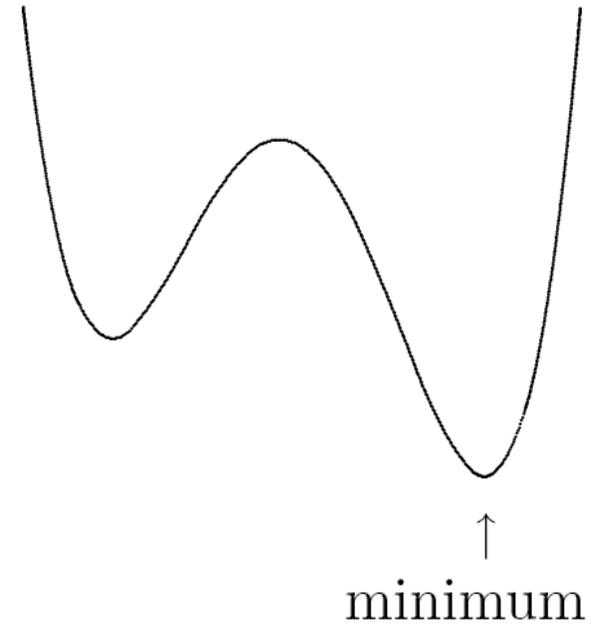


Optimization

<https://www.quantamagazine.org/three-hundred-years-later-a-tool-from-isaac-newton-gets-an-update-20250324/>

Outline:

- Optimization Problems
- One Dimensional Optimization
- Multi-Dimensional Optimization
- Nonlinear Least Squares



Optimization

- Given function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, and set $S \subseteq \mathbb{R}^n$, find $\mathbf{x}^* \in S$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in S$
- \mathbf{x}^* is called minimizer or minimum of f
- It suffices to consider only minimization because maximum of f is minimum of $-f$
- *Objective function*, f , is usually differentiable, and may be linear or non-linear
- Constraint set S is defined by system of equations and inequalities, which may be linear or nonlinear
- Points $\mathbf{x} \in S$ are called *feasible* points
- If $S = \mathbb{R}^n$, problem is *unconstrained*

Optimization Problems

- General continuous optimization problem:

$$\min f(\mathbf{x}) \text{ subject to } \mathbf{g}(\mathbf{x}) = 0 \text{ and } \mathbf{h}(\mathbf{x}) \leq 0$$

where $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $\mathbf{g} : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, $\mathbf{h} : \mathbb{R}^n \longrightarrow \mathbb{R}^p$

- *Linear programming*: f , \mathbf{g} , \mathbf{h} are all linear
- *Nonlinear programming*: at least one of f , \mathbf{g} , \mathbf{h} is nonlinear

Examples: Optimization Problems

- Minimize weight of structure subject to constraint on its strength, or maximize its strength subject to constraint on its weight
- Minimize cost of diet subject to nutritional constraints
- Minimize surface area of cylinder subject to constraint on its volume:

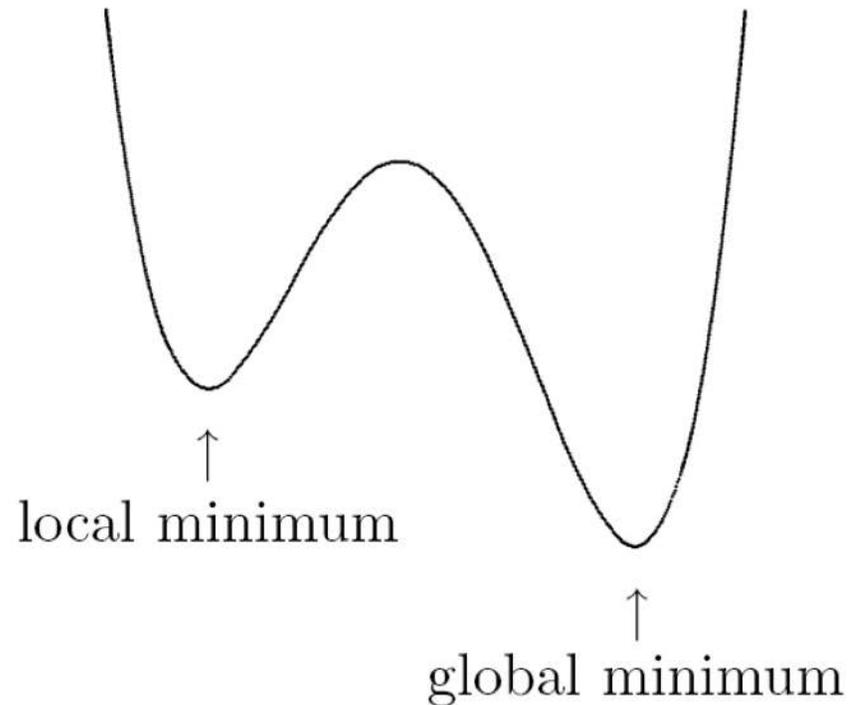
$$\min_{x_1, x_2} f(x_1, x_2) = 2\pi x_1(x_1 + x_2)$$

$$\text{subject to } g(x_1, x_2) = \pi x_1^2 x_2 - V = 0$$

where x_1 is the cylinder radius, x_2 is the height, and V is the required volume

Local vs Global Optimization

- $\mathbf{x}^* \in S$ is *global minimum* if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in S$
- $\mathbf{x}^* \in S$ is *local minimum* if $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all feasible \mathbf{x} in some neighborhood of \mathbf{x}^*



Global Optimization

- Finding or, even verifying, global minimum is difficult in general
- Most optimization methods are designed to find local minimum, which may or may not be global minimum
- If global minimum is desired, one can try several widely separated starting points and see if all produce same result

Existence of Minimum

- If f is continuous on *closed and bounded* set $S \subseteq \mathbb{R}^n$, then f has global minimum on S
- If S is not closed or is unbounded, then f may have no local or global minimum on S
- Continuous function f on unbounded set $S \subseteq \mathbb{R}^n$ is *coercive* if

$$\lim_{\|\mathbf{x}\| \rightarrow \infty} f(\mathbf{x}) = +\infty$$

i.e., $f(\mathbf{x})$ must be large when $\|\mathbf{x}\|$ is large

- If f is coercive on closed, unbounded set $S \subseteq \mathbb{R}^n$, then f has a global minimum on S

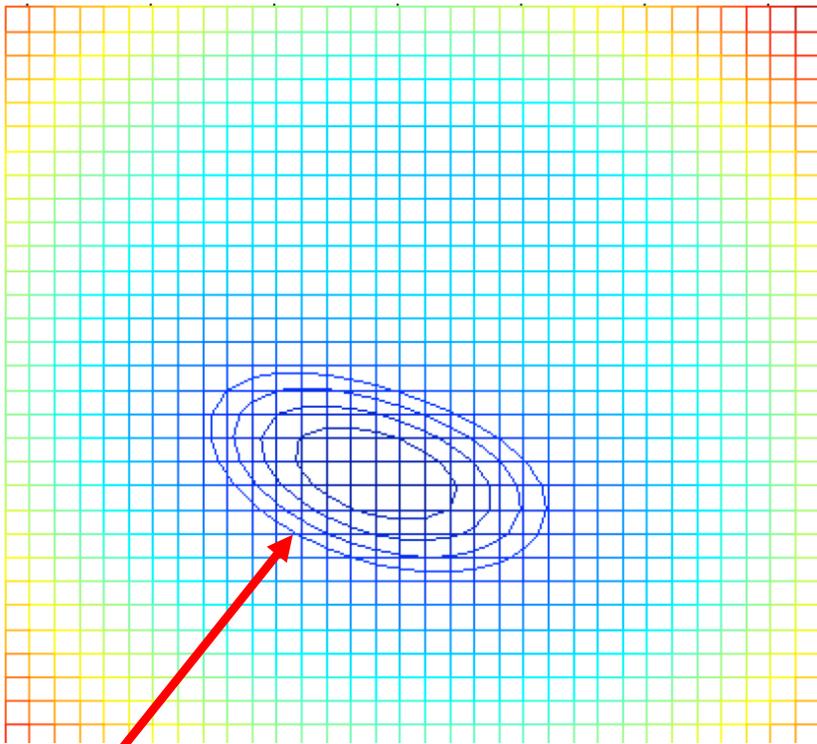
An Example of a Coercive Function

```
x=-1.6:.1:1.5; y=-1.6:.1:1.7;  
[X,Y]=ndgrid(x,y);
```

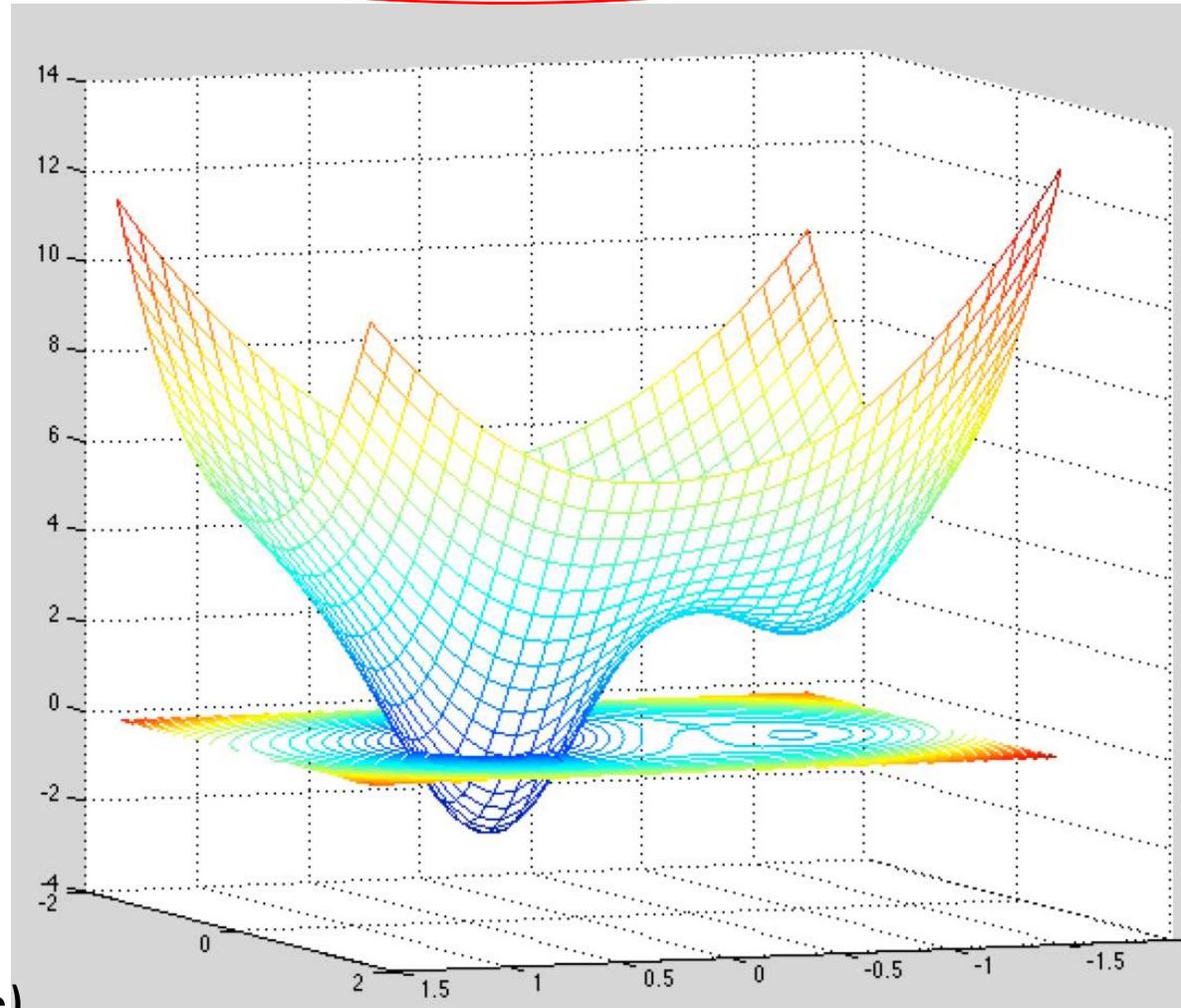
```
u = -3*exp(-X.*X-Y.*Y).*sin(3*X-Y) + 1.2*(X-Y).^2+(X+Y).^2;
```

Goes to ∞ as $\|\mathbf{x}\| \rightarrow \infty$.

```
hold off; mesh(X,Y,u)  
hold on ; contour(X,Y,u,30)
```



Level Sets (contours, isosurfaces)



[demo: coerce.m](#)

Gradient of $f(\mathbf{x})$

- $f(\mathbf{x})$ is a *scalar function* of an n -dimensional *vector input*.

$$f(\mathbf{x}) = f(x_1, x_2, \dots, x_n)$$

$$\mathbf{x} = x_1\mathbf{i} + x_2\mathbf{j} + x_3\mathbf{k} \quad (\text{say, for } n=3).$$

- Its *gradient* has n components,

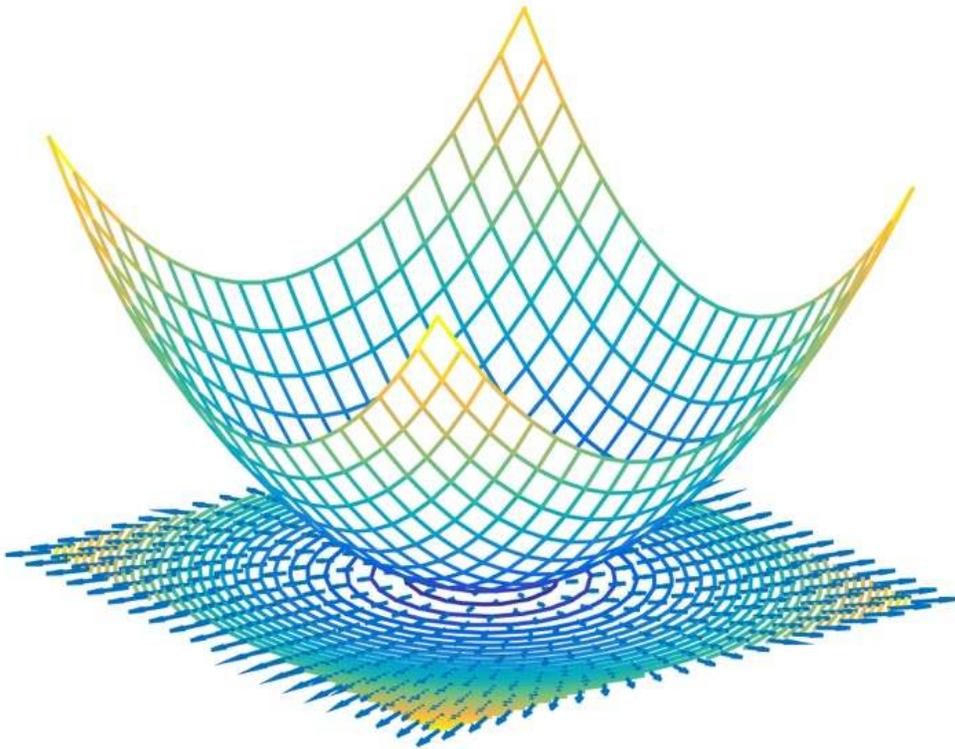
$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial x_1}\mathbf{i} + \frac{\partial f}{\partial x_2}\mathbf{j} + \frac{\partial f}{\partial x_3}\mathbf{k}.$$

- The *vector*, ∇f , is in the domain of $f(\mathbf{x})$ —for each direction, (\mathbf{i} , \mathbf{j} , \mathbf{k}), there is one component of ∇f .

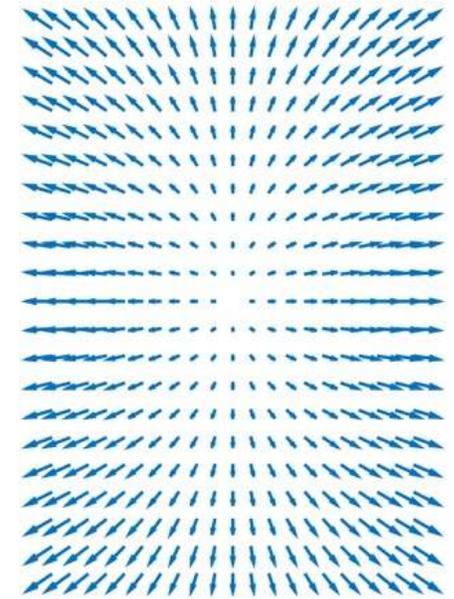
- It is linear in f : $\nabla(2f) = 2\nabla f$, so its units are

$$\frac{[f]}{[x]} = \frac{\text{units of } f}{\text{units of } x}$$

Iso-Contours and Gradients



*Level Sets
(contours,
isosurfaces)*



Gradient field

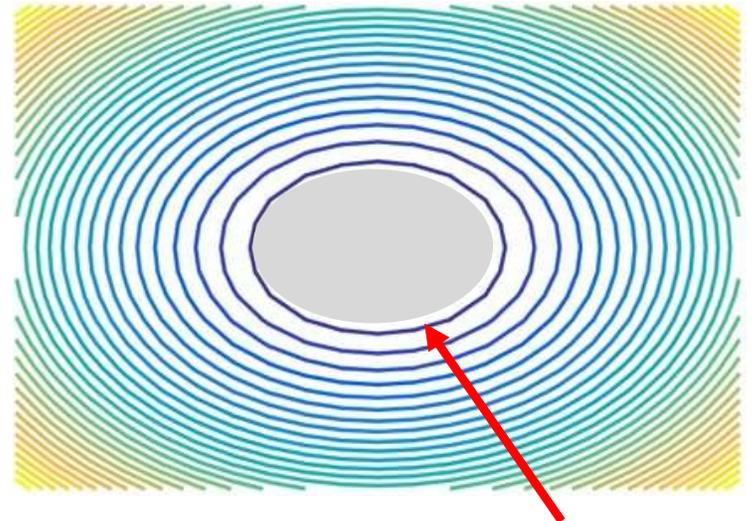
- ❑ *Gradient points **away** from minimum.*
- ❑ *-Gradient points **towards** the minimum.*
- ❑ *Gradient direction is the direction of steepest **ascent** at point (x,y)*

Level Sets

- *Level set* for function $f : S \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ is set of all points in S for which f has some given constant value

- For given $\gamma \in \mathbb{R}$, *sublevel set* is

$$L_\gamma = \{\mathbf{x} \in S : f(\mathbf{x}) \leq \gamma\}$$

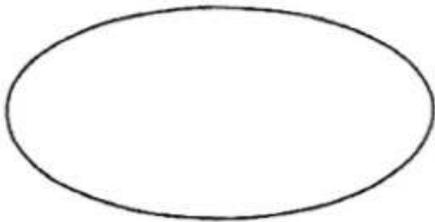


Level Set (isocontour)

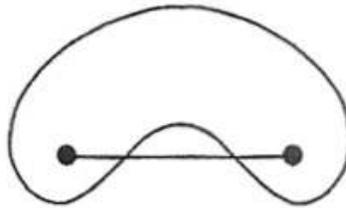
- If continuous function f on $S \subseteq \mathbb{R}^n$ has nonempty sublevel set that is closed and bounded, then f has global minimum on S (*does not have to be coercive*)
- If S is unbounded, then f is coercive on S if, and only if, *all* of its sublevel sets are bounded (*No open contours...*)

Uniqueness of Minimum

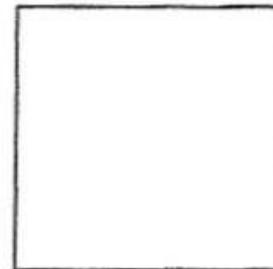
- Set $S \subseteq \mathbb{R}^n$ is *convex* if it contains a line segment between any two of its points



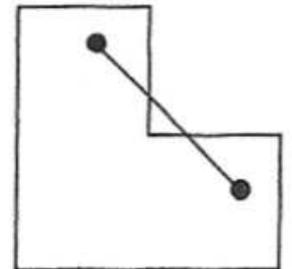
convex



nonconvex



convex



nonconvex

Uniqueness of Minimum

- Function $f : S \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ is *convex* on convex set S if its graph along any line segment in S lies *on or below* chord connecting function values at endpoints of segment



nonconvex



convex



strictly convex

Uniqueness of Minimum

- Set $S \subseteq \mathbb{R}^n$ is *convex* if it contains a line segment between any two of its points
- Function $f : S \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ is *convex* on convex set S if its graph along any line segment in S lies *on or below* chord connecting function values at endpoints of segment
- Any local minimum of convex function f on convex set $S \subseteq \mathbb{R}^n$ is global minimum of f on S
- Any local minimum of *strictly* convex function f on convex set $S \subseteq \mathbb{R}^n$ is *unique* global minimum of f on S

First-Order Optimality Condition

- For function of one variable, one can find extremum by differentiating function and setting derivative to zero
- Generalization to function of n variables is to find *critical point*, i.e., solution of nonlinear system

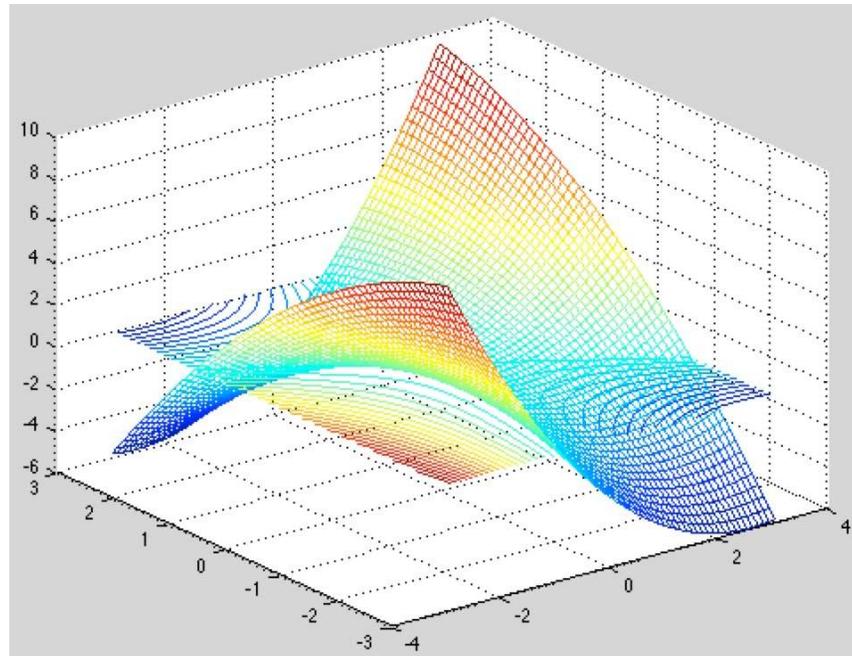
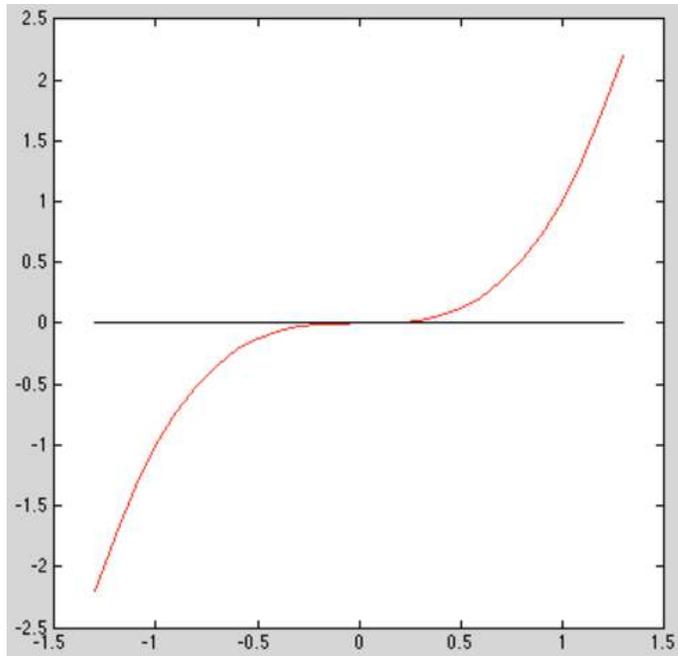
$$\nabla f(\mathbf{x}) = 0$$

where $\nabla f(\mathbf{x})$ is gradient vector of f , whose i th component is $\partial f(\mathbf{x})/\partial x_i$

- For *continuously differentiable* $f : S \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, any interior point \mathbf{x}^* of S at which f has a local minimum must be a critical point of f
- But not all critical points are minima: they can also be maxima or saddle points

First-Order Optimality Condition

- Not all critical points are minima: they can also be maxima or saddle points
- Saddle points in higher dimensions are more common than zero-slope inflection points in 1D



Note the open contours....

First-Order Optimality Condition

- For function of one variable, one can find extremum by differentiating function and setting derivative to zero
- Generalization to function of n variables is to find *critical point*, i.e., solution of nonlinear system

$$\nabla f(\mathbf{x}) = 0$$

where $\nabla f(\mathbf{x})$ is gradient vector of f , whose i th component is $\partial f(\mathbf{x})/\partial x_i$

- We use the solution techniques of Chapter 5 to solve this nonlinear system of n equations $(\partial f/\partial x_i)$ in n unknowns, x_j .

