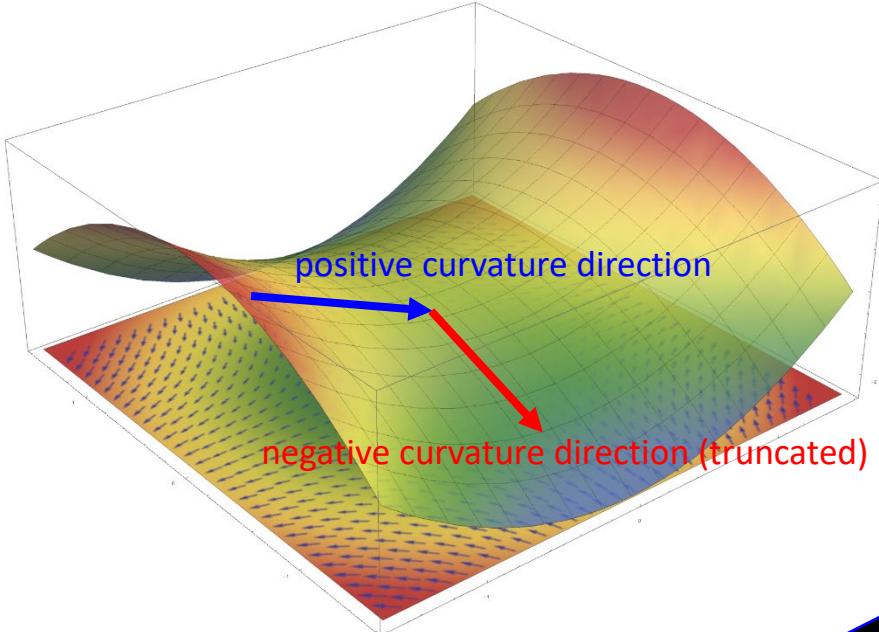
Apply CG to solve $(\nabla^2 f)d = -\nabla f$ via the interface $\gamma(\xi) := (\nabla^2 f)\xi$

- How to handle indefinite Hessian?
- Should the linear system be solved exactly?



Newton-CG Method

Newton-CG Method



Truncated CG:

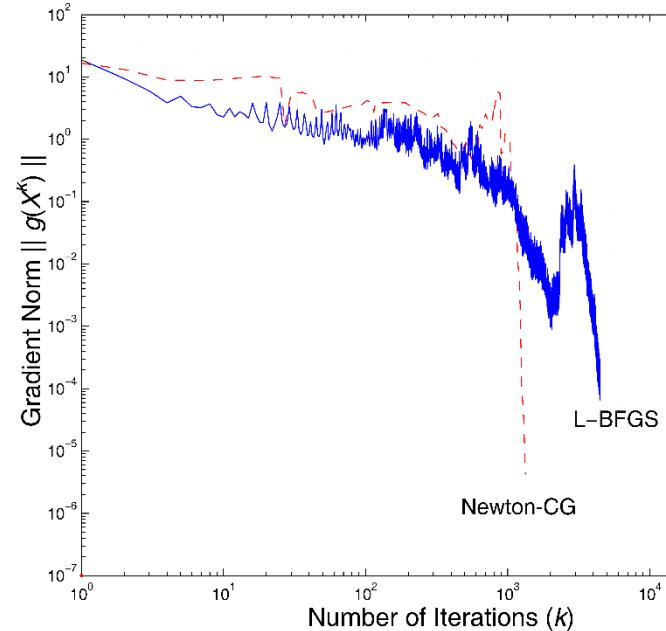
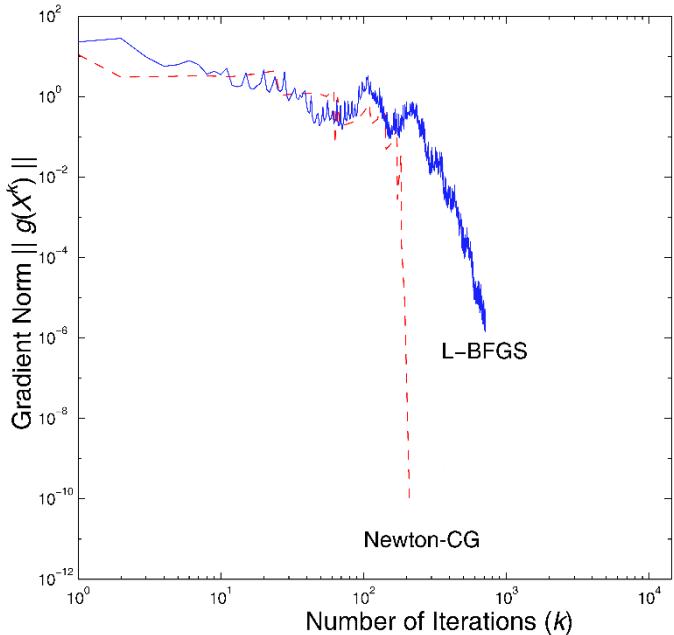
- Start with zero as the initial direction
- Truncating negative curvature direction

```
x1 ← xini, k ← 1
while ‖∇f(xk)‖ > δ
    εk ← min(1, ‖∇f(xk)‖)/10
    d1 ← 0, v1 ← −∇f(xk)
    u1 ← v1, j ← 1
    while ‖vj‖ > εk ‖∇f(xk)‖
        if (uj)T ∇2f(xk) uj ≤ 0
            if j = 1
                dj ← −∇f(xk)
            end (if)
        break
    end (if)
    α ← ‖vj‖2 / (uj)T ∇2f(xk) uj
    dj+1 ← dj + α uj
    vj+1 ← vj − α A uj
    β ← ‖vj+1‖2 / ‖vj‖2
    uj+1 ← vj+1 + β uj
    j ← j + 1
end (while)
α ← backtracking line search (Armijo)
xk+1 ← xk + α dj
end (while)
```