Second-Order Optimality Condition

- For twice continuously differentiable $f : S \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, we can distinguish among critical points by considering the *Hessian matrix*, $\mathbf{H}_f(\mathbf{x})$ defined by

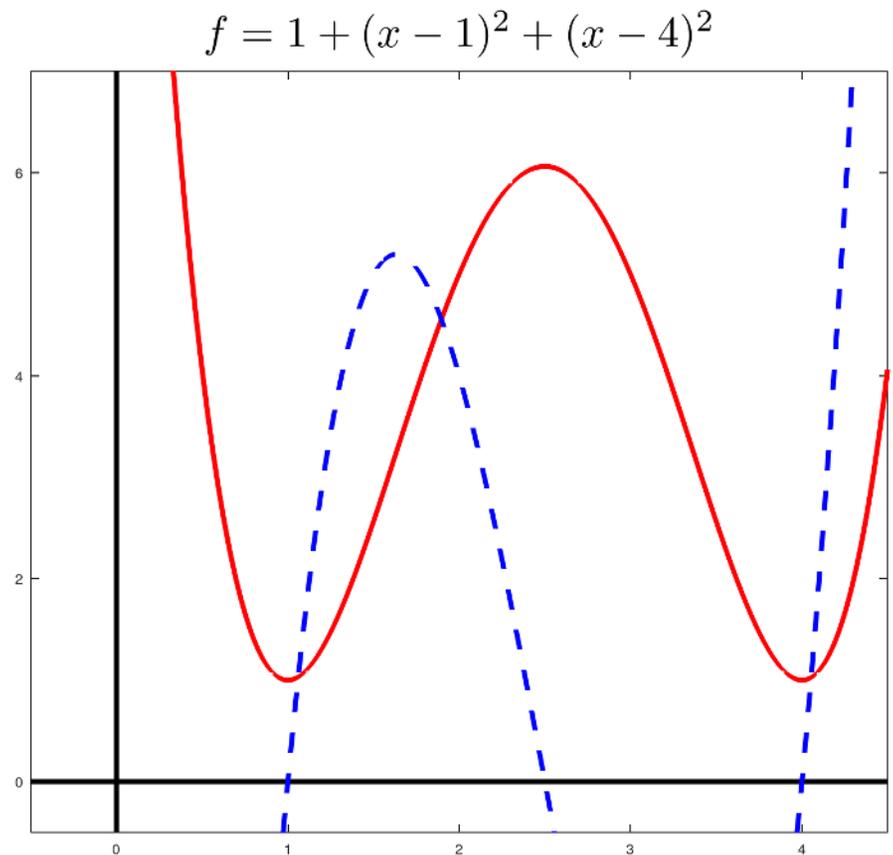
$$[\mathbf{H}_f(\mathbf{x})]_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} = \frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_i}$$

which is symmetric

- At critical point \mathbf{x}^* , if $\mathbf{H}_f(\mathbf{x}^*)$ is
 - positive definite, then \mathbf{x}^* is minimum of f
 - negative definite, then \mathbf{x}^* is maximum of f
 - indefinite, then \mathbf{x}^* is a saddle point of f
 - singular, then various pathological situations are possible

Example: Classifying Critical Points

- $f = 1 + (x - 1)^2 + (x - 4)^2$
- $f' = 4(x - 1)(x - 2.5)(x - 4)$
- $f'' = 4[(x - 2.5)(x - 4) + (x - 1)(x - 4) + (x - 1)(x - 2.5)]$
- Critical points ($f'(x) = 0$), are $x = 1, 2.5$
- Hessian is a 1×1 symmetric matrix
- At $x = 1$, $f'' = 18$ (local minimum)
- At $x = 4$, $f'' = 18$ (local minimum)
- At $x = 2.5$, $f'' = -9$ (local maximum)



Example: Classifying Critical Points

- Example 6.5 from the text:

$$f(\mathbf{x}) = 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6.$$

- Gradient:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 6x_1^2 + 6x_1 + 12x_2 \\ 12x_1 + 6x_2 - 6 \end{bmatrix}.$$

- Hessian is symmetric:

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} \end{bmatrix} = \begin{bmatrix} 12x_1 + 6 & 12 \\ 12 & 6 \end{bmatrix}.$$

Example: Classifying Critical Points

- *Critical points*, solutions to $\nabla f(\mathbf{x}) = 0$ are

$$\mathbf{x}^* = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \text{and} \quad \mathbf{x}^* = \begin{bmatrix} 2 \\ -3 \end{bmatrix}$$

- At first critical point, $\mathbf{x}^* = [1, -1]^T$,

$$H_f(1, -1) = \begin{bmatrix} 18 & 12 \\ 12 & 6 \end{bmatrix},$$

is *indefinite* (both positive and negative eigenvalues), so $[1, -1]^T$ is a saddle point.

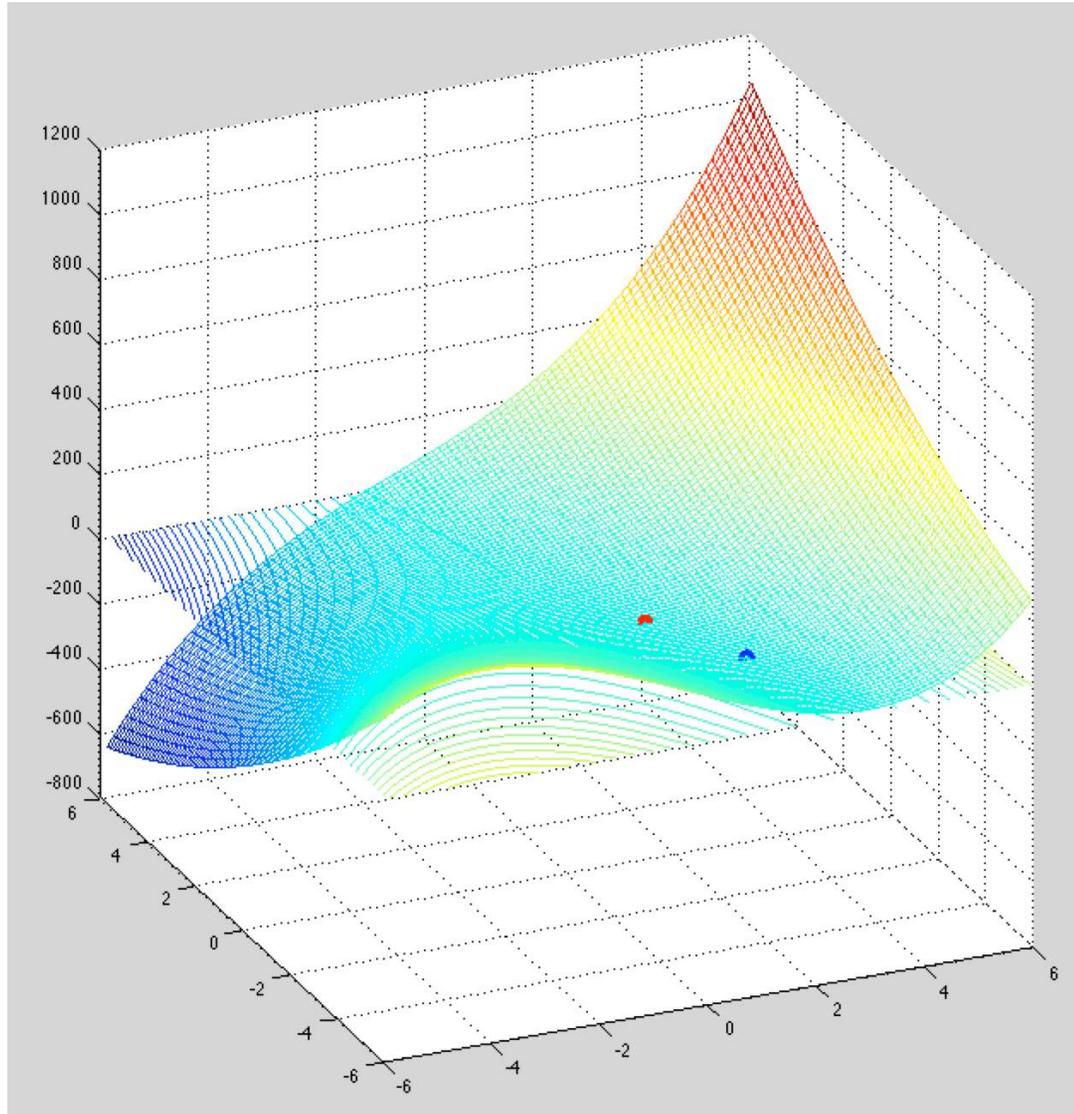
- At second critical point, $\mathbf{x}^* = [2, -3]^T$,

$$H_f(2, -3) = \begin{bmatrix} 30 & 12 \\ 12 & 6 \end{bmatrix},$$

does have positive eigenvalues, so $[2, -3]^T$ is a local minimum.

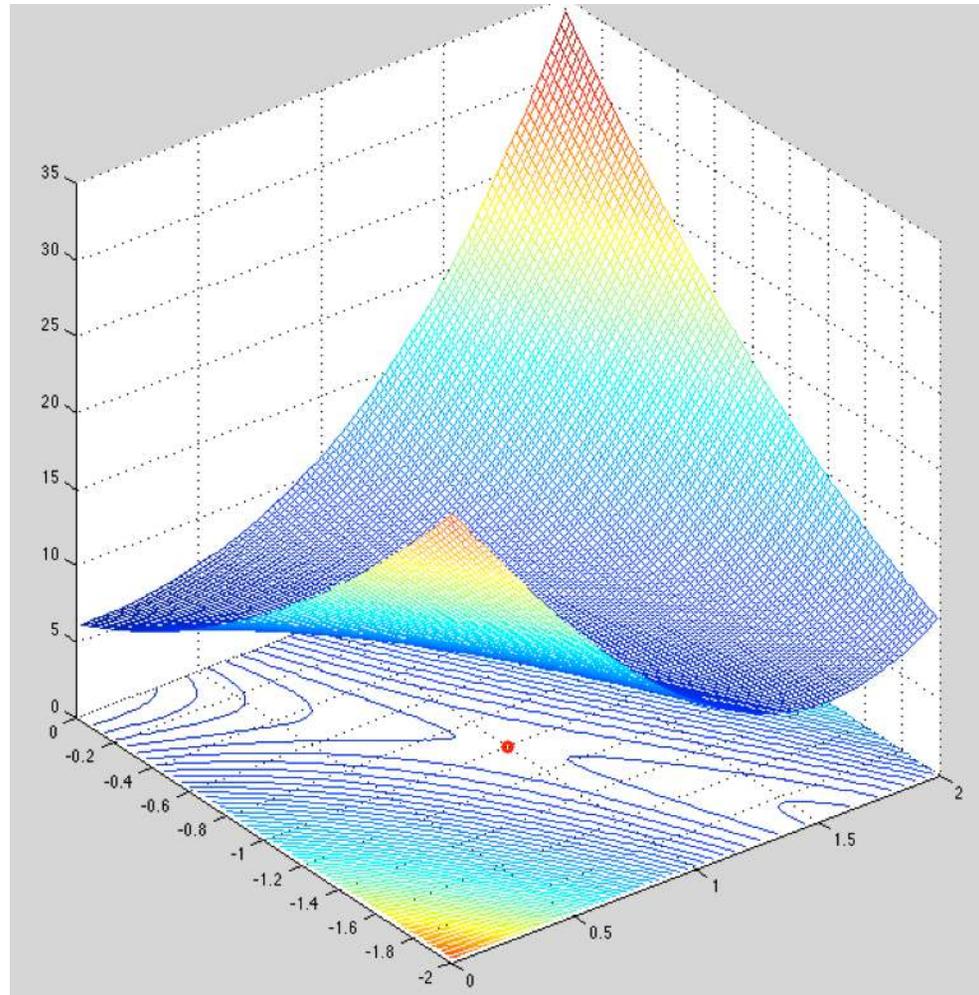
Example 6.5 from text

$$f(\mathbf{x}) = 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6.$$



Example 6.5 from text

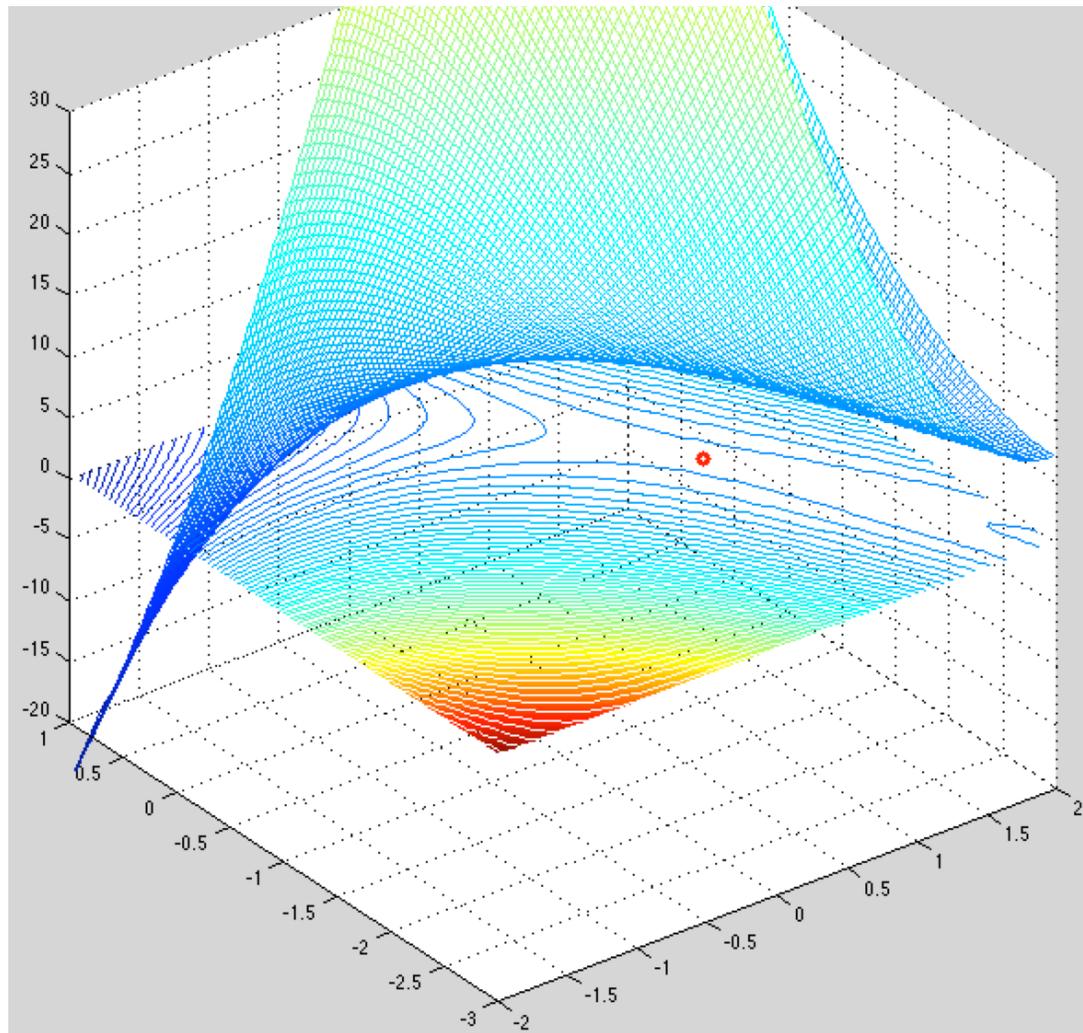
$$f(\mathbf{x}) = 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6.$$



- First critical point, $x^* = [1, -1]^T$, is a saddle point. Hessian not SPD.

Example 6.5 from text

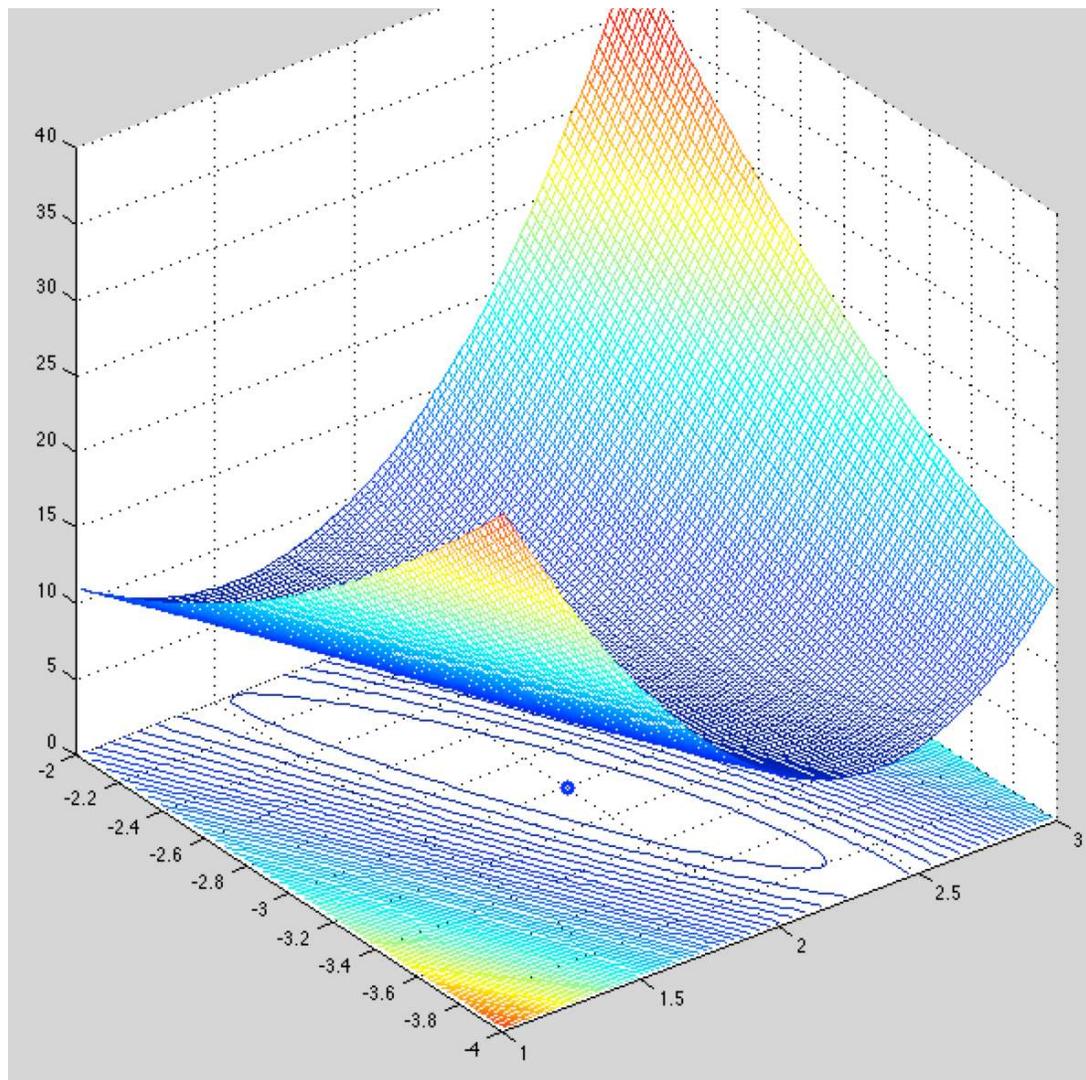
$$f(\mathbf{x}) = 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6.$$



- First critical point, $x^* = [1, -1]^T$, is a saddle point. Hessian not SPD.

Example 6.5 from text

$$f(\mathbf{x}) = 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6.$$



- Second critical point, $x^* = [2, -3]^T$, is a local minimum. Hessian is SPD.

Second-Order Optimality Condition

- For twice continuously differentiable $f : S \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$, we can distinguish among critical points by considering the *Hessian matrix*, $\mathbf{H}_f(\mathbf{x})$ defined by

$$[\mathbf{H}_f(\mathbf{x})]_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$$

which is symmetric

- At critical point \mathbf{x}^* , if $\mathbf{H}_f(\mathbf{x}^*)$ is
 - positive definite, then \mathbf{x}^* is minimum of f
 - negative definite, then \mathbf{x}^* is maximum of f
 - indefinite, then \mathbf{x}^* is a saddle point of f
 - singular, then various pathological situations are possible
-

Example of Singular Hessian in 1D

- Consider,

$$f(x) = (x - 1)^4$$

$$\nabla f = \frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial x} = 4(x - 1)^3 = 0 \text{ when } x = x^* = 1.$$

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x^2} = 12(x - 1)^2 = 0.$$

Here, the Hessian is singular at $x = x^*$ and x^* is a minimizer of f .

- However, if

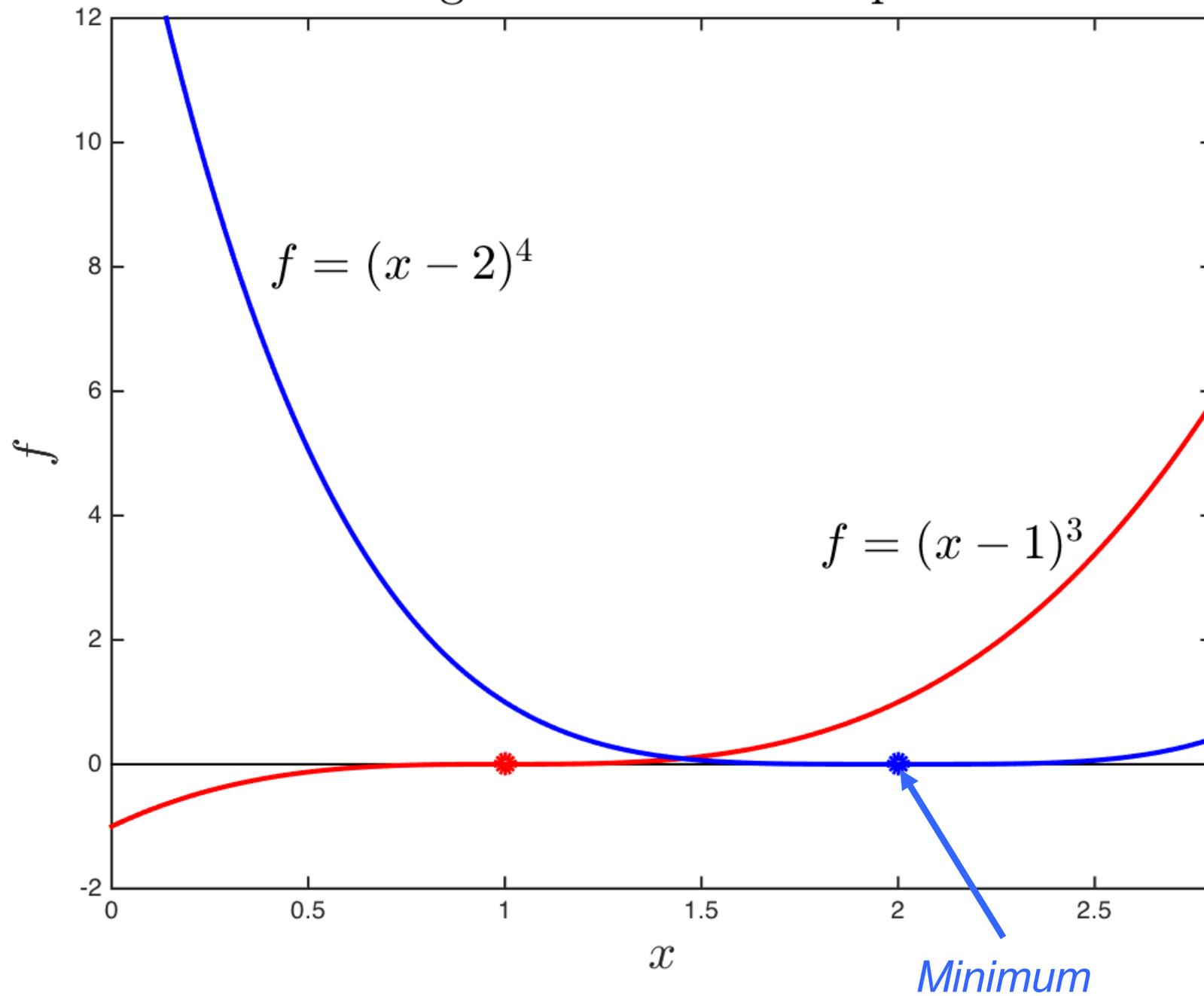
$$f(x) = (x - 1)^3$$

$$\nabla f = \frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial x} = 3(x - 1)^2 = 0 \text{ when } x = x^* = 1.$$

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x^2} = 6(x - 1) = 0.$$

Here, the Hessian is singular at $x = x^*$ and x^* is a *not* a minimizer of f .

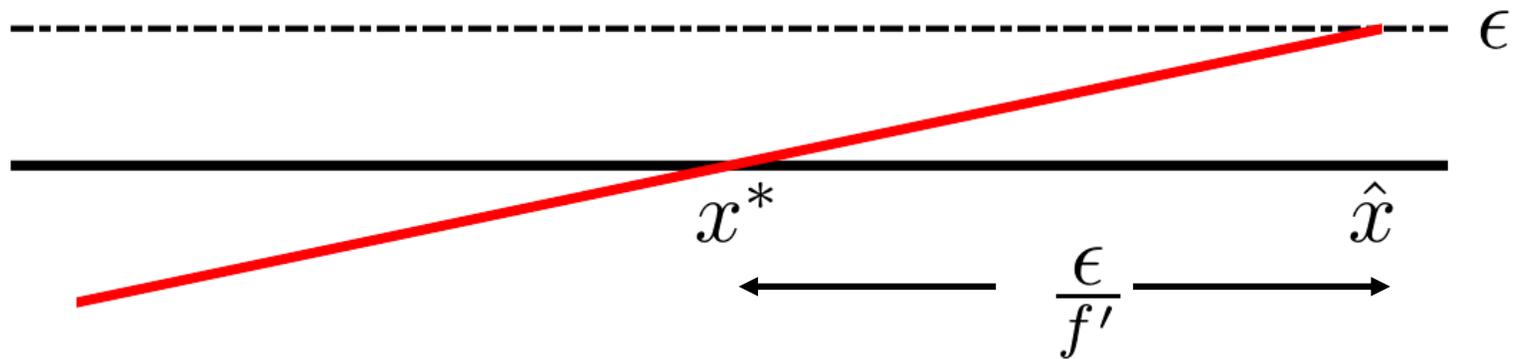
Singular Hessian Examples



***Sensitivity and Conditioning of
Optimization Problems***

Sensitivity and Conditioning

- Function minimization and equation solving are closely related problems, but their sensitivities differ
- In one dimension, absolute condition number of root x^* of equation $f(x) = 0$ is $1/|f'(x^*)|$, so if $|f(\hat{x})| \leq \epsilon$, then $|\hat{x} - x^*|$ may be as large as $\epsilon/|f'(x^*)|$



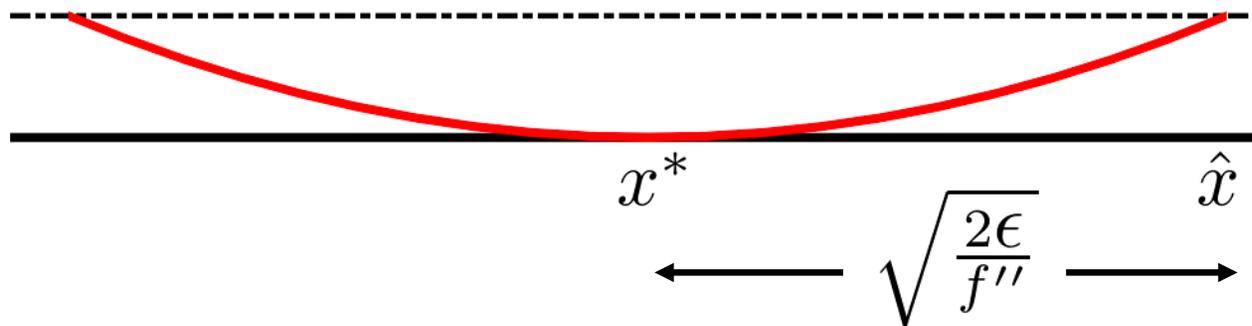
- The smaller the slope, the larger the uncertainty.