Newton-CG Method

Newton-CG Method



Molecular Dynamics Simulation (66 variables)

Minimizer	Iters.(Hess.-vec calls.)	f	$\ \nabla f\ $	$f & \nabla$ evals.	CPU time
Newton-CG	29(210)	-15.25	7.67×10^{-11}	44	1.12 sec.
L-BFGS	711	-15.25	1.4×10^{-6}	740	1.39

Molecular Dynamics Simulation (1704 variables)

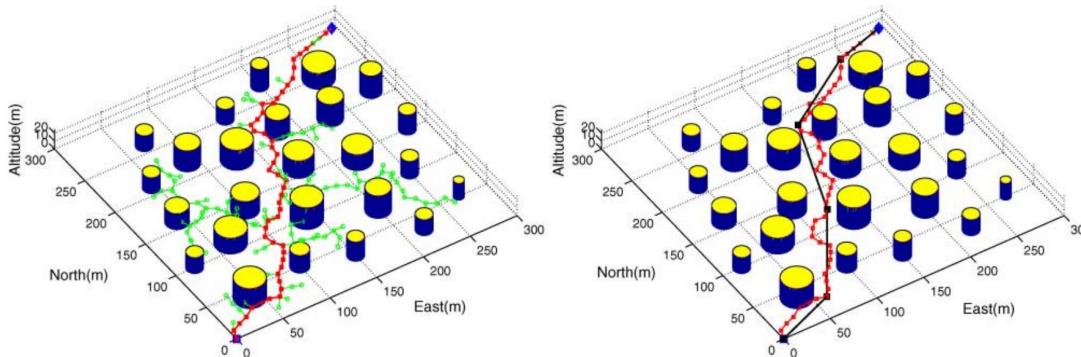
Minimizer	Iters.(Hess.-vec calls.)	f	$\ \nabla f\ $	$f & \nabla$ evals.	CPU time
Newton-CG	65(1335)	-2773.70	4.2×10^{-6}	240	5.21 min.
L-BFGS	4486	-2792.96	6.3×10^{-5}	4622	12.61

Newton-CG normally achieves lower gradient norm.