Sensitivity and Conditioning

- For minimizing f , Taylor series expansion about x^*

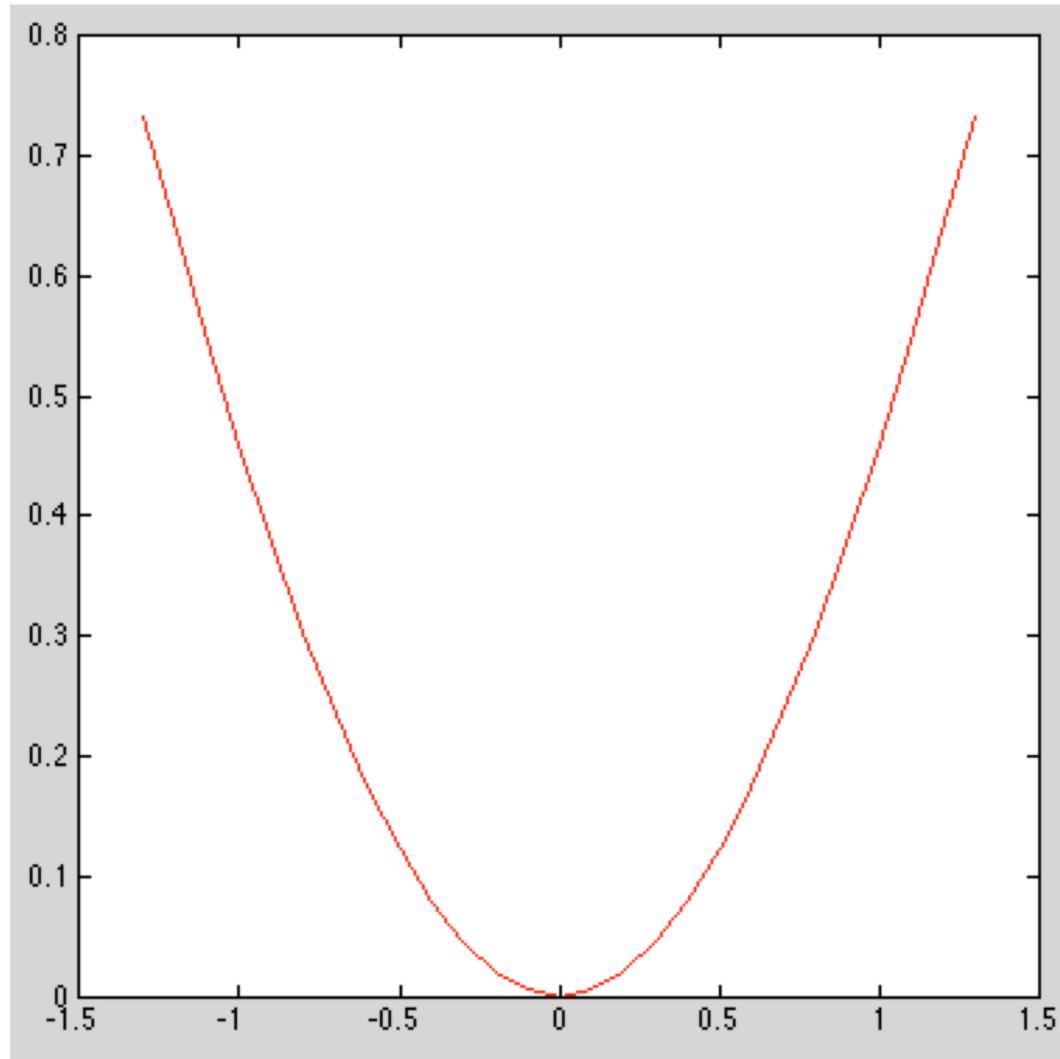
$$\begin{aligned}f(\hat{x}) &= f(x^* + h) \\&= f(x^*) + f'(x^*)h + \frac{1}{2}f''(x^*)h^2 + O(h^3) \\&\sim f(x^*) + \frac{1}{2}f''(x^*)h^2\end{aligned}$$

shows that if $|f(\hat{x}) - f(x^*)| \leq \epsilon$, then uncertainty in minimum, $|\hat{x} - x^*|$, may be as large as $\sqrt{2\epsilon/f''(x^*)} \gg O(\epsilon)$

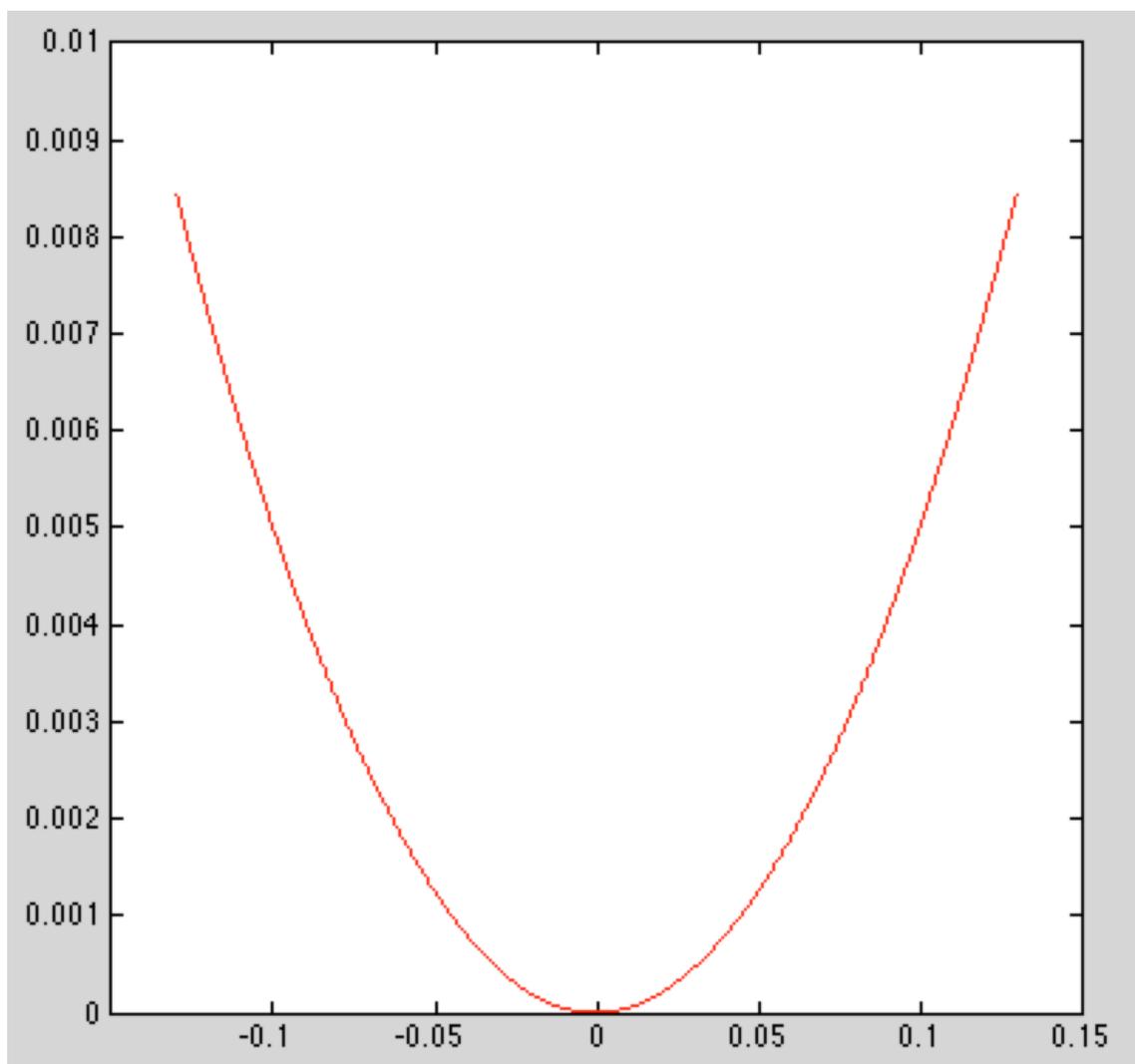


- Thus, based on function values alone, minima can be computed to only about half precision

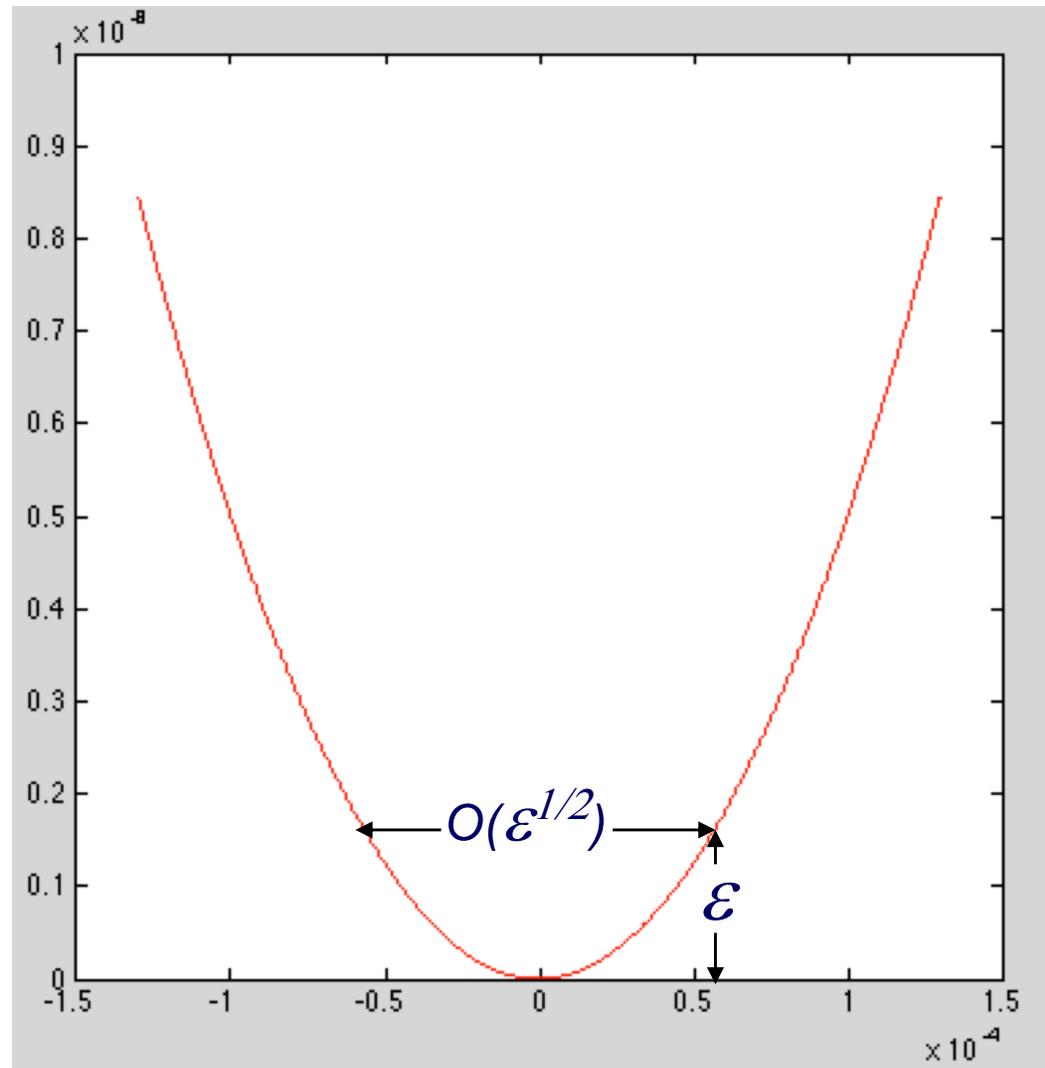
Consider $f=1-\cos(x)$



Consider $f=1-\cos(x)$



Consider $f=1-\cos(x)$



- ❑ As you zoom in, $f(x) \rightarrow 0$ quadratically, but $x \rightarrow x^*$ only linearly
- ❑ So, if $|f(x) - f(x^*)| \sim \epsilon$, then $\| \underline{x} - \underline{x}^* \| = O(\epsilon^{1/2}) \gg O(\epsilon)$

Sensitivity of Minimization

Terminate search when

$$|f(x^* + \Delta x) - f(x^*)| \leq \epsilon.$$

Taylor series:

$$f(x^* + \Delta x) = f(x^*) + \Delta x f'(x^*) + \frac{\Delta x^2}{2} f''(x^*) + O(\Delta x^3)$$

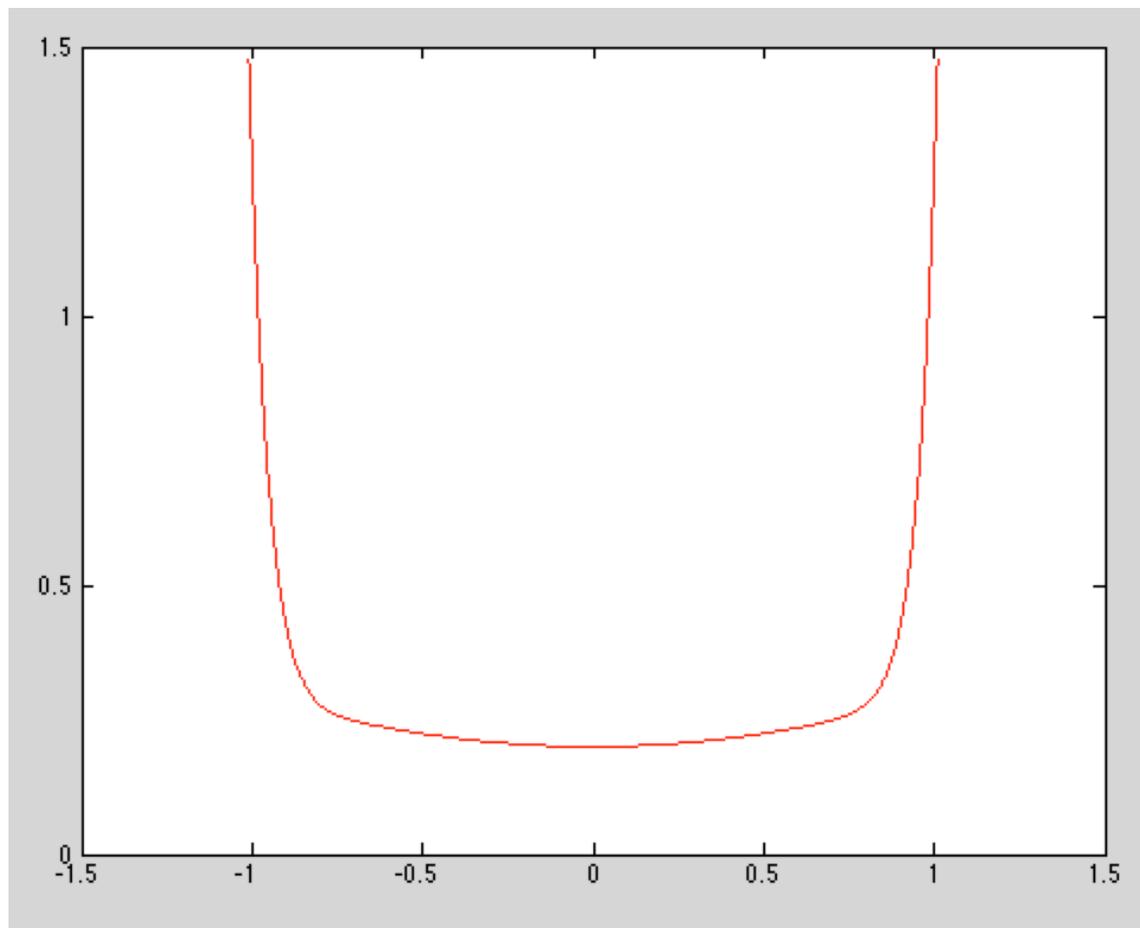
$$\frac{\Delta x^2}{2} \approx \frac{f(x^* + \Delta x) - f(x^*)}{f''(x^*)}$$

$$|\Delta x| \approx \sqrt{\frac{2\epsilon}{|f''(x^*)|}}$$

- So, if $\epsilon \approx \epsilon_M$, can expect accuracy to approximately $\sqrt{\epsilon_M}$.
- Question: Is a small f'' good? Or bad?

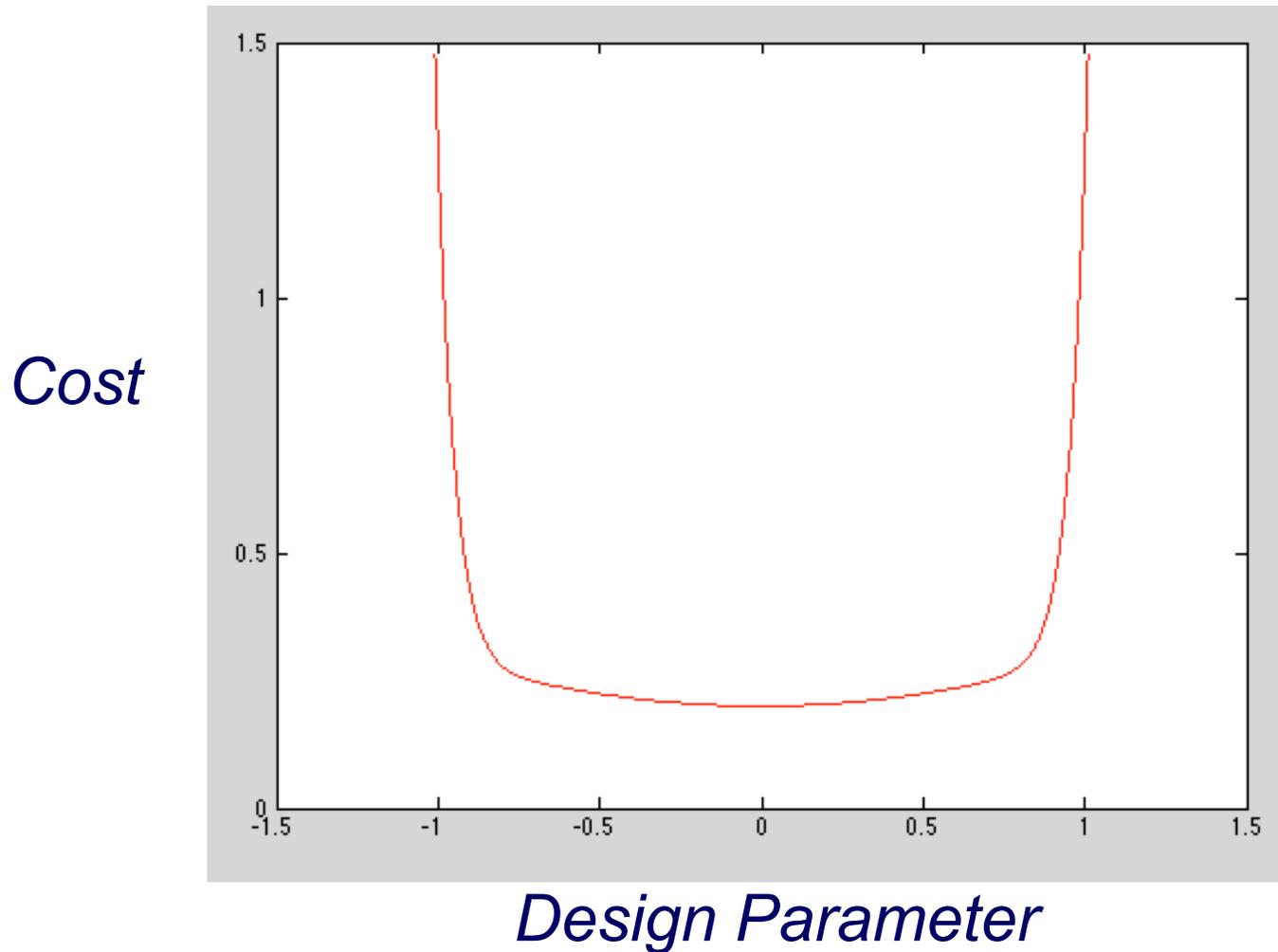
Example: Minimize Cost over Some Design Parameter, x

Cost



Design Parameter

Example: Minimize Cost over Some Design Parameter, x



- ❑ While having $f''(x^*)$ makes it difficult to find the **optimal** x^* , it in fact is a happy circumstance because it gives you liberty to add additional constraints at no cost.
- ❑ So, often, having a broad minimum is **good**.

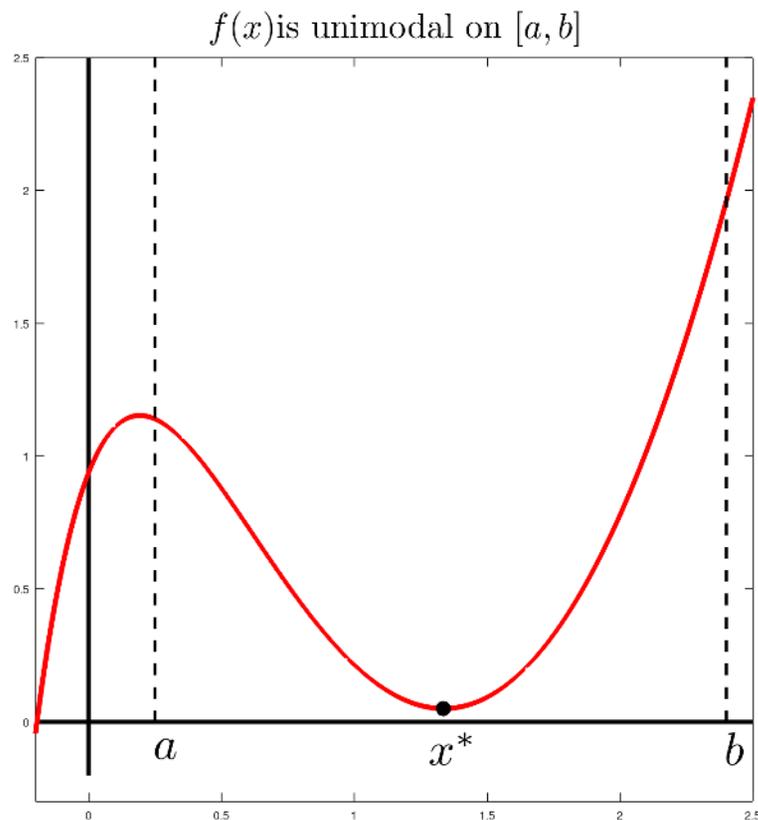
Methods for One-Dimensional Problems

- Demonstrate
 - Basic techniques
 - Bracketing
 - Convergence rates

- Useful for ***line search*** in multi-dimensional problems

Unimodality

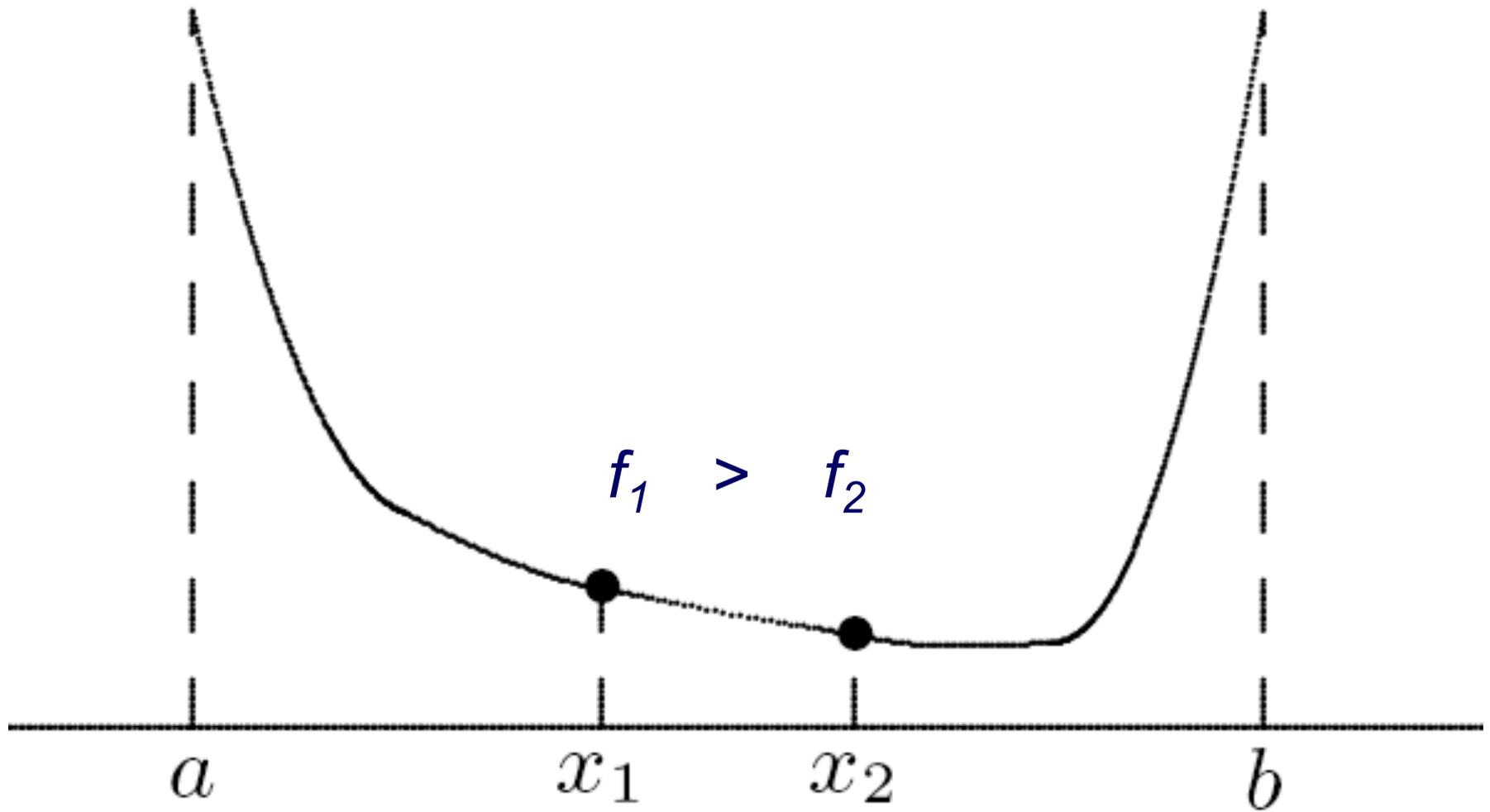
- For minimizing function of one variable, we need “bracket” for solution analogous to sign change for nonlinear equation
- Real-valued function f is *unimodal* on interval $[a, b]$ if there is a unique $x^* \in [a, b]$ such that $f(x^*)$ is minimum of f on $[a, b]$, and f is strictly decreasing for $x \leq x^*$, strictly increasing for $x^* \leq x$