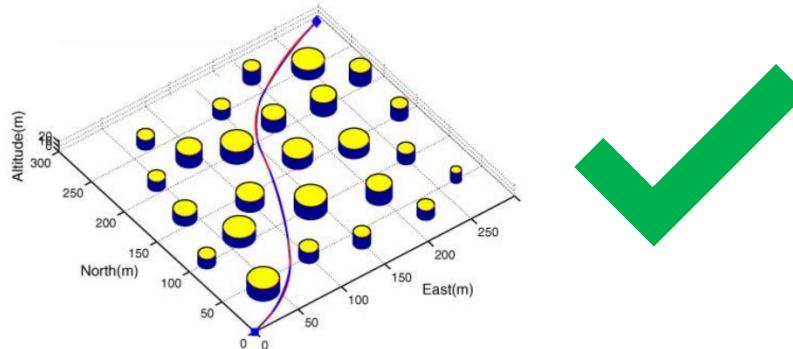
Application: Smooth Navigation Path Generation



Smooth Navigation Path Generation



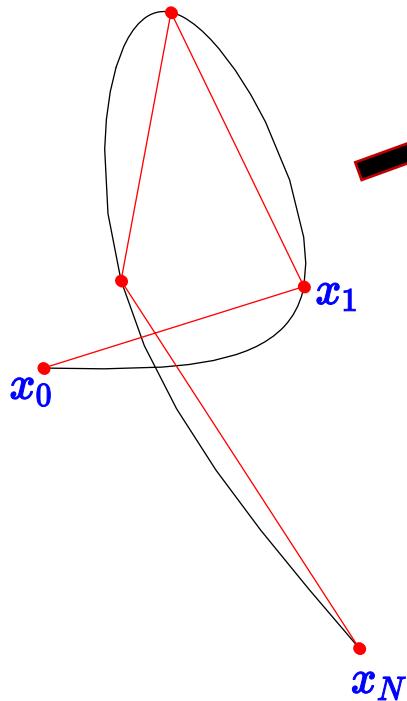
- RRT*-like planners generate low-dim paths efficiently.
- Nonsmooth paths are unsatisfactory for mobile robots.
- Unconstrained opt can efficiently smooth these paths.





Smooth Navigation Path Generation

Cubic Spline (**Minimum Stretch Energy**) Path Parameterization



The i -th segment $p_i(s) = a_i + b_i s + c_i s^2 + d_i s^3, s \in [0, 1]$

Curve and its 1st/2nd order derivative are continuous.

$$C^2 : p_{i-1}(1) = p_i(0), p_{i-1}^{(1)}(1) = p_i^{(1)}(0), p_{i-1}^{(2)}(1) = p_i^{(2)}(0),$$

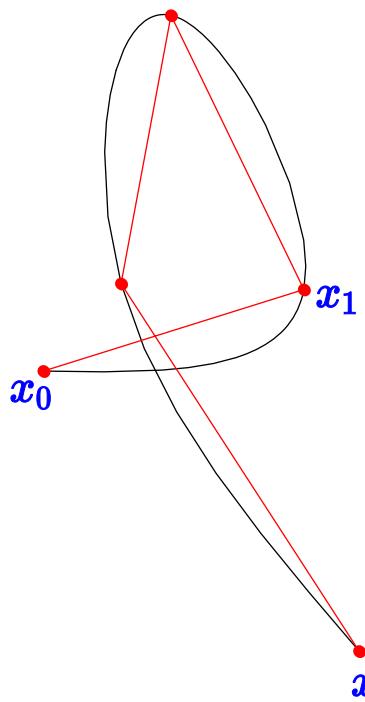
Assume stationary boundary conditions.