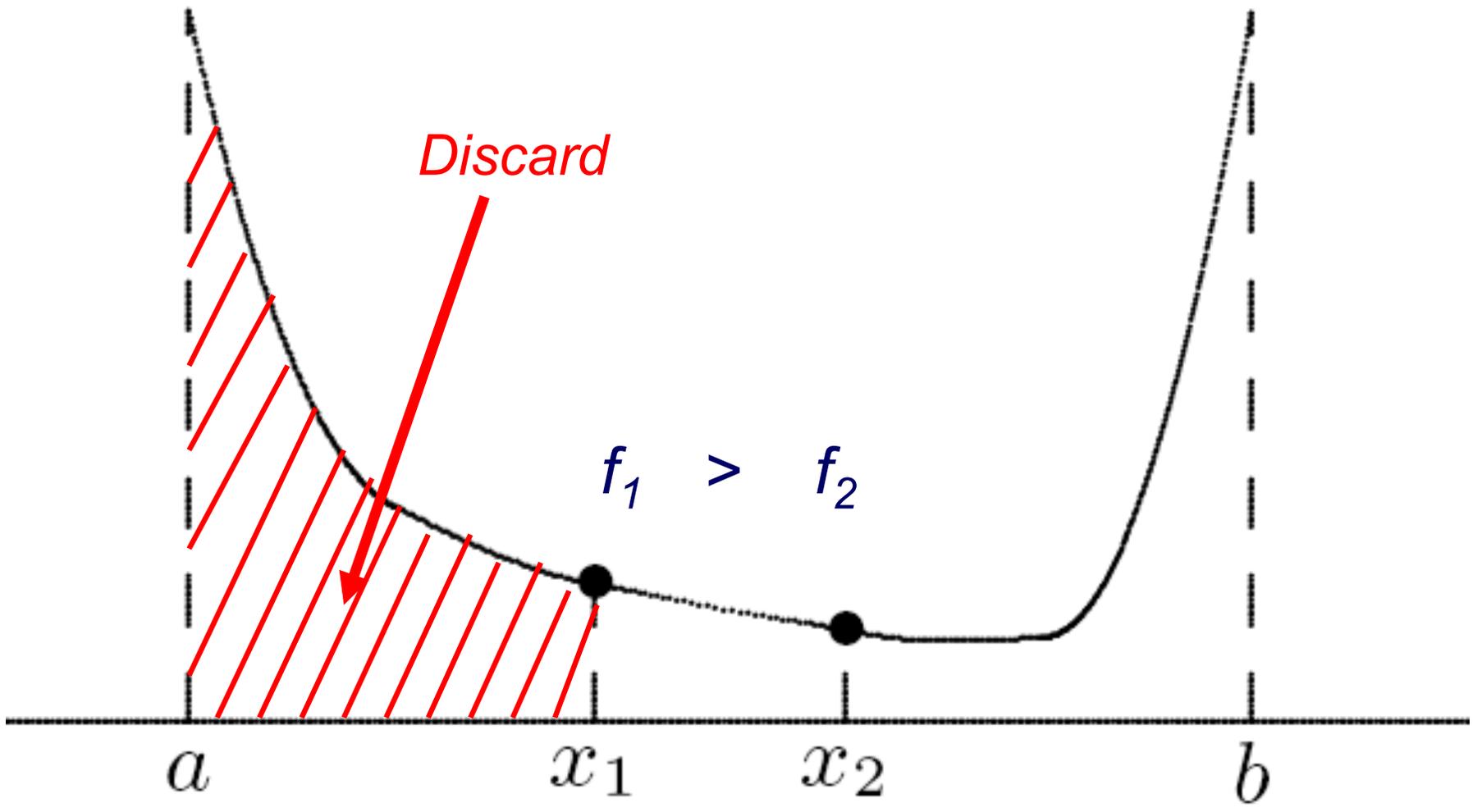
Unimodality

- For minimizing function of one variable, we need “bracket” for solution analogous to sign change for nonlinear equation
- Real-valued function f is *unimodal* on interval $[a, b]$ if there is a unique $x^* \in [a, b]$ such that $f(x^*)$ is minimum of f on $[a, b]$, and f is strictly decreasing for $x \leq x^*$, strictly increasing for $x^* \leq x$
- Unimodality enables discarding portions of interval based on sample function values, analogous to interval bisection for root finding

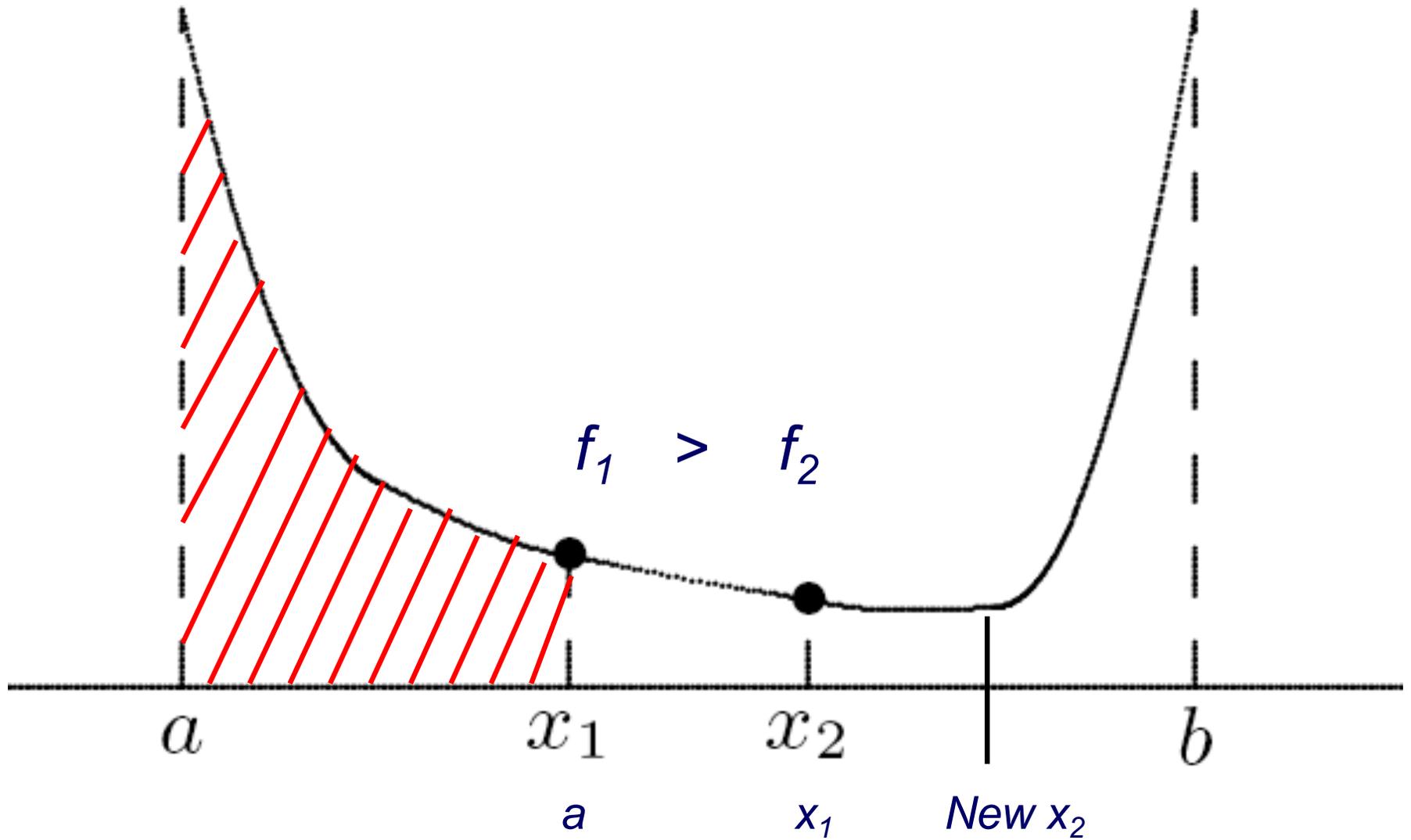
Golden Section Search



Golden Section Search



Golden Section Search



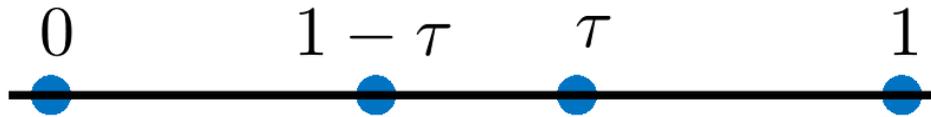
Golden Section Search

- Suppose f is unimodal on $[a, b]$ and let x_1 and x_2 be two points within $[a, b]$, with $x_1 < x_2$
- Evaluating and comparing $f(x_1)$ and $f(x_2)$, we can discard either $[a, x_1)$ or $(x_2, b]$, with minimum known to lie in remaining subinterval
- To repeat process, need to compute only one new function evaluation
- To reduce length of interval by fixed fraction at each iteration, each new pair of points must have same relationship with respect to new interval that previous pair had with respect to previous interval

Golden Section Search, continued

- To accomplish this, choose relative positions of two points as τ and $1 - \tau$, where $\tau^2 = 1 - \tau$, so $\tau = (\sqrt{5} - 1)/2 \approx 0.618$ and $1 - \tau \approx 0.382$
- Whichever subinterval is retained, its length will be τ relative to the previous interval, and interior point will be at position either τ or $1 - \tau$ relative to new interval
- To continue iteration, we need to compute only one new function value at complementary point
- This choice of sample points is called *golden section search*
- Golden section search is safe but convergence rate is only linear, with constant $C \approx 0.618$
- Also, golden section nominally requires unimodality, but will in fact find a local minimum in most cases

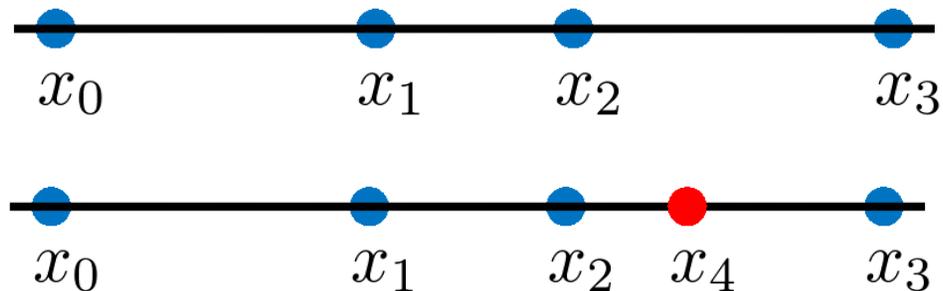
Golden Section Search, continued



- On $[0,1]$, choose τ such that ratio

$$\frac{1 - \tau}{1} = \frac{\tau - (1 - \tau)}{\tau}$$

- Yields $\tau = \frac{\sqrt{5} - 1}{2} \approx .618$
- If the initial points are x_0, \dots, x_3 , this choice allows us to add a point x_4 such that sample points on $[x_1 : x_3]$ have the same *relative* spacing as the original points



Golden Section Search, continued

Algorithm 6.1 Golden Section Search

$$\tau = (\sqrt{5} - 1)/2$$

$$x_1 = a + (1 - \tau)(b - a)$$

$$f_1 = f(x_1)$$

$$x_2 = a + \tau(b - a)$$

$$f_2 = f(x_2)$$

while $((b - a) > tol)$ **do**

if $(f_1 > f_2)$ **then**

$$a = x_1$$

$$x_1 = x_2$$

$$f_1 = f_2$$

$$x_2 = a + \tau(b - a)$$

$$f_2 = f(x_2)$$

else

$$b = x_2$$

$$x_2 = x_1$$

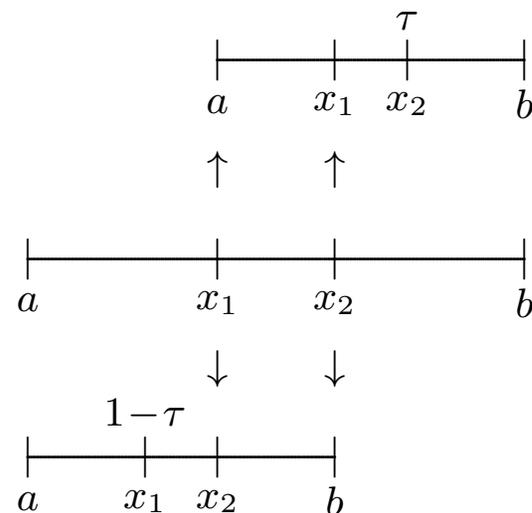
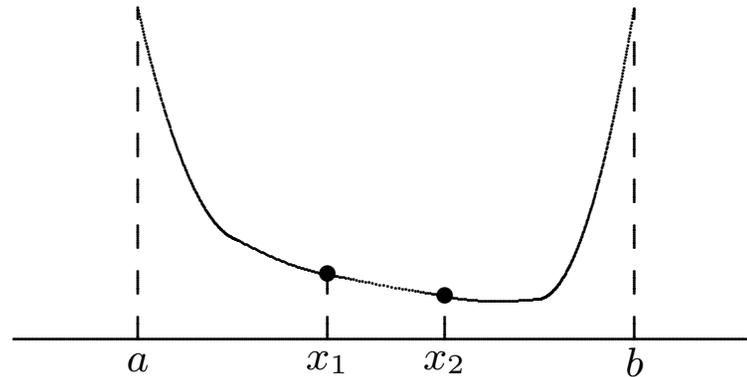
$$f_2 = f_1$$

$$x_1 = a + (1 - \tau)(b - a)$$

$$f_1 = f(x_1)$$

end

end



Example: Golden Section Search

- Use golden section search to minimize $f(x) = 0.5 - xe^{-x^2}$

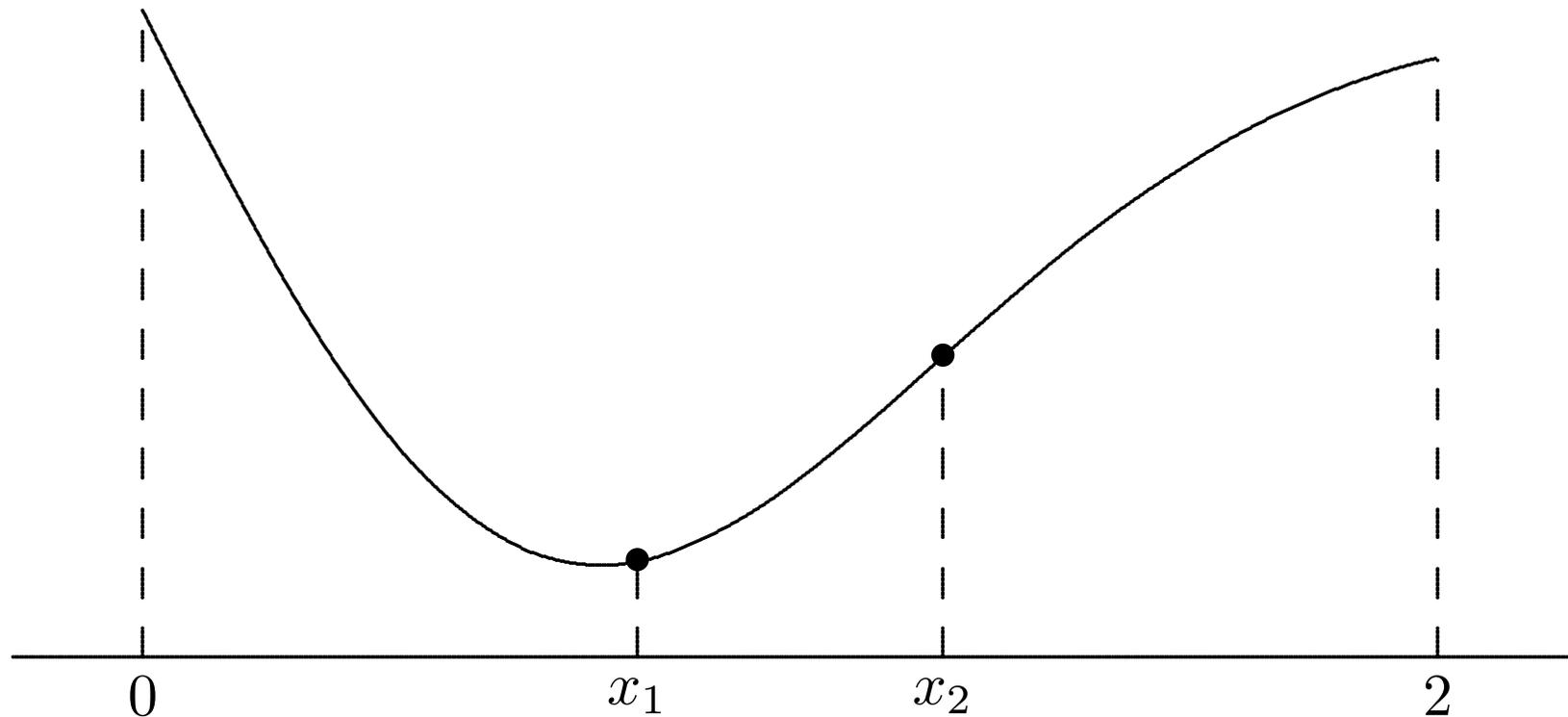


Figure 6.6: First iteration of golden section search for example problem.

[demo_golden.m](#)

Example: Golden Section Search

- Use golden section search to minimize $f(x) = 0.5 - xe^{-x^2}$

k	x_1	f_1	x_2	f_2
1	0.472	0.122	0.764	0.074
2	0.764	0.074	0.944	0.113
3	0.652	0.074	0.764	0.074
4	0.584	0.085	0.652	0.074
5	0.652	0.074	0.695	0.071
6	0.695	0.071	0.721	0.071
7	0.679	0.072	0.695	0.071
8	0.695	0.071	0.705	0.071
9	0.705	0.071	0.711	0.071

demo_golden.m

Successive Parabolic Interpolation

- Fit quadratic polynomial to three function values
- Take minimum of quadratic function to be new approximation to minimum of function

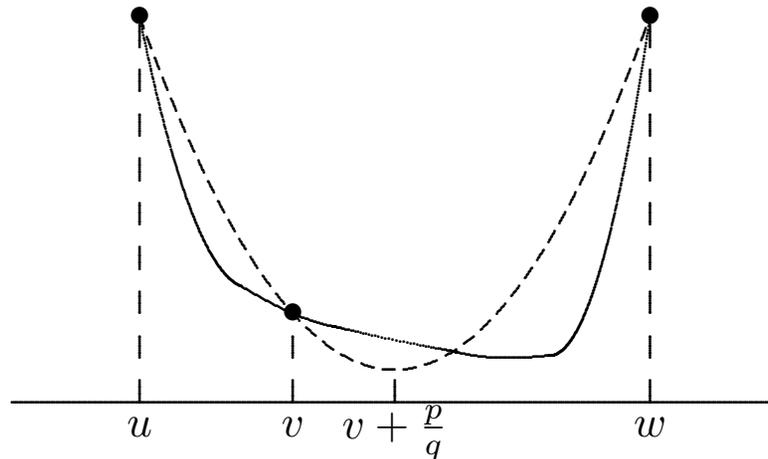


Figure 6.7: Successive parabolic iteration for minimizing a function.

- New point replaces oldest of three previous points and process is repeated until convergence
- Convergence rate of successive parabolic interpolation is superlinear with $r \approx 1.324$

Example: Successive Parabolic Interpolation

- Use successive parabolic interpolation to minimize $f(x) = 0.5 - xe^{-x^2}$

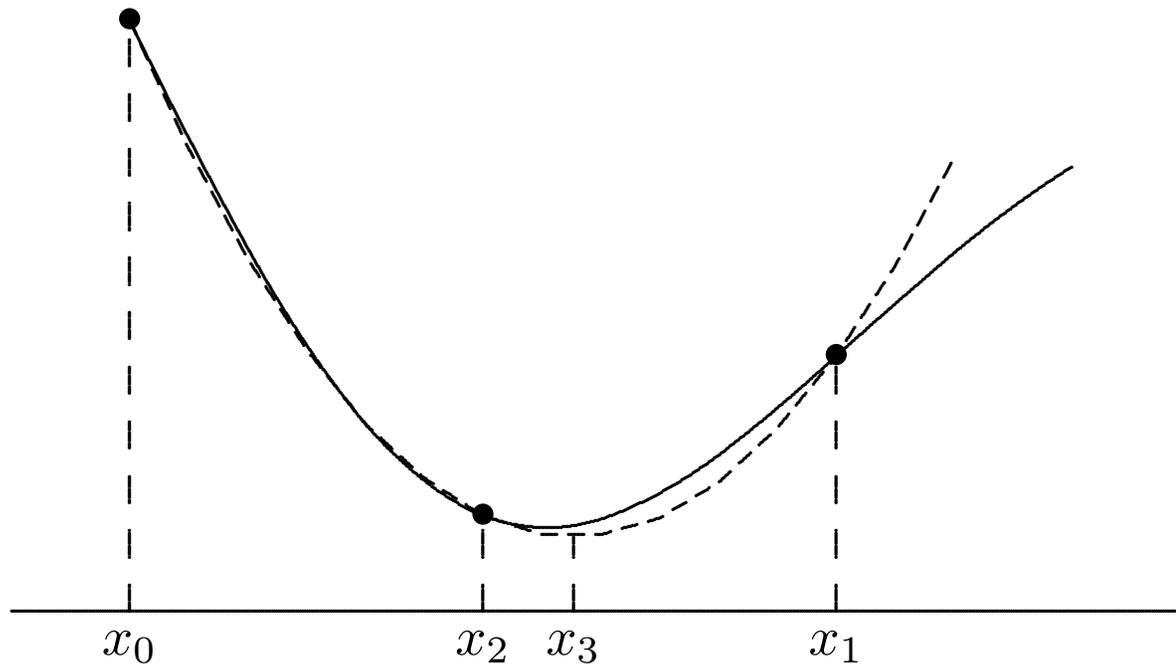


Figure 6.8: First iteration of successive parabolic iteration for example problem.

Matlab: Successive Parabolic Interpolation – **Replace Oldest**

```
function x=parab(a,b,c);  
  
fa=f(a); fb=f(b); fc=f(c); x=1; fx=f(x);  
  
for k=1:15;  
  
    num = ((fb-fc)*(b-a)^2-(fb-fa)*(b-c)^2);    % m'  
    den = 2*((fb-fc)*(b-a)-(fb-fa)*(b-c));      % m"  
    x=b-num/den; fx = f(x);  
  
    c=b; fc=fb; b=a; fb=fa; a=x; fa=fx; % Push c off  
  
end;
```

demo_para.m

Matlab: Successive Parabolic Interpolation – *Replace Oldest*

```
function x=parab(a,b,c);

xx=a:.001:c; yy=f(xx);
hold off; plot(xx,yy,'r-'); hold on;

format compact; format longe;

fa=f(a); fb=f(b); fc=f(c); x=1; fx=f(x);

for k=1:15;
    xo=x; fo=fx;

    num = ((fb-fc)*(b-a)^2-(fb-fa)*(b-c)^2); % m'
    den = 2*((fb-fc)*(b-a)-(fb-fa)*(b-c)); % m''
    x=b-num/den; fx = f(x);

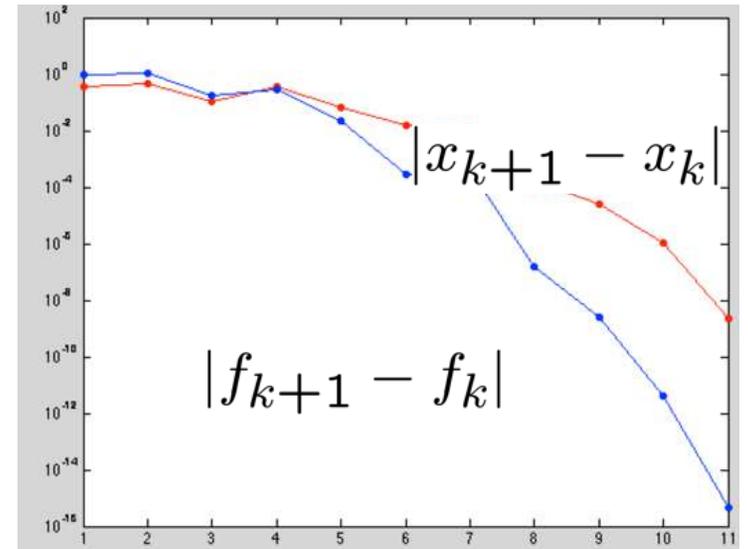
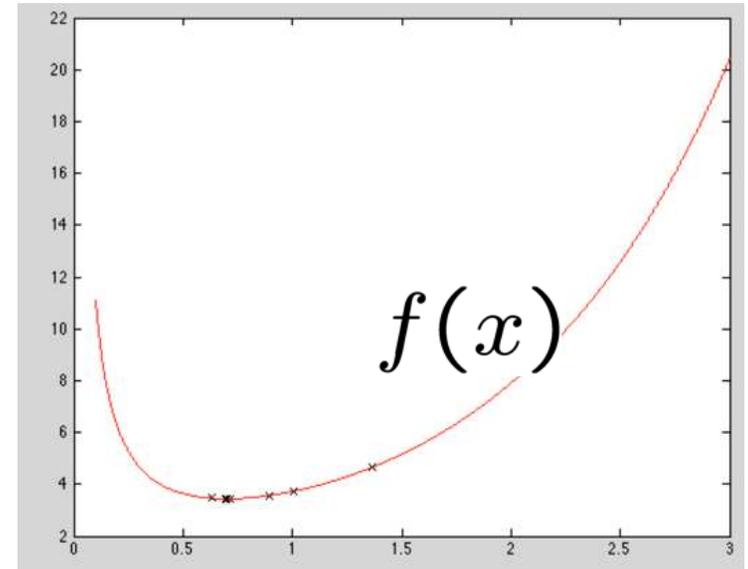
    c=b; fc=fb; b=a; fb=fa; a=x; fa=fx; % Push c off

    dx=xo-x; df=fo-fx; [ x fx dx df ] % Plot

    kk(k)=k; dk(k)=abs(dx);fk(k)=abs(df);
    plot(x,fx,'kx'); hold on
    if abs(df) < 20*eps; break; end;

end;
pause
hold off; semilogy(kk,dk,'r.-',kk,fk,'b.-')

function fx=f(x);
    exx = exp(-x.*x); fx = .5 - x.*exx;
    fx = 1./sin(x);
    fx = exp(x) + 1./x;
```



Example: Successive Parabolic Interpolation

- Use successive parabolic interpolation to minimize $f(x) = 0.5 - xe^{-x^2}$

k	x_k	f_k	$ \Delta x_k / x_k $	$ \Delta f_k / f_k $
0	$1.5000e + 00$	$3.4190e - 01$	—	—
1	$1.0262e + 00$	$1.4200e - 01$	$4.6171e - 01$	$1.4078e + 00$
2	$1.0270e + 00$	$1.4230e - 01$	$7.6100e - 04$	$2.1207e - 03$
3	$7.5682e - 01$	$7.3186e - 02$	$3.5697e - 01$	$9.4439e - 01$
4	$6.2748e - 01$	$7.6743e - 02$	$2.0613e - 01$	$4.6352e - 02$
5	$7.1154e - 01$	$7.1135e - 02$	$1.1814e - 01$	$7.8837e - 02$
6	$7.0803e - 01$	$7.1119e - 02$	$4.9566e - 03$	$2.2605e - 04$
7	$7.0720e - 01$	$7.1118e - 02$	$1.1780e - 03$	$1.0140e - 05$
8	$7.0711e - 01$	$7.1118e - 02$	$1.2640e - 04$	$9.4006e - 08$
9	$7.0711e - 01$	$7.1118e - 02$	$1.5041e - 06$	$1.4127e - 11$
10	$7.0711e - 01$	$7.1118e - 02$	$2.6763e - 08$	$5.4638e - 15$

Superlinear

- Successive Parabolic Interpolation is Newton's Method applied to a quadratic model of the data, $r = 1.324$
- We turn to Newton's method next, $r = 2$

Newton's Method

- Another local quadratic approximation is truncated Taylor series

$$f(x + h) \approx f(x) + f'(x)h + \frac{f''(x)}{2}h^2$$

- By differentiation, minimum of quadratic function given by $h = -f'(x)/f''(x)$
- Suggests iteration scheme

$$x_{k+1} = x_k - f'(x_k)/f''(x_k)$$

- No surprise, Newton's method for minimization is equivalent to Newton's method for root finding applied to $f'(x)$
- Newton's method for finding minimum normally has quadratic convergence rate, but must be started close enough to solution to converge

Example: Newton's Method

- Use Newton's method to minimize $f(x) = 0.5 - xe^{-x^2}$
- First and second derivatives are

$$f'(x) = (2x^2 - 1)e^{-x^2}$$

$$f''(x) = 2x(3 - 2x^2)e^{-x^2}$$

- Newton iteration for zero of f' is given by

$$x_{k+1} = x_k - (2x_k^2 - 1)/(2x_k(3 - 2x_k^2))$$

- Using starting guess $x_0 = 1$, we obtain

k	x_k	f_k
0	1.000	0.132
1	0.500	0.111
2	0.700	0.071
3	0.707	0.071
4	0.707	0.071

Matlab Example: newton1d.m

```
format compact; format longe;

x=1; f=0;

for k=1:10;
    fo = f; exx = exp(-x.*x); f = .5 - x.*exx;

    % fp = (2.*x.*x-1).*exx;
    % fpp = 2.*x.*(3-2.*x.*x).*exx;

    s = -(2.*x.*x-1)/(2.*x.*(3-2.*x.*x));
    x = x+s;

    y = f-fo; [k x f s y] % print convergence
end;
```

Safeguarded Methods

- As with nonlinear equations in one dimension, slow-but-sure and fast-but-risky optimization methods can be combined to provide both robustness and efficiency
- Most library routines for one-dimensional optimization are based on this hybrid approach
- Popular combination is golden section search and successive parabolic interpolation, for which no derivatives are required

Matlab: Successive Parabolic Interpolation: *Naive Bracketing*

```
function x=parab(a,b,c);

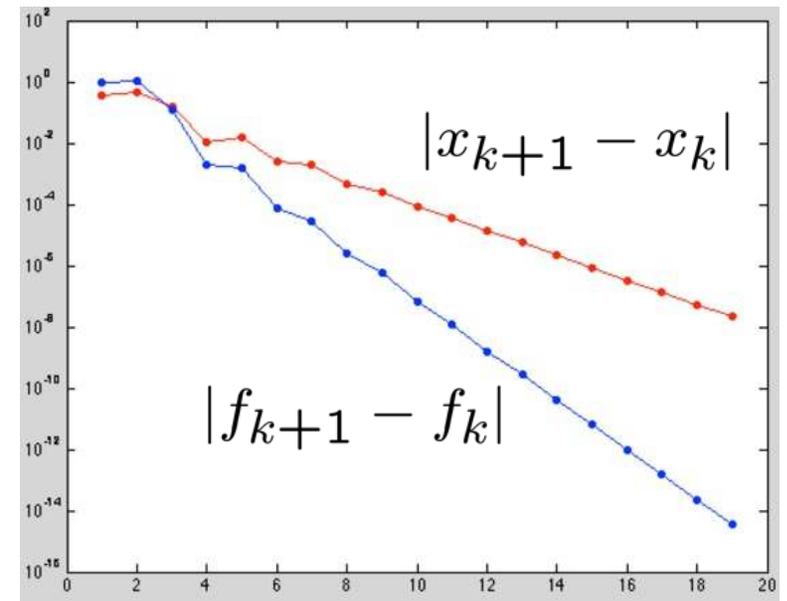
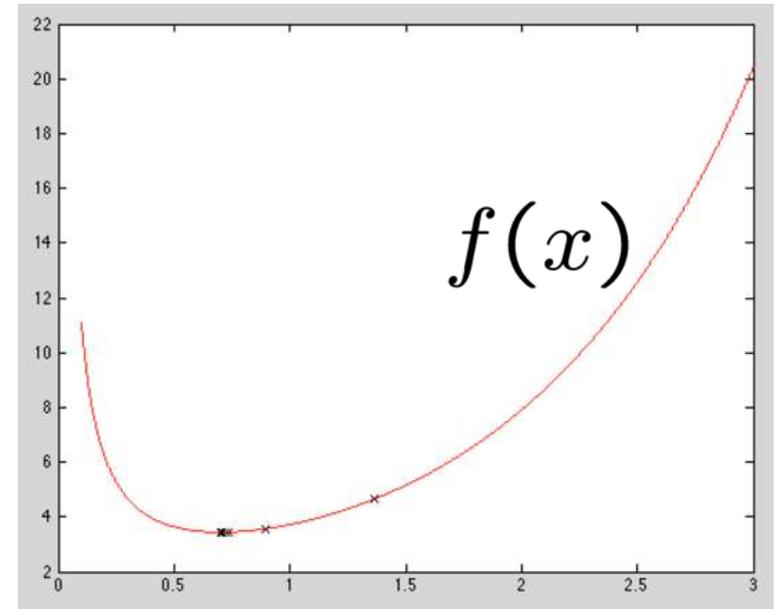
xx=a:.001:c; yy=f(xx); hold off; plot(xx,yy,'r-');
format compact; format longe;

fa=f(a); fb=f(b); fc=f(c);

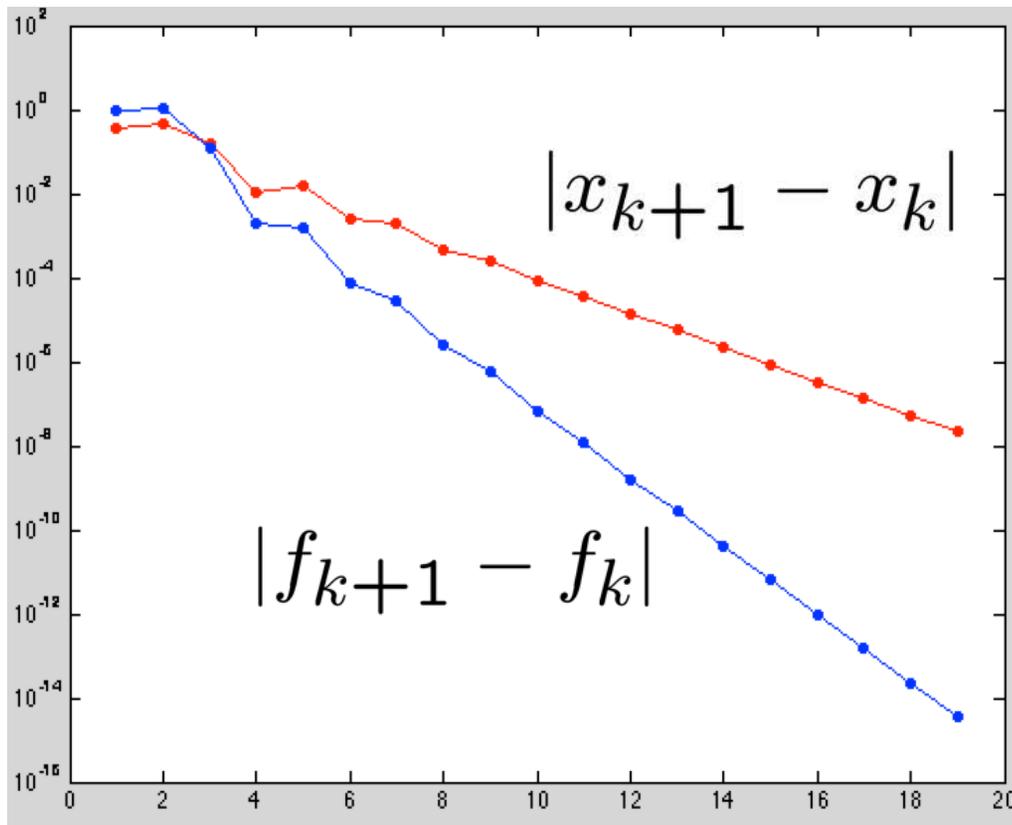
x=1; fx=f(x);

for k=1:90;
    num = ((fb-fc)*(b-a)^2-(fb-fa)*(b-c)^2);
    den = ((fb-fc)*(b-a)-(fb-fa)*(b-c));
    xo=x; fo=fx;
    x=b-.5*num/den; fx = f(x);
    if x>b;
        if fx>fb; c=x; fc=fx; else;
            a=b;fa=fb;b=x;fb=fx; end;
    else
        if fx>fb; a=x; fa=fx; else;
            c=b;fc=fb;b=x;fb=fx; end;
    end;
    dx=xo-x; df=fo-fx;
    [ x fx dx df ]
    kk(k)=k; dk(k)=abs(dx);fk(k)=abs(df);
    plot(x,fx,'kx'); hold on
    if abs(df)<20*eps; break; end;
end;
pause
hold off; semilogy(kk,dk,'r.-',kk,fk,'b.-')

function fx=f(x);
    exx = exp(-x.*x); fx = .5 - x.*exx;
    fx = 1./sin(x);
    fx = exp(x) + 1./x;
```



Convergence Behavior

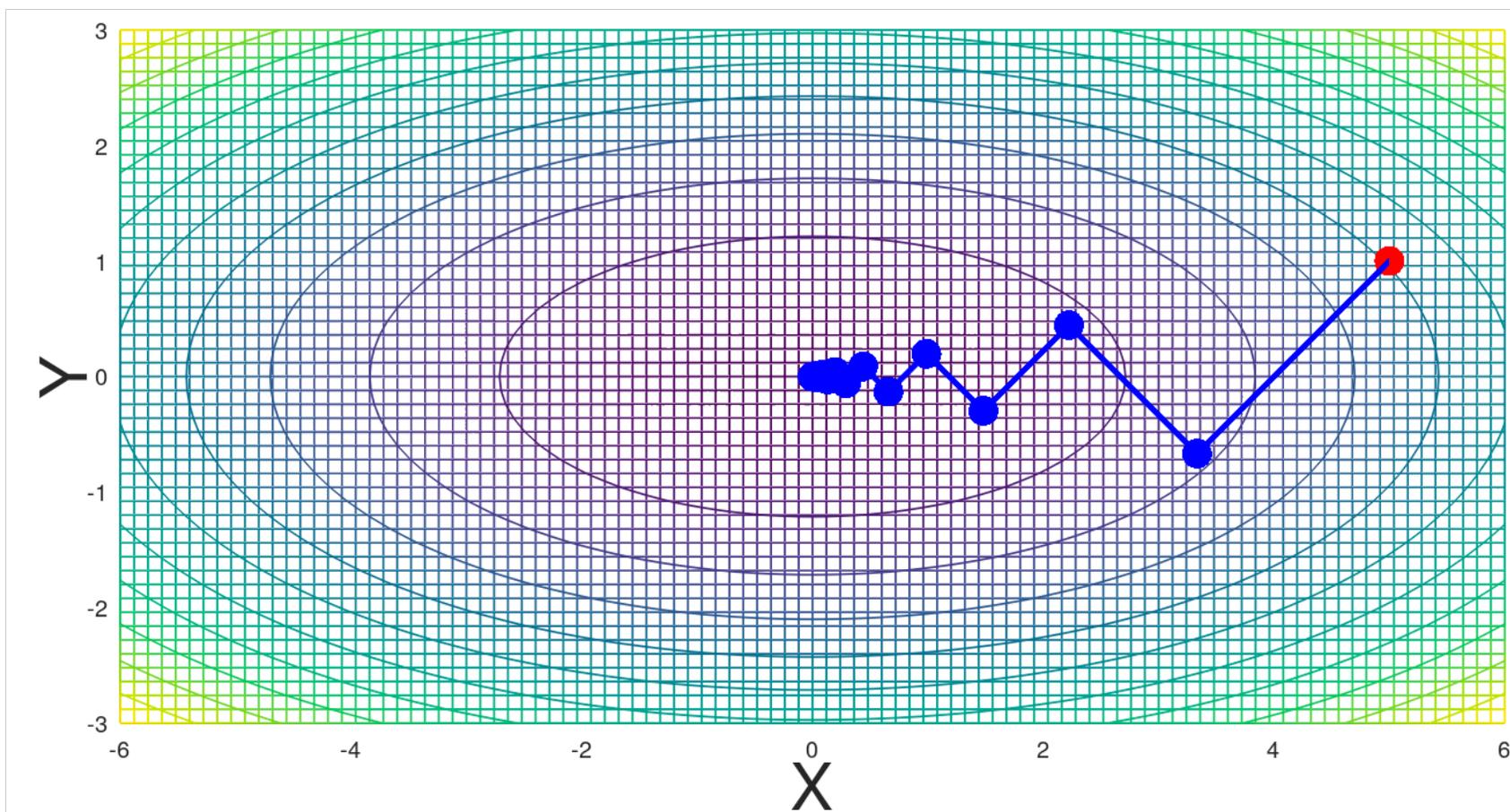


❑ Question 1: What type of convergence is this?

❑ Question 2: What does this plot say about conditioning of

$$\min_x f(x)$$

Methods for Multidimensional Problems



Methods for Multidimensional Problems

- Steepest Descent
- Newton's Method
- Quasi-Newton Methods
- Secant-Updating Methods
- BFGS
- Conjugate Gradient Iteration
- Truncated Newton Methods

Practical Considerations: Multidimensional Problems

- Most of these methods require line searches, which should be robust
- A common approach (e.g., *Numerical Recipes*) is to first bracket the minimum using successively larger sample spacings until one has three points $a < b < c$ with $f(b) < \min(f(a), f(c))$
- Note that you may need to convert your search problem to operate on a single scalar variable, say α , such that

$$f_s(\alpha) = f(\mathbf{x} + \alpha\mathbf{s})$$

where \mathbf{x} is typically your current iterate and \mathbf{s} is the chosen search direction.

- Then a combination of golden section and successive parabolic interpolation can be used to isolate the minimum for α where, presumably one of the endpoints a or b is zero, corresponding to $\alpha = 0$
- *Numerical Recipes* has `mnbrak` and `brent` to perform the bracketing and the safeguarded line search.
- These utilities can be found in almost every language (or converted by chatgpt)

Steepest Descent Method

- Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be real-valued function of n real variables
- At any point \mathbf{x} where gradient vector is nonzero, the negative gradient, $-\nabla f(\mathbf{x})$, points downhill toward lower values of f
- In fact, $-\nabla f(\mathbf{x})$ is locally the direction of *steepest descent*: f decreases more rapidly along direction of negative gradient than along any other
- *Steepest descent method*: starting from initial guess \mathbf{x}_0 , successive approximate solutions given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$$

where α_k is *line search* parameter that determines how far to go in given direction

Gradient of $f(\mathbf{x})$

- $f(\mathbf{x})$ is a *scalar function* of an n -dimensional *vector input*.

$$f(\mathbf{x}) = f(x_1, x_2, \dots, x_n)$$

$$\mathbf{x} = x_1\mathbf{i} + x_2\mathbf{j} + x_3\mathbf{k} \quad (\text{say, for } n=3).$$

- Its *gradient* has n components,

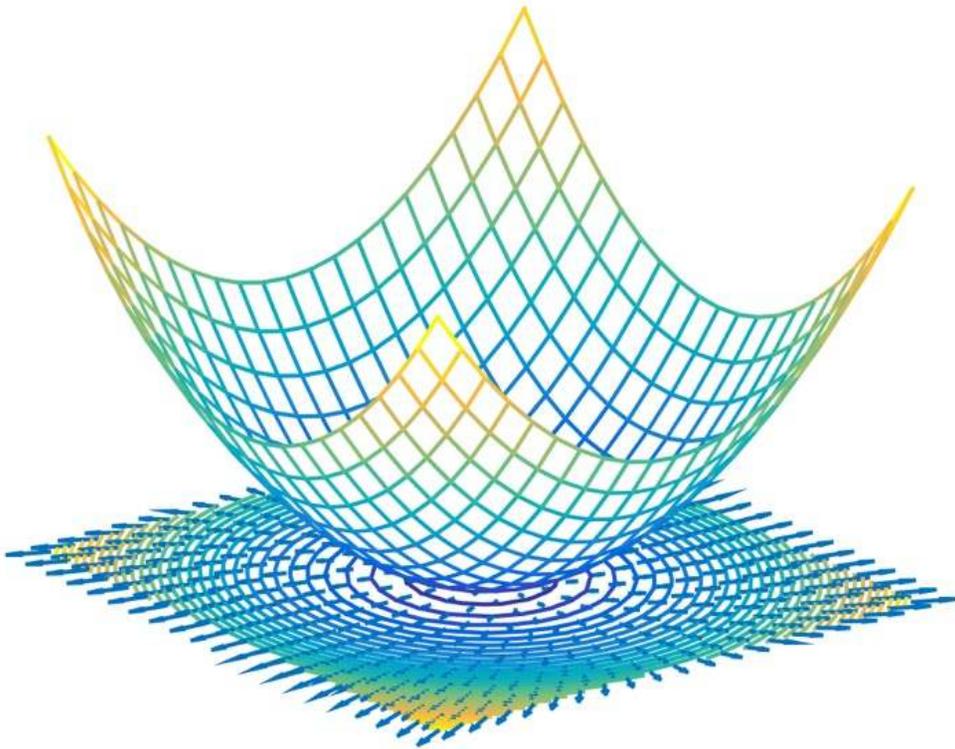
$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial x_1}\mathbf{i} + \frac{\partial f}{\partial x_2}\mathbf{j} + \frac{\partial f}{\partial x_3}\mathbf{k}.$$

- The *vector*, ∇f , is in the domain of $f(\mathbf{x})$ —for each direction, (\mathbf{i} , \mathbf{j} , \mathbf{k}), there is one component of ∇f .

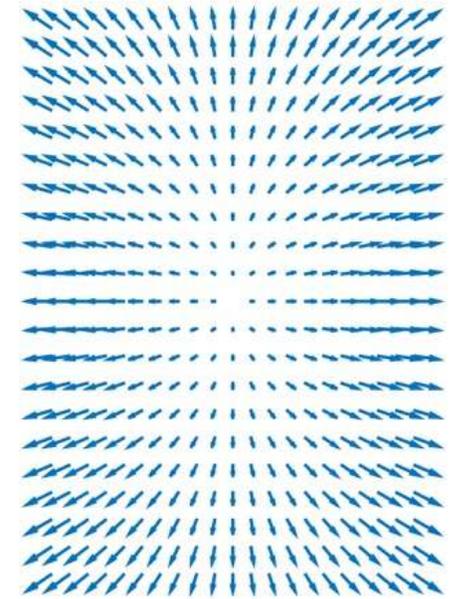
- It is linear in f : $\nabla(2f) = 2\nabla f$, so its units are

$$\frac{[f]}{[x]} = \frac{\text{units of } f}{\text{units of } x}$$

Iso-Contours and Gradients



*Level Sets
(contours,
isosurfaces)*



Gradient field

- ❑ *Gradient points **away** from minimum.*
- ❑ *-Gradient points **towards** the minimum.*
- ❑ *Gradient direction is the direction of steepest **ascent** at point (x,y)*

Steepest Descent, continued

- Given descent direction, such as negative gradient, determine appropriate α_k at each iteration by solving one-dimensional minimization problem,

$$\alpha_k = \operatorname{argmin}_{\alpha} f(\mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k)),$$

which can be solved by the one-dimensional methods introduced earlier

- Steepest descent is very reliable: it can always make progress when gradient is nonzero
- But method is “too local” in its view of the function’s behavior and can cause successive iterates to zigzag back and forth with little progress
- Convergence rate is generally *linear*, with constant factor that can be arbitrarily close to 1

Example: Steepest Descent

- Use steepest descent to minimize

$$f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$$

- Gradient is $\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$

- Taking $\mathbf{x}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$, we have $\nabla f(\mathbf{x}_0) = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$

- Performing search along negative gradient direction,

$$\min_{\alpha_0} f(\mathbf{x}_0 - \alpha_0 \nabla f(\mathbf{x}_0))$$

exact minimum along line is given by $\alpha_0 = 1/3$, so

next approximation is $\mathbf{x}_1 = \begin{bmatrix} 3.333 \\ -0.667 \end{bmatrix}$

Example: Steepest Descent

- Use steepest descent to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$

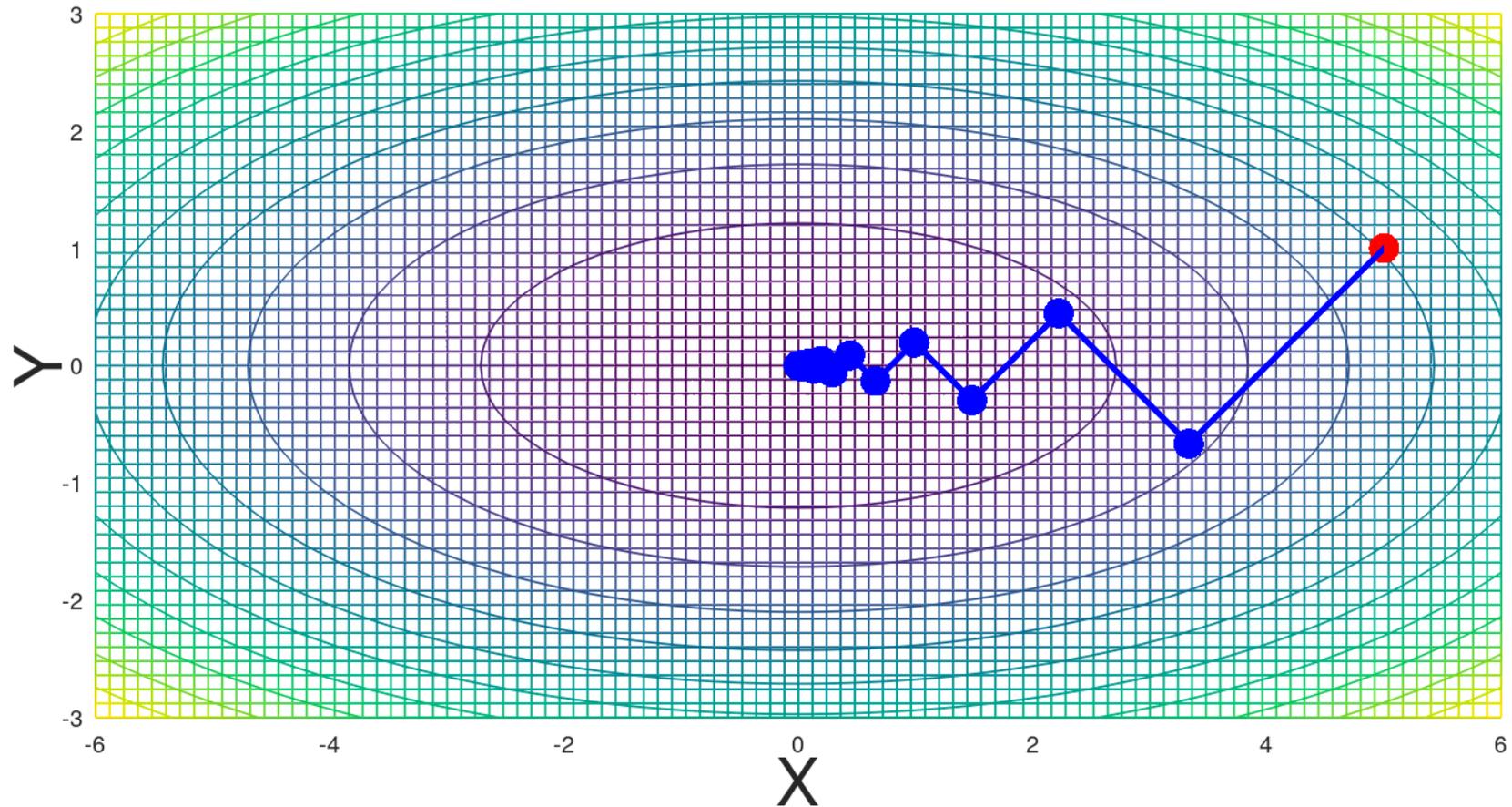
k	x_1	x_2	$f(\mathbf{x}_k)$	$\frac{\partial f}{\partial x_1}$	$\frac{\partial f}{\partial x_2}$
0	5.0000	1.0000	15.0000	5.0000	5.0000
1	3.3333	-0.6667	6.6667	3.3333	-3.3333
2	2.2222	0.4444	2.9630	2.2222	2.2222
3	1.4815	-0.2963	1.3169	1.4815	-1.4815
4	0.9877	0.1975	0.5853	0.9877	0.9877
5	0.6584	-0.1317	0.2601	0.6584	-0.6584
6	0.4390	0.0878	0.1156	0.4390	0.4390
7	0.2926	-0.0585	0.0514	0.2926	-0.2926
8	0.1951	0.0390	0.0228	0.1951	0.1951