$$p_1^{(1)}(0) = 0, p_N^{(1)}(1) = 0$$



Smooth Navigation Path Generation

Cubic Spline (Minimum Stretch Energy) Path Parameterization



A cubic curve sequentially crossing x_0, x_1, \dots, x_N is given by

$$a_i = x_i$$

$$b_i = D_i$$

$$c_i = 3(x_{i+1} - x_i) - 2D_i - D_{i+1}$$

$$d_i = 2(x_i - x_{i+1}) + D_i + D_{i+1}$$

where

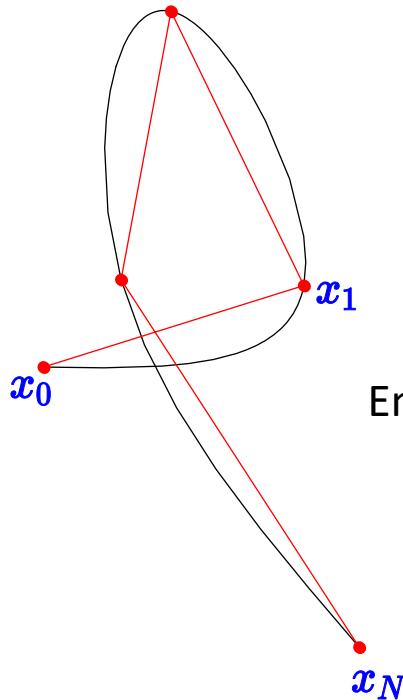
$$\begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ \vdots \\ D_{n-2} \\ D_{n-1} \end{bmatrix} = \begin{bmatrix} 4 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 3(x_2 - x_0) \\ 3(x_3 - x_1) \\ 3(x_4 - x_2) \\ 3(x_5 - x_3) \\ \vdots \\ 3(x_{n-1} - x_{n-3}) \\ 3(x_n - x_{n-2}) \end{bmatrix}, \text{ and } D_0 = D_N = 0$$

Thus the entire trajectory is determined by x_0, x_1, \dots, x_N



Smooth Navigation Path Generation

Cubic Spline (**Minimum Stretch Energy**) Path Generation



Minimize the stretch energy to make it as smooth as possible

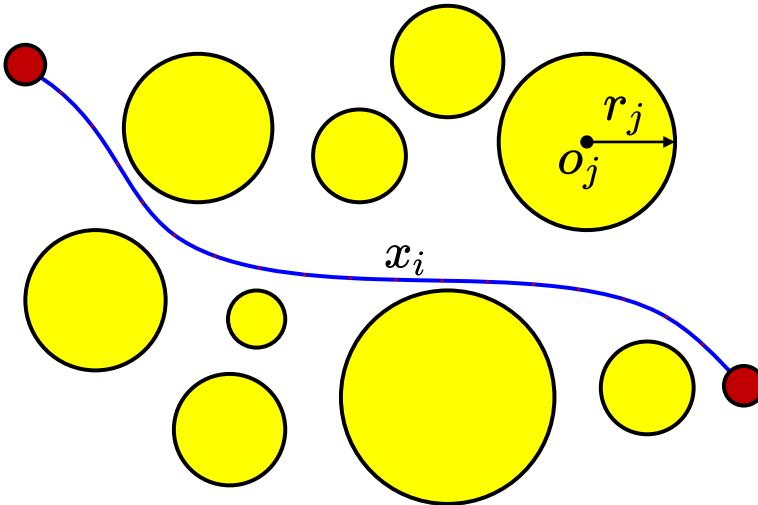
$$\text{Energy}(x_1, x_2, \dots, x_{N-1}) = \sum_{i=0}^N \int_0^1 \|p_i^{(2)}(s)\|^2 ds$$

Energy is a function of points since all coefficients are determined by points.



Smooth Navigation Path Generation

Cubic Spline (**Minimum Stretch Energy**) Path Generation



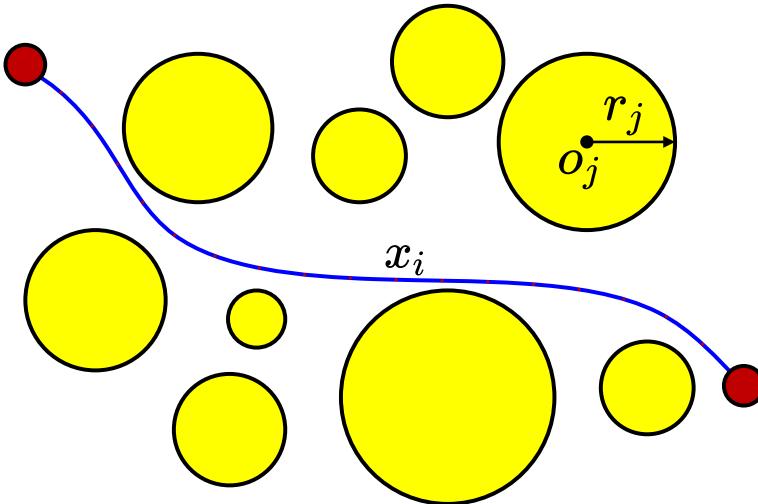
$$\text{Potential}(x_1, x_2 \dots, x_{N-1}) = 1000 \sum_{i=1}^{N-1} \sum_{j=1}^M \max(r_j - \|x_i - o_j\|, 0)$$