- Notice that, for this example, the x_2 value is oscillating about 0—the solution is zigzagging as $x_1 \rightarrow 0$.
- It's not unreasonable to expect more rapid convergence if we occasionally take a half step (at least for 2D cases)
- steep1.m and steep2.m demos

Example: Steepest Descent

- Use steepest descent to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$

k=34 ds,df=2.5754e-06 -2.5754e-06 1.9897e-11



Newton's Method

- Broader view can be obtained by local quadratic approximation, which is equivalent to Newton's method
- In multidimensional optimization, seek zero of gradient, so *Newton Iteration* has the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_f^{-1}(\mathbf{x}_k) \nabla f(\mathbf{x}_k)$$

where $\mathbf{H}_f(\mathbf{x})$ is the Hessian matrix of second partial derivatives of f ,

$$[\mathbf{H}_f(\mathbf{x})]_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$$

Newton's Method, continued

- Do not explicitly invert Hessian matrix, but instead solve linear system

$$\mathbf{H}_f(\mathbf{x}_k)\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$$

for Newton step \mathbf{s}_k and take as next iterate

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$$

- Convergence rate of Newton's method for minimization is normally quadratic
- As usual, Newton's method is unreliable unless started close enough to solution to converge

Example: Newton's Method

- Use Newton's method to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$

- Gradient and Hessian are

$$\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix} \quad \text{and} \quad \mathbf{H}_f(\mathbf{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

- Taking $\mathbf{x}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$, we have $\nabla f(\mathbf{x}_0) = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$

- Linear system for Newton step is $\begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} \mathbf{s}_0 = -\begin{bmatrix} 5 \\ 5 \end{bmatrix}$, so

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{s}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix} + \begin{bmatrix} -5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

which is exact solution, as expected for quadratic function

- Generally, Hessian will not be constant, and deviation from “exact” step will be governed by the amount of variance between $\mathbf{H}_f(\mathbf{x}_k)$ and $\mathbf{H}_f(\mathbf{x}^*)$

Newton's Method, continued

- In principle, line search parameter is unnecessary with Newton's method because quadratic model determines the direction *and* length of next step
- When started far from the solution, however, it may still be advisable to perform line search along direction of Newton step \mathbf{s}_k to make method more robust (*damped Newton*)
- Once iterates are near the solution then $\alpha_k = 1$ should suffice for quadratic convergence

Newton's Method, continued

- If objective function f has continuous second partial-derivatives then Hessian matrix \mathbf{H}_f is symmetric and, near \mathbf{x}^* , is positive definite
- Thus, using Cholesky, linear system for step to next iterate can be solved in about half the work required for LU factorization
- Far from minimum, $\mathbf{H}_f(\mathbf{x}_k)$ may not be positive definite, so Newton step \mathbf{s}_k may not be a *descent direction* for f , i.e., we may not have

$$\nabla f(\mathbf{x}_k)^T \mathbf{s}_k < 0$$

- In this case, alternative descent direction can be computed, such as $-\nabla f(\mathbf{x}_k)$, and then perform line search

Quasi-Newton Methods

- Newton's method costs $O(n^3)$ arithmetic and $O(n^2)$ scalar function evaluations per iteration for dense problem
- Many variants of Newton's method improve reliability and reduce overhead
- *Quasi-Newton* methods have form

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{B}_k^{-1} \nabla f(\mathbf{x}_k)$$

where α_k is line search parameter and \mathbf{B}_k is approximation to Hessian

- Many quasi-Newton methods are
 - *more robust than Newton's method,*
 - *are superlinearly convergent, and*
 - *have lower overhead per iteration,*

which more than offsets their slower convergence rate

Secant Updating Methods

- Could use Broyden's method to seek zero of gradient, but this would not preserve the symmetry of the Hessian matrix
- Several secant updating formulas have been developed for minimization that not only preserve symmetry in approximate Hessian but also preserve positive definiteness
- Symmetry reduces amount of work required by about half while positive definiteness guarantees that quasi-Newton step will be a descent direction

BFGS Method

One of the most effective secant updating methods for minimization is *BFGS* (Broyden–Fletcher–Goldfarb–Shanno)

\mathbf{x}_0 = initial guess

\mathbf{B}_0 = initial Hessian approximation

for $k = 0, 1, 2, \dots$

Solve $\mathbf{B}_k \mathbf{s}_k = -\nabla f(\mathbf{x}_k)$ for \mathbf{s}_k

$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$  **Replace with line search** $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$

$\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$

$\mathbf{B}_{k+1} = \mathbf{B}_k + (\mathbf{y}_k \mathbf{y}_k^T) / (\mathbf{y}_k^T \mathbf{s}_k) - (\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k) / (\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k)$

end

BFGS Method, continued

- In practice, factorization of \mathbf{B}_k is updated rather than \mathbf{B}_k itself, so linear system for \mathbf{s}_k can be solved at cost of $O(n^2)$ rather than $O(n^3)$
- Unlike Newton's method for minimization, no second derivatives are required
- Can start with $\mathbf{B}_0 = \mathbf{I}$, so initial step is along negative gradient
- Second derivative information is gradually built up in \mathbf{B}_k over successive iterations
- BFGS normally has superlinear convergence rate, even though approximate Hessian does not necessarily converge to true Hessian
- Line search can be used to enhance effectiveness (especially if using $\mathbf{B}_0 = \mathbf{I}$ on step 1)

Example: BFGS Method

- Use BFGS to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$

- Gradient is $\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$

- Taking $\mathbf{x}_0 = [5 \ 1]^T$ and $\mathbf{B}_0 = \mathbf{I}$, initial step is negative gradient, so

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{s}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix} - \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} 0 \\ -4 \end{bmatrix}$$

- Updating approximate Hessian using BFGS formula yields

$$\mathbf{B}_1 = \begin{bmatrix} 0.667 & 0.333 \\ 0.333 & 0.667 \end{bmatrix}$$

- Then new step is computed and process is repeated

Example: BFGS

- Use BFGS to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$

k	x_1	x_2	$f(\mathbf{x}_k)$	$\frac{\partial f}{\partial x_1}$	$\frac{\partial f}{\partial x_2}$
0	5.0000	1.0000	15.0000	5.0000	5.0000
1	0.0000	-4.0000	40.0000	0.0000	-20.0000
2	-2.2222	0.4444	2.9630	-2.2222	2.2222
3	0.8163	0.0816	0.3499	0.8163	0.4082
4	-0.0092	-0.0153	0.0006	-0.0092	-0.0767
5	-0.0005	0.0009	0.0000	-0.0005	0.0046

- Increase in function value can be avoided by using line search, which generally enhances convergence
- For quadratic objective function, BFGS with exact line search finds exact solution in at most n iterations

Example: BFGS

- Use BFGS to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$
- bfgs1.m

```
k=0; x=[5,1]; f=funcf(x); g=gradf(x); s=-g; gg1=g*g';

n=2; B=eye(n);
for k=1:100;
    xo = x; fo = f; go=g;
    s = -B\go';
    [a,xm,f] = line_search(s',x,xtol); a=1;
    x = x+a*s'; f = funcf(x); g = gradf(x); dx = xo-x; df=fo-f;
    y = (g - go)';
    Bs = B*s;
    B = B + (y*y')/(y'*s)-(Bs*Bs')/(s'*Bs);

    fprintf("%2d %7.4f %7.4f %7.4f %7.4f %7.4f",[k x f g])
    plot([xo(1),x(1)],[xo(2),x(2)],'k--',lw,2,x(1),x(2),'k.',ms,22);
    title(['k=' int2str(k) ' ds,df=' num2str([dx,df])],fs,24);
    pause

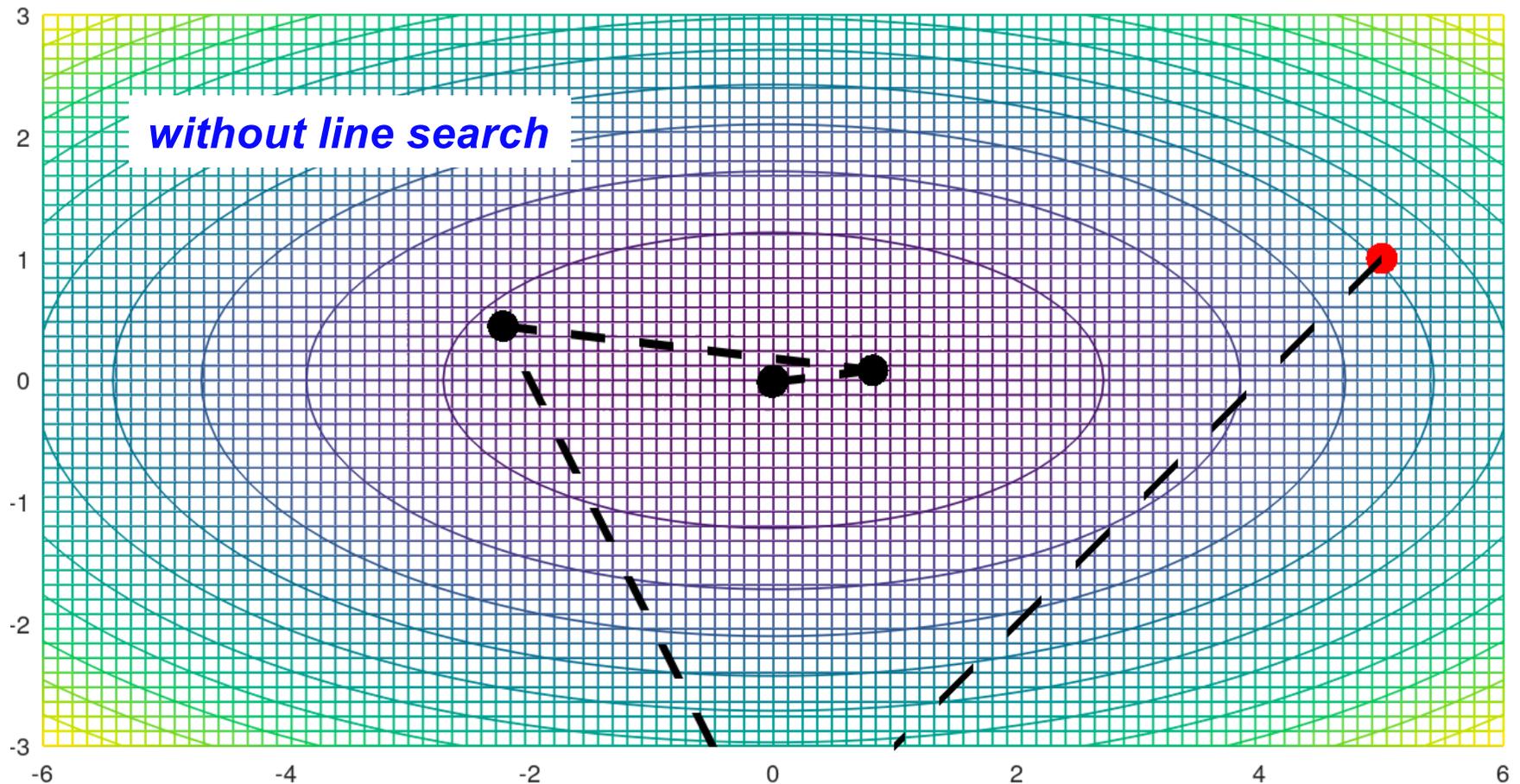
    if norm(dx)< xtol; break; end;
    if abs (df)< ftol; break; end;

end;
```

Example: BFGS

- Use BFGS to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$
- bfgs1.m

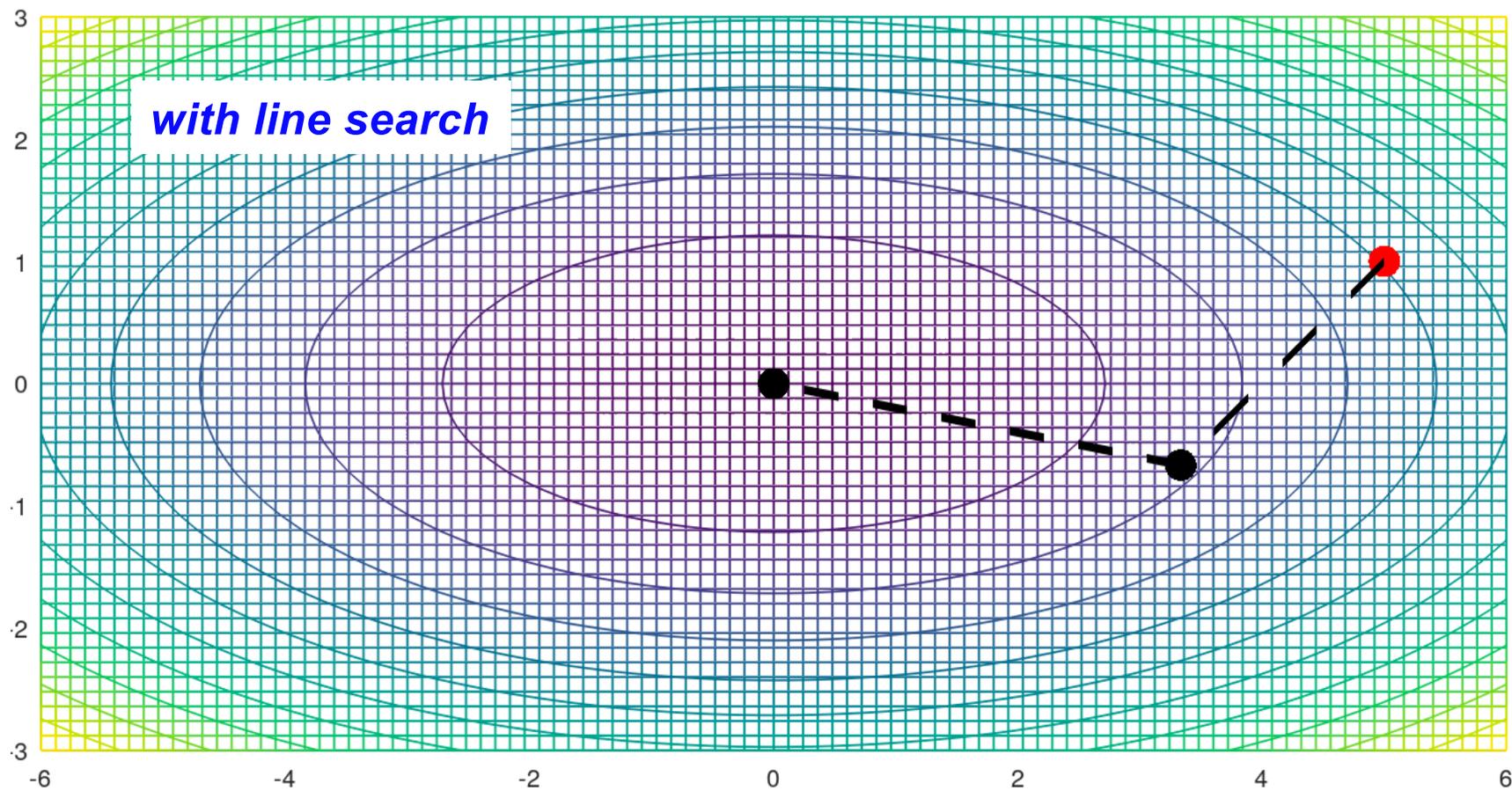
k=9 ds,df=4.4714e-11 8.663e-11 1.9699e-20



Example: BFGS

- Use BFGS to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$
- bfgs1.m

k=3 ds,df=-4.3032e-17 -1.0758e-16 2.9859e-32



Conjugate Gradient Method

- Another method that does not require explicit second derivatives and does not even store approximation to the Hessian matrix is *conjugate gradient* (CG) method
- CG generates sequence of conjugate search directions, implicitly accumulating information about the Hessian matrix
- For quadratic objective function, CG with exact line search is theoretically exact after at most n iterations
- CG is effective for general unconstrained minimization

Conjugate Gradient Method, continued

\mathbf{x}_0 = initial guess

$\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$

$\mathbf{s}_0 = -\mathbf{g}_0$

for $k = 0, 1, 2, \dots$

Choose α_k to minimize $f(\mathbf{x}_k + \alpha_k \mathbf{s}_k)$

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$

$\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$

$\beta_{k+1} = (\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}) / (\mathbf{g}_k^T \mathbf{g}_k)$

$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{s}_k$

end

- Alternative formula for β_{k+1} is

$$\beta_{k+1} = ((\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{g}_{k+1}) / (\mathbf{g}_k^T \mathbf{g}_k) \quad \textit{Try this ?}$$

Example: Conjugate Gradient Method

- Use CG method to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$
- Gradient is $\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$
- Taking $\mathbf{x}_0 = [5 \ 1]^T$ and $\mathbf{B}_0 = \mathbf{I}$, initial search direction is negative gradient

$$\mathbf{s}_0 = -\mathbf{g}_0 = -\nabla f(\mathbf{x}_0) = -\begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

- Exact minimum along line is given by $\alpha_0 = 1/3$, so next approximation is $\mathbf{x}_1 = [3.333 \ -0.667]^T$, and we compute new gradient

$$\mathbf{g}_1 = \nabla f(\mathbf{x}_1) = \begin{bmatrix} 3.333 \\ -3.333 \end{bmatrix}$$

CG Example, continued

- So far, there is no difference from steepest descent method
- At this point, however, rather than search along new negative gradient, we compute instead

$$\beta_1 = (\mathbf{g}_1^T \mathbf{g}_1) / (\mathbf{g}_0^T \mathbf{g}_0) = 0.444$$

which gives as the next (*conjugate*) search direction

$$\mathbf{s}_1 = -\mathbf{g}_1 + \beta_1 \mathbf{s}_0 = \begin{bmatrix} 3.333 \\ -3.333 \end{bmatrix} - 0.444 \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} -5.556 \\ 1.111 \end{bmatrix}$$

- Minimum along this direction is given by $\alpha_1 = 0.6$, which gives exact solution at origin, as expected for quadratic function
- cg1.m demo

Example: Conjugate Gradients

- Use CG to minimize $f(\mathbf{x}) = 0.5x_1^2 + 2.5x_2^2$

k	x_1	x_2	$f(\mathbf{x}_k)$	$\frac{\partial f}{\partial x_1}$	$\frac{\partial f}{\partial x_2}$
0	5.0000	1.0000	15.0000	5.0000	5.0000
1	3.3333	-0.6667	6.6667	3.3333	-3.3333
2	-0.0000	0.0000	0.0000	-0.0000	0.0000

- Here, we see that CG converges in 2 iterations, which we would expect for this quadratic form in two dimensions
- steep2.m compares CG to steepest descent for the Rosenbrock equation

Example: CG vs Steepest Descent

- CG and steepest descent differ only by about one line of code
- Their convergence rates, however, are dramatically different

```
k=0; x=[5,1]; f=funcf(x); g=gradf(x); s=-g; gg1=g*g';

for k=1:100;
    xo = x; fo = f;
    [a,xm,f] = line_search(s,x,xtol);
    x = xo+a*s; g = gradf(x); dx = xo-x; df=fo-f;

    fprintf("%2d %7.4f %7.4f %7.4f %7.4f %7.4f",[k x f g])

    if norm(dx)< xtol; break; end;
    if abs (df)< ftol; break; end;

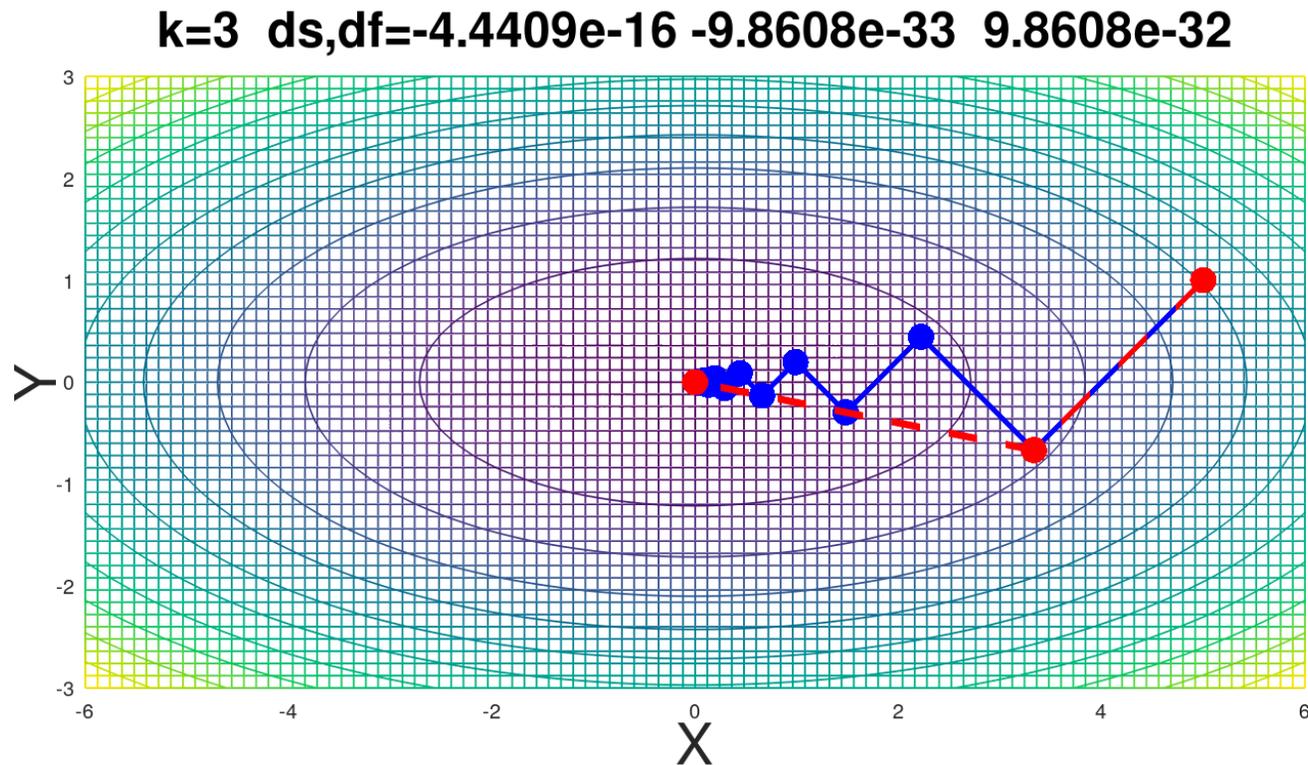
    gg0 = gg1; gg1=g*g'; beta = gg1/gg0;
% beta = 0; % Uncomment this for steepest descent
    s = -g + beta*s;

end;
```

- Moreover, for quadratic objective functions in n dimensions, CG will theoretically converge in at most $m \leq n$ iterations, where m =number of distinct eigenvalues in the (constant) Hessian matrix

Example: CG vs Steepest Descent

- CG and steepest descent differ only by about one line of code
- Their convergence rates, however, are dramatically different



- Moreover, for quadratic objective functions in n dimensions, CG will theoretically converge in at most $m \leq n$ iterations, where m =number of distinct eigenvalues in the (constant) Hessian matrix

Conjugate Gradient Iteration for Optimization

- Consider the objective function

$$\begin{aligned} f(x) &= \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} = \mathbf{x}^T \mathbf{b} + c \\ &= \frac{1}{2} (\mathbf{x} - \mathbf{x}^*)^T \mathbf{H} (\mathbf{x} - \mathbf{x}^*) + \tilde{c}. \end{aligned}$$

- Let's take the case $\mathbf{H}=\text{constant}$, which is essentially the case for $\mathbf{x} \approx \mathbf{x}^*$ as it represents the first two terms in the Taylor series about \mathbf{x}^* .

- The gradient is

$$\nabla f = \mathbf{H} \mathbf{x} - \mathbf{b},$$

and the minimizer is the point \mathbf{x}^* such that $\nabla f(\mathbf{x}^*) = 0$,

$$\implies \mathbf{H} \mathbf{x}^* = \mathbf{b}.$$

- Near the minimizer, need to solve a linear system.

Update Algorithm

- For steepest descents, Newton's method, CG, etc., the update algorithm is of the form:

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{s}_k \\ &= \mathbf{x}_0 + \alpha_0 \mathbf{s}_0 + \alpha_1 \mathbf{s}_1 + \cdots + \alpha_k \mathbf{s}_k.\end{aligned}$$

- If $\mathbf{x}_0 = \mathbf{0}$,

$$\begin{aligned}\mathbf{x}_{k+1} &= \underbrace{[\mathbf{s}_0 \ \mathbf{s}_1 \ \cdots \ \mathbf{s}_k]}_{\mathbf{S}_k} \underbrace{\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_k \end{pmatrix}}_{\mathbf{a}_k} \\ &= \mathbf{S}_k \mathbf{a}_k.\end{aligned}$$

- That is, \mathbf{x}_{k+1} is a linear combination of the preceding $k + 1$ search directions \mathbf{s}_j .

- **Main Idea:** Find $\mathbf{a}_k = [\alpha_0 \alpha_1 \cdots \alpha_k]^T$ such that

$$\mathbf{S}_k \mathbf{a}_k \approx \mathbf{x}^*.$$

- Use linear least squares, with a computable inner product.
- Since we only know $\nabla f = \mathbf{H}\mathbf{x} - \mathbf{b} = \mathbf{H}\mathbf{x} - \mathbf{H}\mathbf{x}^* = \mathbf{H}(\mathbf{x} - \mathbf{x}^*)$, rather than \mathbf{x}^* itself, it makes sense to use the \mathbf{H} inner product:
- Actually, we won't even need to know \mathbf{H} or \mathbf{b} !
- For the moment, we proceed as if we do know them.

Linear Least Squares in H Inner-Product.

- In the \mathbf{H} -norm, the **closest** element in our search space ($= \text{span}\{\mathbf{s}_0 \cdots \mathbf{s}_k\}$) is given by the vector $\mathbf{S}_k \mathbf{a}_k$ satisfying,

$$\min_{\mathbf{a}_k} [\mathbf{S}_k \mathbf{a}_k - \mathbf{x}^*]_H \implies \mathbf{s}_i^T \mathbf{H} (\mathbf{S}_k \mathbf{a}_k - \mathbf{x}^*) = 0, \quad i = 0, \dots, k.$$

$$\mathbf{s}_i^T \mathbf{H} \mathbf{S}_k \mathbf{a}_k = \mathbf{s}_i^T \mathbf{H} \mathbf{x}^* = \mathbf{s}_i^T \mathbf{b}$$

$$(\mathbf{S}_k^T \mathbf{H} \mathbf{S}_k) \mathbf{a}_k = \mathbf{S}_k^T \mathbf{b}.$$

- If $\mathbf{s}_i^T \mathbf{H} \mathbf{s}_j = 0$ for $i \neq j$, then $\mathbf{S}_k^T \mathbf{H} \mathbf{S}_k$ is **diagonal**:

$$(\mathbf{S}_k^T \mathbf{H} \mathbf{S}_k) \mathbf{a}_k = \begin{bmatrix} \mathbf{s}_0^T \mathbf{H} \mathbf{s}_0 & & \\ & \ddots & \\ & & \mathbf{s}_k^T \mathbf{H} \mathbf{s}_k \end{bmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_k \end{pmatrix} = \begin{pmatrix} \mathbf{s}_0^T \mathbf{b} \\ \mathbf{s}_1^T \mathbf{b} \\ \vdots \\ \mathbf{s}_k^T \mathbf{b} \end{pmatrix}.$$

- Solving for α_k becomes trivial:

$$\alpha_k = \frac{\mathbf{s}_k^T \mathbf{b}}{\mathbf{s}_k^T \mathbf{H} \mathbf{s}_k}.$$

- Note that, for optimization problems, we don't know \mathbf{H} nor \mathbf{b} .
 - We can find α_k , however, through a **line search** in the direction \mathbf{s}_k .
- The main point is that each update step yields the **best fit** in \mathbf{S}_k , *independent* of the previous steps,

$$\begin{aligned} \mathbf{x}_{k+1} &= \alpha_0 \mathbf{s}_0 + \alpha_1 \mathbf{s}_1 + \cdots + \alpha_{k-1} \mathbf{s}_{k-1} + \alpha_k \mathbf{s}_k \\ &= \mathbf{x}_k + \alpha_k \mathbf{s}_k. \end{aligned}$$

- Thus, we have a *short-term recurrence* to go from \mathbf{x}_k to \mathbf{x}_{k+1} that yields the optimal solution.
- The *only* requirement for optimality is: $\mathbf{s}_k \perp_H \mathbf{S}_{k-1}$.

- The *only* requirement for optimality is: $\mathbf{s}_k \perp_H \mathbf{S}_{k-1}$.
- Thus, set:

$$\mathbf{s}_{k+1} = - \left(\mathbf{g}_{k+1} - \sum_{j=1}^k \gamma_j \mathbf{s}_j \right)$$

with

$$\gamma_j = \frac{\mathbf{s}_j^T \mathbf{H} \mathbf{g}_{k+1}}{\mathbf{s}_j^T \mathbf{H} \mathbf{s}_j} = 0.$$

- For constant SPD \mathbf{H} , $\gamma_j = 0$ for $j < k$ because the gradient is $\mathbf{H} \times$ the error: $\mathbf{g}_{k+1} = \mathbf{H} \mathbf{e}_{k+1}$ and the error is orthogonal to the search space because of the *best fit* property, $\mathbf{e}_{k+1} \perp_H \mathbf{H} \mathbf{S}_j$, $j < k$
- So, we also have a short-term recurrence for \mathbf{s}_{k+1} :

$$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{s}_k,$$

with (after some manipulation...)

$$\beta_{k+1} = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \text{ or } \beta_{k+1} = \frac{(\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k}.$$

Summary of CG for Optimization

- CG iteration produces projection of \mathbf{x}^* onto $\text{span}\{\mathbf{s}_0 \mathbf{s}_1 \dots \mathbf{s}_k\}$
- For constant SPD Hessian, \mathbf{H} , CG will be exact after m iterations, where $m \leq n$ is the number of distinct eigenvalues in \mathbf{H}
- For nonconstant $\mathbf{H}(\mathbf{x})$, CG is not exact. However, as $\mathbf{x} \longrightarrow \mathbf{x}^*$, we have $\mathbf{H}(\mathbf{x}) \longrightarrow \mathbf{H}(\mathbf{x}^*)$, which is constant.
- Potentially important to restart orthogonalization process after n iterations, otherwise $\mathbf{S}^T \mathbf{H} \mathbf{S}$ will be $k \times k$ with $k > n$, and therefore singular
- Method does not need \mathbf{H} nor \mathbf{b} . Only ∇f and line search along direction \mathbf{s}_k are required

Example: Rosenbrock Function

- Compare steepest descent, CG, and BFGS for Rosenbrock [1960] function:

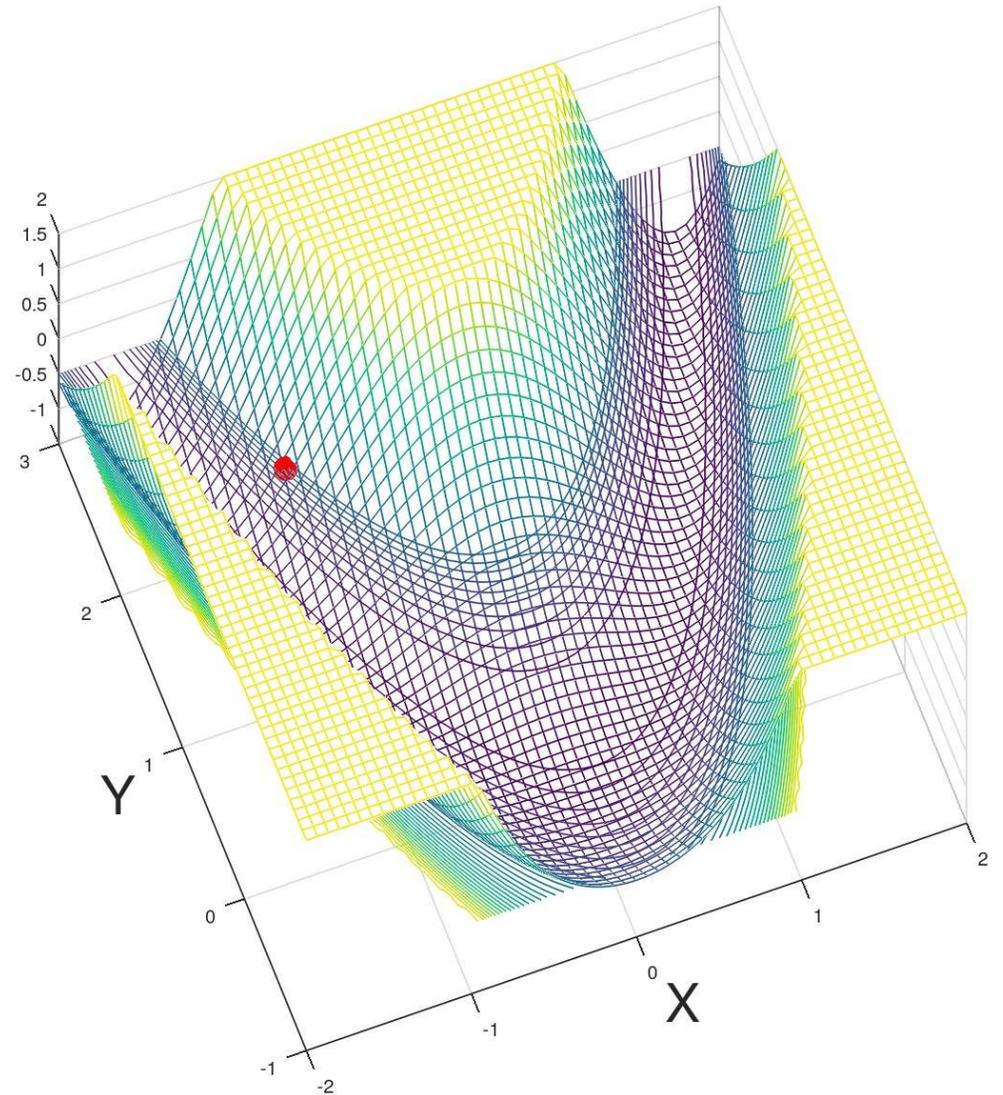
$$f(\mathbf{x}) = (a - x)^2 + b(y - x^2)^2, \text{ with } a = 1, b = 100$$

- Minimum is at $\mathbf{x}^* = [x^*, y^*]^T = [a, a^2]^T$

steep2.m

cg2.m

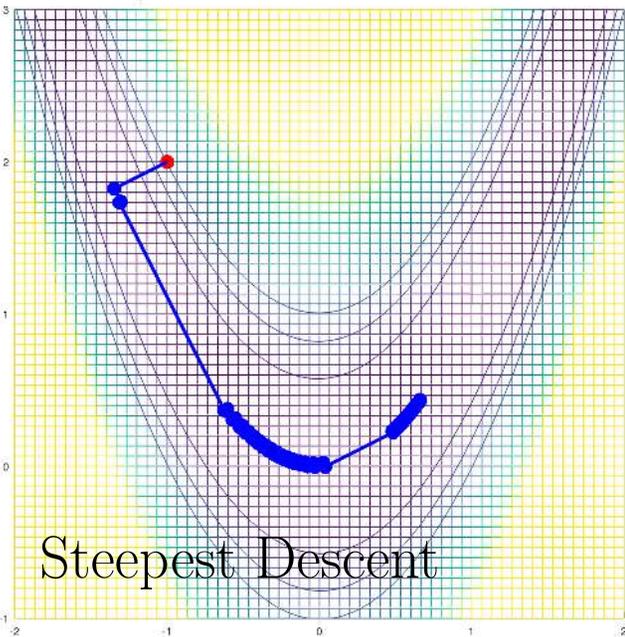
bfgs2.m



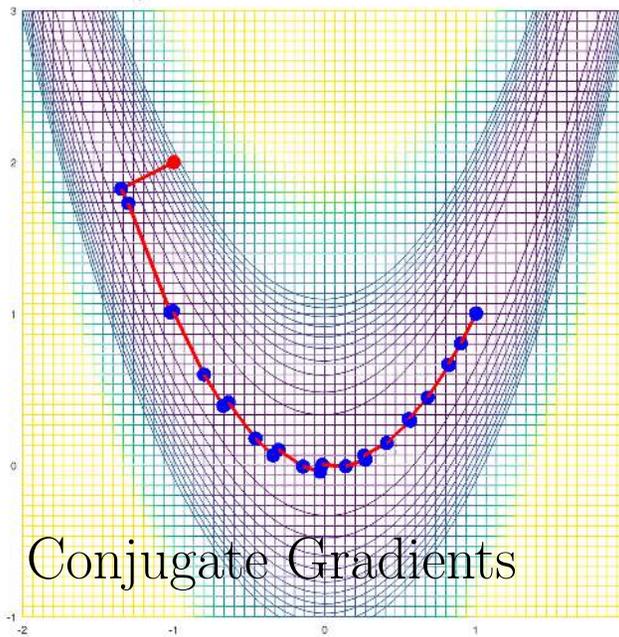
Example: Rosenbrock Function

- Compare steepest descent, CG, and BFGS for Rosenbrock [1960] function:
 $f(\mathbf{x}) = (a - x)^2 + b(y - x^2)^2$, with $a = 1$, $b = 100$
- Minimum is at $\mathbf{x}^* = [x^*, y^*]^T = [a, a^2]^T$

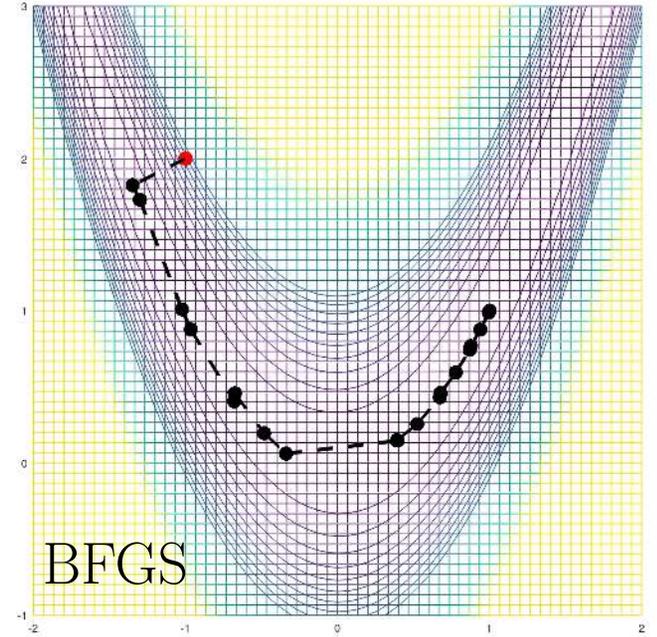
k=100 ds,df=0.00094194 -0.0018652 0.00096857



k=27 ds,df=1.3749e-06 2.0963e-07 6.4711e-10

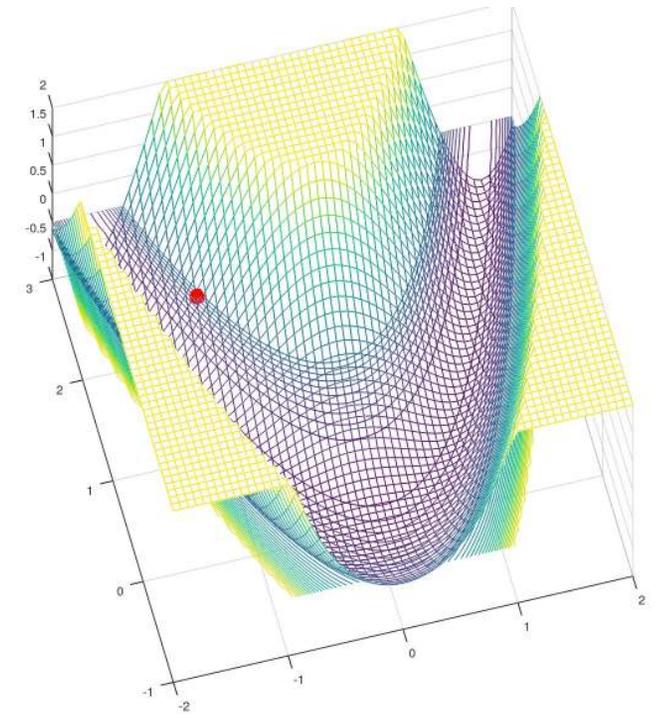


k=23 ds,df=-5.4295e-10 -8.9385e-10 3.9829e-18



Example: Rosenbrock Function

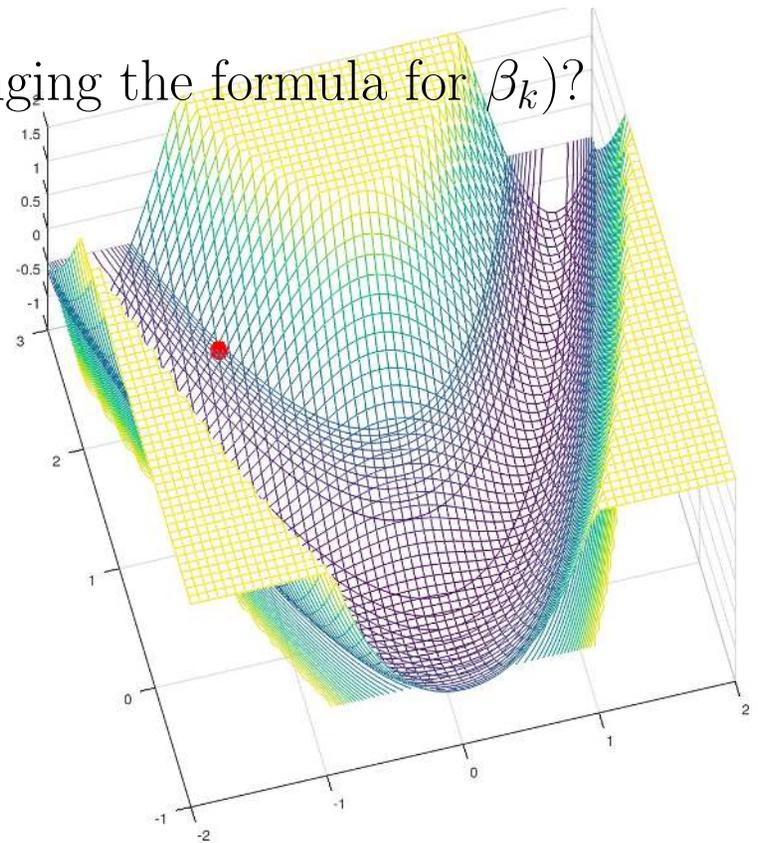
- Compare steepest descent, CG, and BFGS for Rosenbrock [1960] function:
 $f(\mathbf{x}) = (a - x)^2 + b(y - x^2)^2$, with $a = 1$, $b = 100$
- Minimum is at $\mathbf{x}^* = [x^*, y^*]^T = [a, a^2]^T$



- With $\mathbf{x}_0 = [-1, 2]$, steepest descent has not converged after a 100 iterations
- Restarted CG-3 ($\beta = 0$ if $\text{mod}(k,3)=0$) requires 27 iterations
- BFGS with line search requires 23 iterations

Practical Considerations: Rosenbrock Function

- Steepest descent can be improved by taking a damped step, $\alpha_k = \frac{1}{2}\alpha_k$, every few iterations
- BFGS starting with $\mathbf{B}_0 = \mathbf{I}$ generally requires a line search.
- CG requires setting $\beta_k = 0$ every few ($\approx n + 1$) iterations
- Could RP-CG outperform FR-CG (i.e., changing the formula for β_k)?

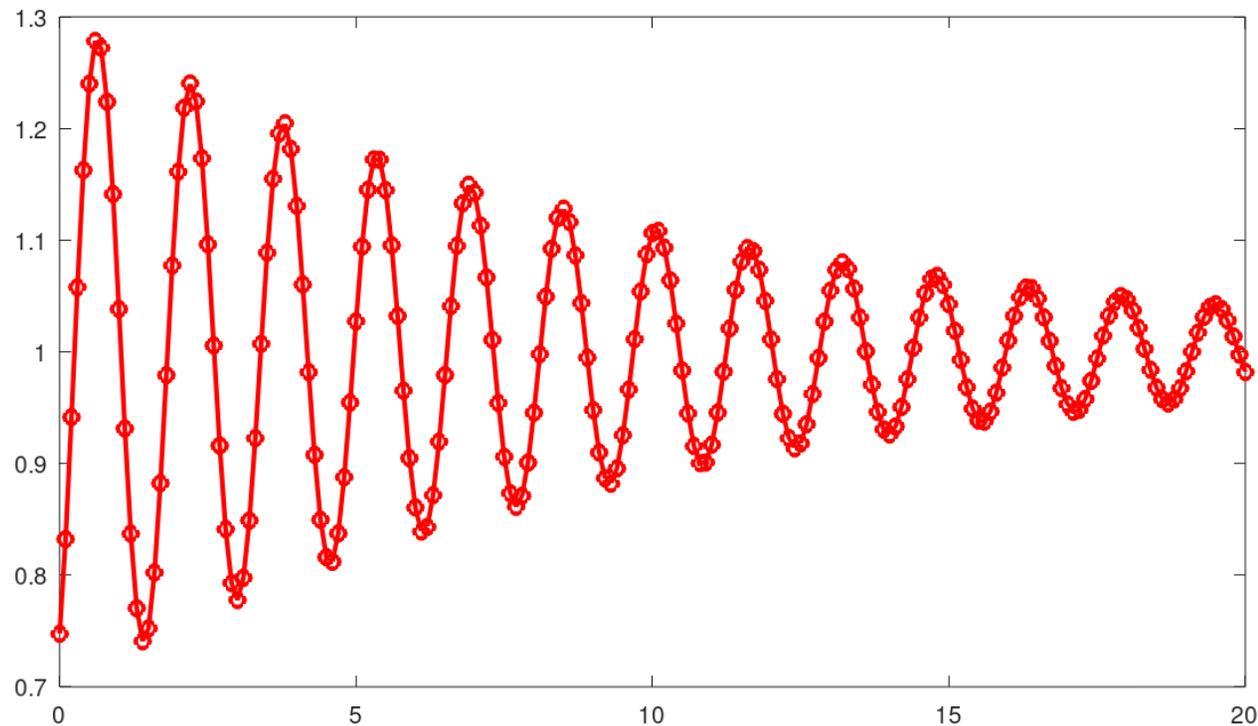


Nonlinear Least Squares

Nonlinear Least Squares

- Given data (t_i, b_i) , $i = 1, \dots, m$, find vector $\mathbf{x} \in \mathbb{R}^n$ of parameters that give “best fit” in least squares sense to model function $f(t, \mathbf{x})$, where f is nonlinear in \mathbf{x}
- **Example:** find a decaying sinusoidal function that matches observed data.

$$f(t, \mathbf{x}) = ae^{-bt} \sin(\omega t - \phi) + c$$



- Here, the vector of unknown parameters is $\mathbf{x} = [a, b, \omega, \phi, c]^T$

Nonlinear Least Squares

- Given data (t_i, b_i) , $i = 1, \dots, m$, find vector $\mathbf{x} \in \mathbb{R}^n$ of parameters that give “best fit” in least squares sense to model function $f(t, \mathbf{x})$, where f is nonlinear in \mathbf{x}
- **Example:** find a decaying sinusoidal function that matches observed data.

$$f(t, \mathbf{x}) = ae^{-bt} \sin(\omega t - \phi) + c$$

- Here, the vector of unknown parameters is $\mathbf{x} = [a, b, \omega, \phi, c]^T$
- Most often, we are interested in the *decay rate*, b , and the *frequency*, ω .
Why?
- Because these parameters are intrinsic to the physical system, whereas phase (ϕ), amplitude (a), and offset (c) are usually associated with the particular trial
- Think of a guitar string: frequency and decay are functions of the string and the guitar, amplitude and phase are related to how hard you pluck and when you start

Nonlinear Least Squares

- Nonlinear least squares seek to minimize the square of the 2-norm of the residual, or difference, between a *user-defined* model, $\mathbf{y} = f(\mathbf{t}, \mathbf{x}) \in \mathbb{R}^m$ and observational data $\mathbf{b} \in \mathbb{R}^m$.

- Like linear least squares, the quantity to be minimized is

$$\phi(\mathbf{x}) := \frac{1}{2} \|\mathbf{y} - \mathbf{b}\|_2^2$$

only here, instead of the linear form $\mathbf{y} = \mathbf{A}\mathbf{x}$, we have $\mathbf{y} = f(\mathbf{t}, \mathbf{x})$ or, more explicitly, $\mathbf{y}_i = f(t_i, \mathbf{x})$, where \mathbf{x} is the vector of parameters that define f .

(Note that, in this chapter, the text uses a *different* definition of \mathbf{y} —here, I try to retain the same usage of \mathbf{y} as in Chapter 3.)

- This minimization problem fits within the framework of the multidimensional optimization problems considered so far.
- However, because of the special nature of the objective function $\phi(\mathbf{x})$, more efficient approaches such as Gauss-Newton and, particularly, Levenberg-Marquardt, can be used.

Nonlinear Least Squares

- Given data (t_i, b_i) , $i = 1, \dots, m$, find vector $\mathbf{x} \in \mathbb{R}^n$ of parameters that give “best fit” in least squares sense to model function $f(t, \mathbf{x})$, where f is nonlinear in \mathbf{x}
- Define components of *residual function*

$$r_i(\mathbf{x}) = b_i - f(t_i, \mathbf{x}), \quad i = 1, \dots, m$$

and seek to minimize 2-norm of \mathbf{r} .

- Specifically, we want to minimize the objective function

$$\phi(\mathbf{x}) := \frac{1}{2} \mathbf{r}^T(\mathbf{x}) \mathbf{r}(\mathbf{x})$$

- Gradient vector is $\nabla \phi(\mathbf{x}) = \mathbf{J}^T(\mathbf{x}) \mathbf{r}(\mathbf{x})$ and Hessian matrix is

$$\mathbf{H}_\phi = \mathbf{J}^T(\mathbf{x}) \mathbf{J}(\mathbf{x}) + \sum_{i=1}^m r_i(\mathbf{x}) \mathbf{H}_i(\mathbf{x})$$

where $\mathbf{J}(\mathbf{x})$ is Jacobian of $\mathbf{r}(\mathbf{x})$ and $\mathbf{H}_i(\mathbf{x})$ is Hessian of $r_i(\mathbf{x})$

Nonlinear Least Squares, continued

- Linear system for Newton step is

$$\mathbf{H}_\phi(\mathbf{x}_k)\mathbf{s}_k = \left(\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \sum_{i=1}^m r_i(\mathbf{x}_k)\mathbf{H}_i(\mathbf{x}_k) \right) \mathbf{s}_k = -\mathbf{J}^T(\mathbf{x}_k)\mathbf{r}(\mathbf{x}_k)$$

- m Hessian matrices \mathbf{H}_i are usually inconvenient and expensive to compute
- Moreover, in \mathbf{H}_ϕ , each \mathbf{H}_i is multiplied by residual component r_i , which is small at solution if model function is a good fit to the data

Gauss Newton Method

- These observations motivate Gauss-Newton method for nonlinear least squares, in which second-order term involving \mathbf{H}_i s is dropped and linear system

$$\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k = -\mathbf{J}^T(\mathbf{x}_k)\mathbf{r}(\mathbf{x}_k)$$

is solved for approximate Newton step, \mathbf{s}_k at each iteration

- This is a system of normal equations for linear least squares problem

$$\mathbf{J}(\mathbf{x}_k)\mathbf{s}_k \approx -\mathbf{r}(\mathbf{x}_k)$$

which can be solved better by QR factorization

- Next approximate solution is then given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$$

and process is repeated until convergence

Nonlinear Least Squares

A bit of notation...