Ensure the curve is collision-free via potential functions!



Smooth Navigation Path Generation

Cubic Spline (**Minimum Stretch Energy**) Path Generation



Complete the smooth path generation by unconstrained minimization:

$$\min_{x_1, x_2 \dots, x_{N-1}} \text{Energy}(x_1, x_2 \dots, x_{N-1}) + \text{Potential}(x_1, x_2 \dots, x_{N-1})$$

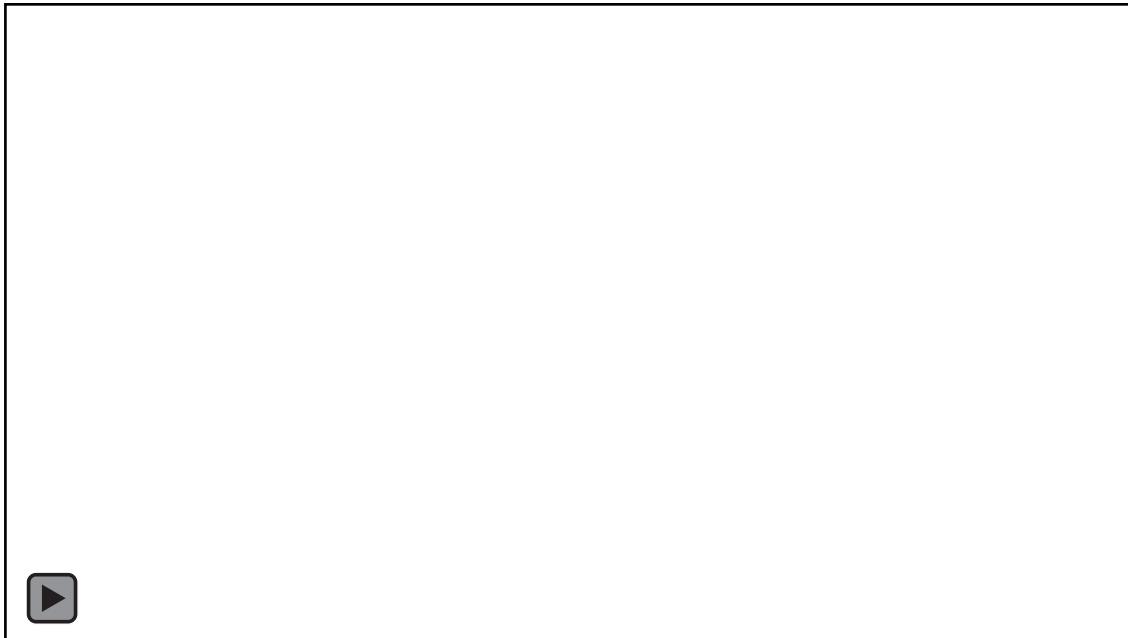
The non-smooth path can provide an initial guess for decision variables.



Smooth Navigation Path Generation

Cubic Spline (**Minimum Stretch Energy**) Path Generation

$$\min_{x_1, x_2 \dots, x_{N-1}} \text{Energy}(x_1, x_2 \dots, x_{N-1}) + \text{Potential}(x_1, x_2 \dots, x_{N-1})$$



Thanks for Listening!

*Ack.: The slide is prepared by Zhepei Wang and Fei Gao,
improved by Lingfeng Wang, Song Zhao, and Shuo Yang.*