- Recall *linear* least squares problem: *Find \mathbf{x} such that*

$$\mathbf{y} = \mathbf{A}\mathbf{x} \approx \mathbf{b}$$

for $m \times n$ matrix with $m > n$.

- The *nonlinear* least squares problem is almost the same: *Find \mathbf{x} such that*

$$\mathbf{y} = \mathbf{A}(\mathbf{x})\mathbf{x} \approx \mathbf{b}.$$

- Here, $\mathbf{A} = \mathbf{A}(\mathbf{x})$ is a function of the unknown parameters $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T$
- In many cases, cannot readily write residual evaluation as $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$.
- Instead, seek parameters x_j , $j = 1, \dots, n$ that minimize the 2-norm of the residual vector with components

$$r_i(\mathbf{x}) = b_i - f(t_i, \mathbf{x}),$$

with $f(t, \mathbf{x})$ a nonlinear function of \mathbf{x} .

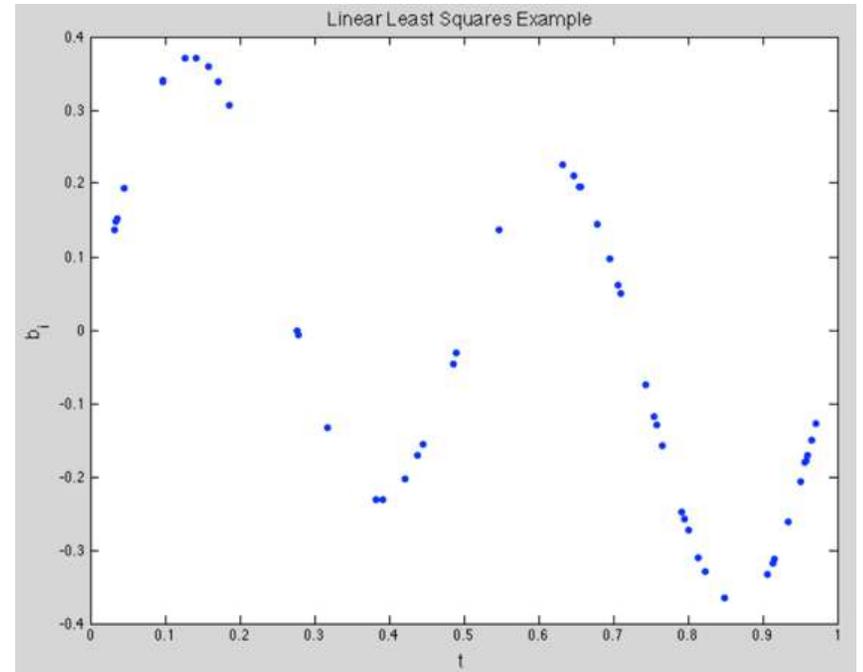
- Here, $f(t_i, \mathbf{x})$ is the *model function* which approximates the data

Example: Linear Least Squares

Model: $f(t, \mathbf{x}) = x_1 \sin 2\pi t + x_2 \sin 4\pi t.$

- This problem is *linear* in the unknown model parameters, x_1 and x_2 .
- (It is nonlinear in independent parameter, t , however.)
- In graph, the b_i are given data as a function of t_i .
- The linear least squares problem is

$$\begin{bmatrix} \sin 2\pi t_1 & \sin 4\pi t_1 \\ \sin 2\pi t_2 & \sin 4\pi t_2 \\ \vdots & \vdots \\ \sin 2\pi t_n & \sin 4\pi t_n \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \approx \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$



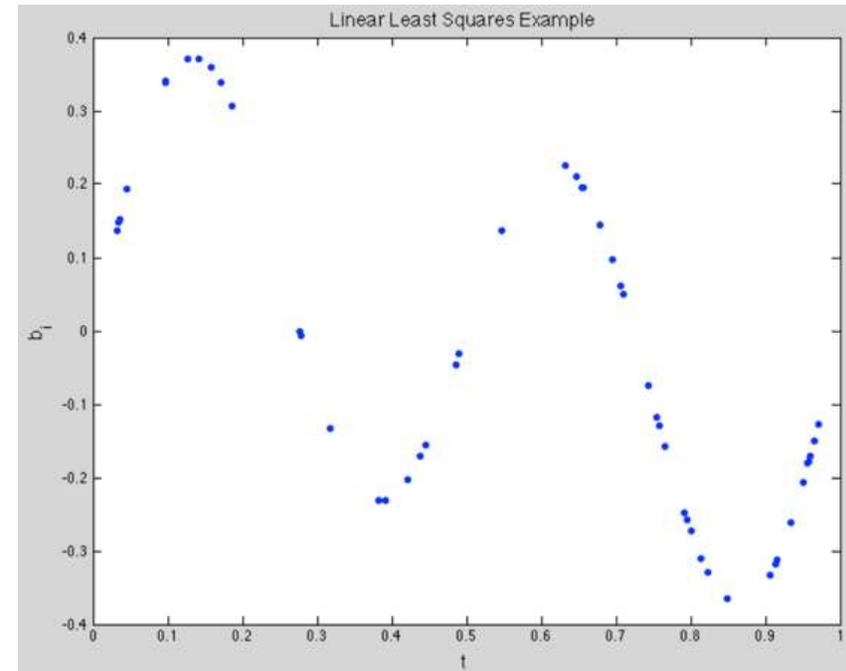
Example: Nonlinear Least Squares

Model: $f(t, \mathbf{x}) = x_1 \sin x_2 \pi t + x_3 \sin x_4 \pi t.$

- This problem is *nonlinear* in the unknown model parameters, $\mathbf{x} = [x_1 \ x_2 \ x_3 \ x_4]^T$.
- The nonlinear least squares problem is:

Find $[x_1 \ x_2 \ x_3 \ x_4]$ *that minimizes* $\|\mathbf{r}\|$, *with*

$$\mathbf{r} := \mathbf{b} - f(t_i, \mathbf{x}).$$



Nonlinear Least Squares

- $r_i = b_i - f(t_i, \mathbf{x})$ (\mathbf{x} are the unknowns.)

Example: $f(t, \mathbf{x}) = x_1 e^{\mathbf{x}_2 t}$.

- Minimization problem:

$$\phi(\mathbf{x}) = \frac{1}{2} \mathbf{r}^T \mathbf{r} = \frac{1}{2} \sum_{i=1}^m r_i^2.$$

- Stationary point:

$$\begin{aligned} \nabla \phi(\mathbf{x}) &= \frac{\partial}{\partial x_j} \phi \\ &= \frac{\partial}{\partial x_j} \left[\frac{1}{2} \sum_{i=1}^m r_i^2 \right] \\ &= \sum_{i=1}^m r_i \frac{\partial r_i}{\partial x_j} = \sum_{i=1}^m \frac{\partial r_i}{\partial x_j} r_i = \mathbf{J}^T(\mathbf{x}) \mathbf{r}(\mathbf{x}), \end{aligned}$$

with

$$J_{ij} := \frac{\partial r_i}{\partial x_j}.$$

- Hessian:

$$\begin{aligned}\mathbf{H}_\phi(\mathbf{x}) &= \frac{\partial}{\partial x_k} \nabla \phi \\ &= \frac{\partial}{\partial x_k} \left(\sum_{i=1}^m \frac{\partial r_i}{\partial x_j} r_i \right)\end{aligned}$$

$$\mathbf{H}_{\phi,jk} = \sum_{i=1}^m \frac{\partial r_i}{\partial x_j} \frac{\partial r_i}{\partial x_k} + \sum_{i=1}^m r_i \frac{\partial^2 r_i}{\partial x_j \partial x_k}$$

$$\mathbf{H}_\phi(\mathbf{x}) = \mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + \sum_{i=1}^m r_i(\mathbf{x})\mathbf{H}_i(\mathbf{x})$$

- \mathbf{H}_ϕ is the sum of SPD matrices (near \mathbf{x}^* , at least), where the \mathbf{H}_i s are the Hessians associated with each residual component, $r_i(\mathbf{x})$.
- The contributions of \mathbf{H}_i are *weighted* by the residual component, $r_i(\mathbf{x}) \rightarrow 0$ as $\mathbf{x} \rightarrow \mathbf{x}^*$.
- Since we typically spend most of the work near \mathbf{x}^* , it is convenient to neglect the \mathbf{H}_i s when considering the Hessian of ϕ .

- Newton Step:

$$\mathbf{H}_\phi \mathbf{s}_k = -\mathbf{J}_k^T \mathbf{r}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$$

- Search direction via approximate Hessian:

$$\left(\mathbf{J}_k^T \mathbf{J}_k + \sum_{i=1}^m r_i \mathbf{H}_{r_i} \right) \mathbf{s}_k = -\mathbf{J}(\mathbf{x})^T \mathbf{r}_k$$

ignore

$$\approx \left(\mathbf{J}_k^T \mathbf{J}_k \right) \mathbf{s}_k = -\mathbf{J}(\mathbf{x})^T \mathbf{r}_k$$

- Equivalent to normal equations, linear least squares:

$$\mathbf{J}_k \mathbf{s}_k \approx -\mathbf{r}_k.$$

- Solve via (reduced) QR instead of Gaussian elimination:

$$\begin{aligned} Q_1 R_1 &= \mathbf{J}_k \\ Q_1^T Q_1 R_1 \mathbf{s}_k &= -Q_1^T \mathbf{r}_k \\ \mathbf{s}_k &= -R_1^{-1} Q_1^T \mathbf{r}_k \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{s}_k. \end{aligned}$$

Gauss Newton Method, continued

- Gauss-Newton method replaces nonlinear least square (NLSQ) problem by a sequence of linear least squares (LLSQ) problems whose solutions converge to solution of original nonlinear problem
- If residual at solution is large, then second-order term omitted from Hessian is not negligible and Gauss-Newton method may converge slowly or fail to converge
- In such “large-residual” cases, it may be best to use general nonlinear minimization that takes into account the true full Hessian matrix

Levenberg-Marquardt

- Standard Gauss-Newton step is like Newton's method applied to truncated Hessian:

$$\left(\mathbf{J}_k^T \mathbf{J}_k \right) \mathbf{s}_k = -\mathbf{J}(\mathbf{x})^T \mathbf{r}_k$$

- Suffers from not being robust.
- Steepest descent would be:

$$\mathbf{s}_k = -\mathbf{J}(\mathbf{x}_k)^T \mathbf{r}_k = -\nabla \phi(\mathbf{x}_k)$$

or, with a scale factor $\frac{1}{\mu_k}$,

$$\mathbf{s}_k = -\frac{1}{\mu_k} \mathbf{J}(\mathbf{x}_k)^T \mathbf{r}_k \iff \mu_k \mathbf{s}_k = -\mathbf{J}(\mathbf{x}_k)^T \mathbf{r}_k$$

- Levenberg-Marquardt is a combination of these two:

$$\left(\mathbf{J}_k^T \mathbf{J}_k + \mu_k \mathbf{I} \right) \mathbf{s}_k = -\mathbf{J}(\mathbf{x})^T \mathbf{r}_k,$$

which, for large μ_k , corresponds to steepest descent.

- If $\mu_k \rightarrow 0$ as the iteration proceeds, recover Gauss-Newton and quadratic convergence.

Levenberg-Marquardt Method

- Levenberg-Marquardt method is another useful alternative when Gauss-Newton approximation is inadequate, yields rank-deficient LLSQ problem, or is too sensitive to initial guess \mathbf{x}_0

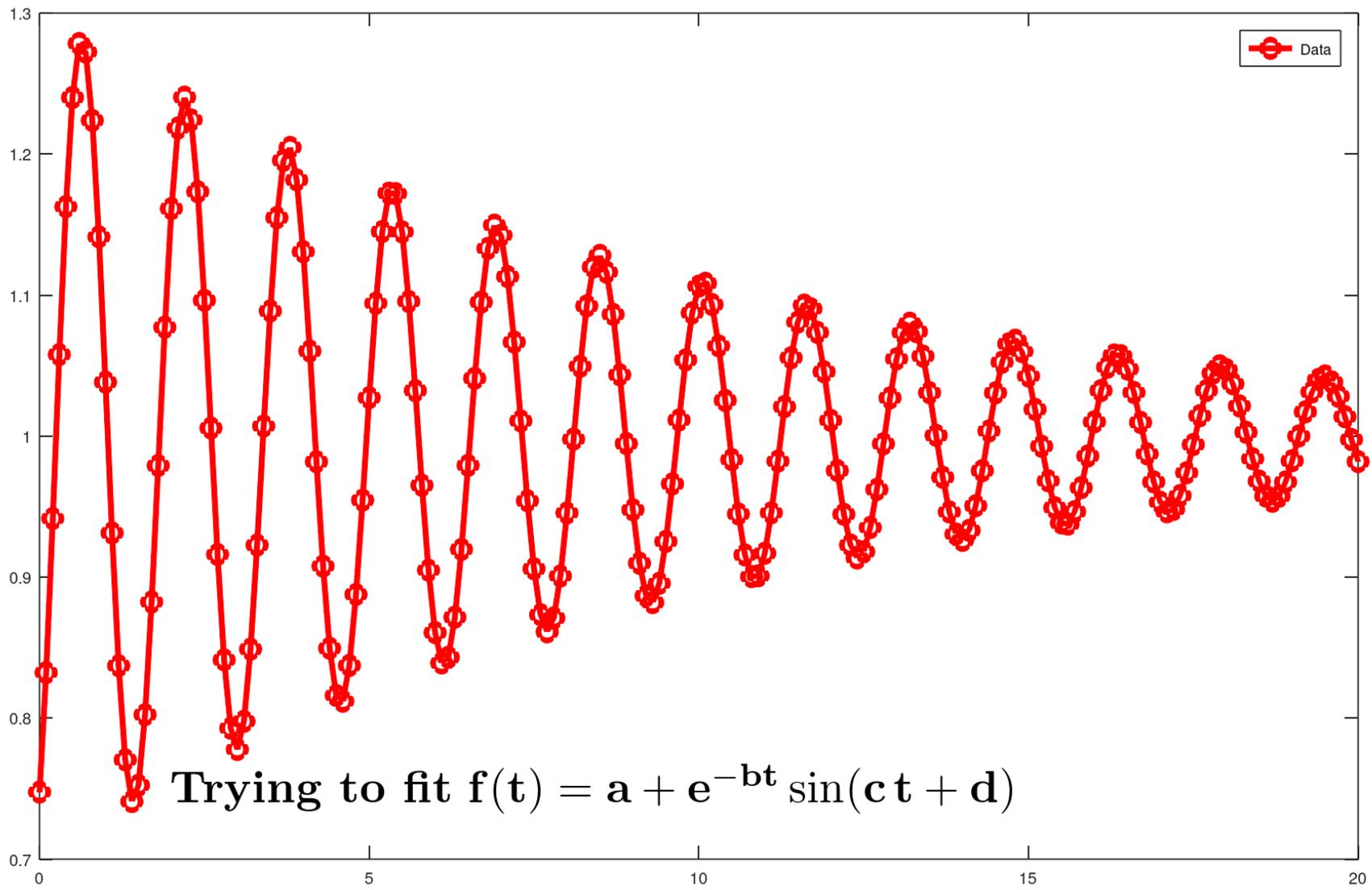
- In this method, linear system at each iteration is of form

$$(\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I})\mathbf{s}_k = -\mathbf{J}^T(\mathbf{x}_k)\mathbf{r}(\mathbf{x}_k)$$

where μ_k is a scalar parameter chosen by some strategy

- Corresponding LLSQ is
$$\begin{bmatrix} \mathbf{J}(\mathbf{x}_k) \\ \sqrt{\mu_k}\mathbf{I} \end{bmatrix} \mathbf{s}_k \approx \begin{bmatrix} -\mathbf{r}(\mathbf{x}_k) \\ \mathbf{0} \end{bmatrix}$$

- Typically, one starts with a relatively large value of μ_k and then drives it to zero as k increases
- With suitable strategy for choosing μ_k , this method can be very robust in practice and it forms the basis for several effective software packages
- Note that large μ_k implies that the update is more like a steepest-descent step than a Newton step



Example: Nonlinear Least Squares

```
a=0.2; b=.01; omega=4.0; phase=-1; c=1; %% Success
a=0.2; b=.01; omega=3.5; phase=-1; c=1; %% FAIL

x = [ a b omega phase c]'; % Initial guess

m = length(x); I=eye(m);

for k=1:50;

    [J,r] = jacobian(x); % Exact Jacobian and residual

    if gauss_newton==1; % Gauss-Newton step

        s = -J\r;

    else % Levenberg-Marquardt step

        smu = 1/sqrt(k);
        L = [ J ; smu*I ];
        r0 = [ r ; zeros(m,1) ];
        s = - L \ r0;

    end;

    x = x+s;
    f = model(x,t); plot(t,y,'ro-',lw,2,t,f,clr,lw,2);
                    legend('Data','Model'); title(ttl,fs,29);
                    disp([ k x' norm(s) norm(r) ]), pause

end;
```

levenberg.m

```
m = length(x); I=eye(m);
for k=1:50;

    % Exact Jacobian and residual
    [J,r] = jacobian(x);

    % Gauss-Newton step
    % s = -J\r;

    % Levenberg-Marquardt step

    smu = 1/sqrt(k);
    L = [ J ; smu*I ];
    r0 = [ r ; zeros(m,1) ];
    s = - L \ r0;

    x = x+s;

    f = model(x,t); plot(t,f,'b--'); pause

    [ k x' norm(s) norm(r) ], %pause

end;
```

← *this is Gauss-Newton
(commented out)*

← *this is Levenberg-
Marquardt*

Constrained Optimality

Constrained Optimality

- If problem is constrained, only *feasible* directions are relevant
- Consider equality-constrained problem

$$\min f(\mathbf{x}) \text{ subject to } \mathbf{g}(\mathbf{x}) = \mathbf{0}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, with $m \leq n$

- Necessary condition for feasible point \mathbf{x}^* to be solution is that negative gradient of f lie in space spanned by constraint normal,

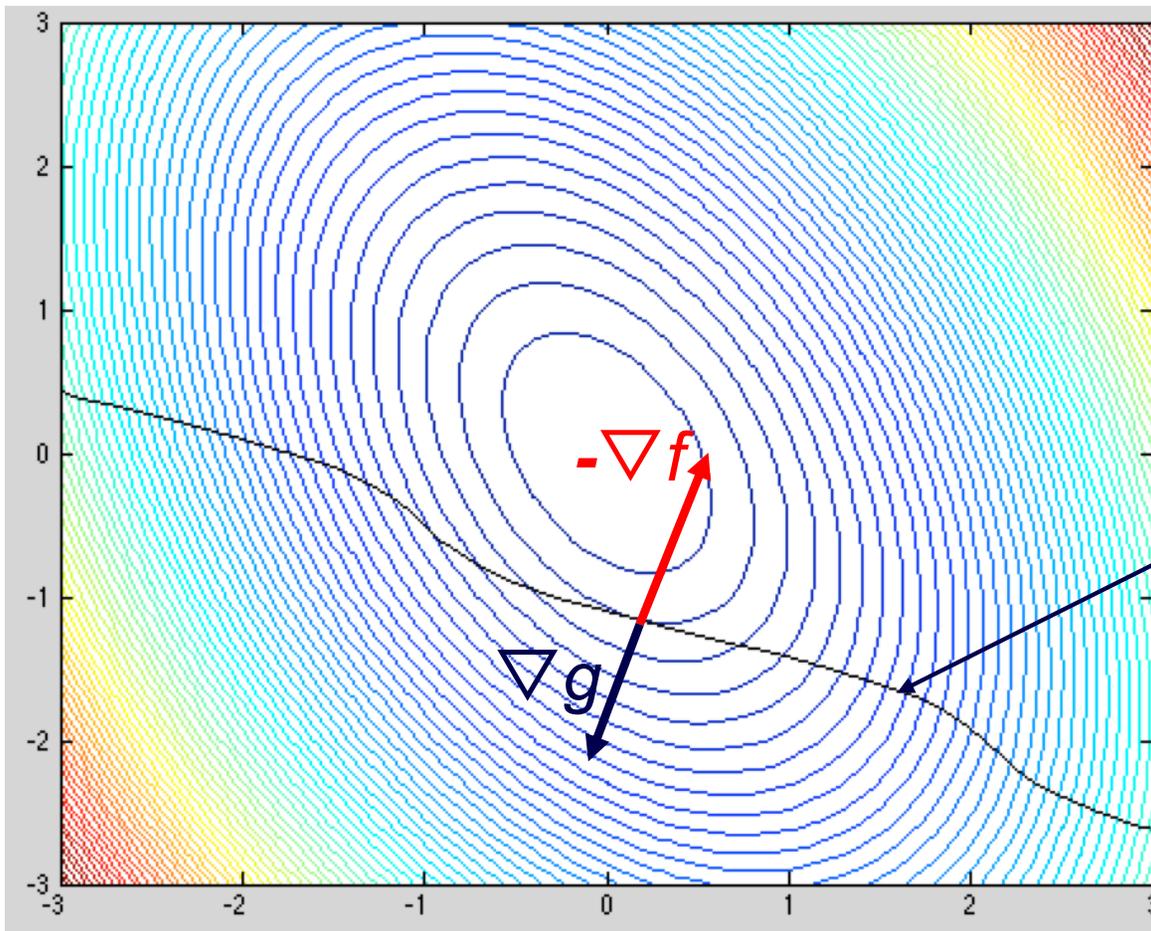
$$-\nabla f(\mathbf{x}^*) = \mathbf{J}_g^T(\mathbf{x}^*)\lambda$$

where \mathbf{J}_g is Jacobian matrix of \mathbf{g} and λ is vector of *Lagrange multipliers*

- This condition says that we cannot reduce objective function without violating the constraints

Constrained Optimality Example, $g(\mathbf{x})$ a Scalar.

- Comment on: *“This condition says we cannot reduce objective function without violating constraints.”*
- A key point is that, \mathbf{x}^* , ∇f is parallel to ∇g
- If there are multiple constraints, then ∇f in $span\{\nabla g_k\} = \mathbf{J}_g^T$.



$$-\nabla f(\mathbf{x}^*) = \mathbf{J}_g^T(\mathbf{x}^*)\lambda$$

$$-\frac{\partial f}{\partial x_i} = \sum_{k=1}^m \frac{\partial g_k}{\partial x_i} \lambda_k$$

$g(\mathbf{x}) = 0$

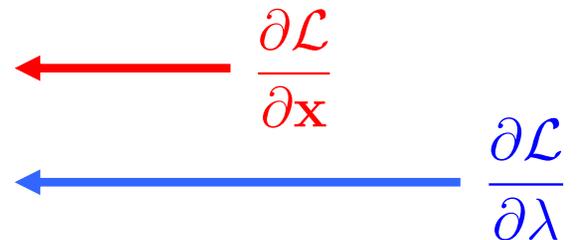
Constrained Optimality, continued

- *Lagrangian* function, $\mathcal{L} : \mathbb{R}^{n+m} \rightarrow \mathbb{R}$, is defined by

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda^T \mathbf{g}(\mathbf{x})$$

- Its gradient is given by

$$\nabla \mathcal{L}(\mathbf{x}, \lambda) = \begin{bmatrix} \nabla f(\mathbf{x}) + \mathbf{J}_g^T \lambda \\ \mathbf{g}(\mathbf{x}) \end{bmatrix}$$

A red arrow points from the expression $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$ to the top row of the gradient vector. A blue arrow points from the expression $\frac{\partial \mathcal{L}}{\partial \lambda}$ to the bottom row of the gradient vector.

- Its Hessian is given by

$$\nabla \mathbf{H}_{\mathcal{L}}(\mathbf{x}, \lambda) = \begin{bmatrix} \mathbf{B}(\mathbf{x}, \lambda) & \mathbf{J}_g^T(\mathbf{x}) \\ \mathbf{J}_g(\mathbf{x}) & \mathbf{O} \end{bmatrix}$$

*Notice that Hessian will never be SPD because of 0's on the diagonal
→ critical point is a saddle point*

where

$$\mathbf{B}(\mathbf{x}, \lambda) = \mathbf{H}_f(\mathbf{x}) + \sum_{i=1}^m \lambda_i \mathbf{H}_{g_i}(\mathbf{x})$$

Constrained Optimality, continued

- Lagrangian is designed so that the critical point satisfies necessary and feasibility conditions for constrained minimization

$$\nabla \mathcal{L}(\mathbf{x}, \lambda) = \begin{bmatrix} \nabla f(\mathbf{x}) + \mathbf{J}_g^T \lambda \\ \mathbf{g}(\mathbf{x}) \end{bmatrix} = \mathbf{0}$$

Equality Constraints: Case $g(\mathbf{x})=\text{scalar}$.

$$\min f(\mathbf{x}) \text{ subject to } g(\mathbf{x}) = 0.$$

- Lagrangian

$$\mathcal{L}(\mathbf{x}, \lambda) := f(\mathbf{x}) + \lambda g(\mathbf{x})$$

$$\lambda := \text{Lagrange multiplier}$$

$$\nabla \mathcal{L} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial x_i} \\ \frac{\partial \mathcal{L}}{\partial \lambda} \end{pmatrix} = \begin{pmatrix} \nabla f + \lambda \nabla g \\ g \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}$$

at critical point $(\mathbf{x}^*, \lambda^*)$.

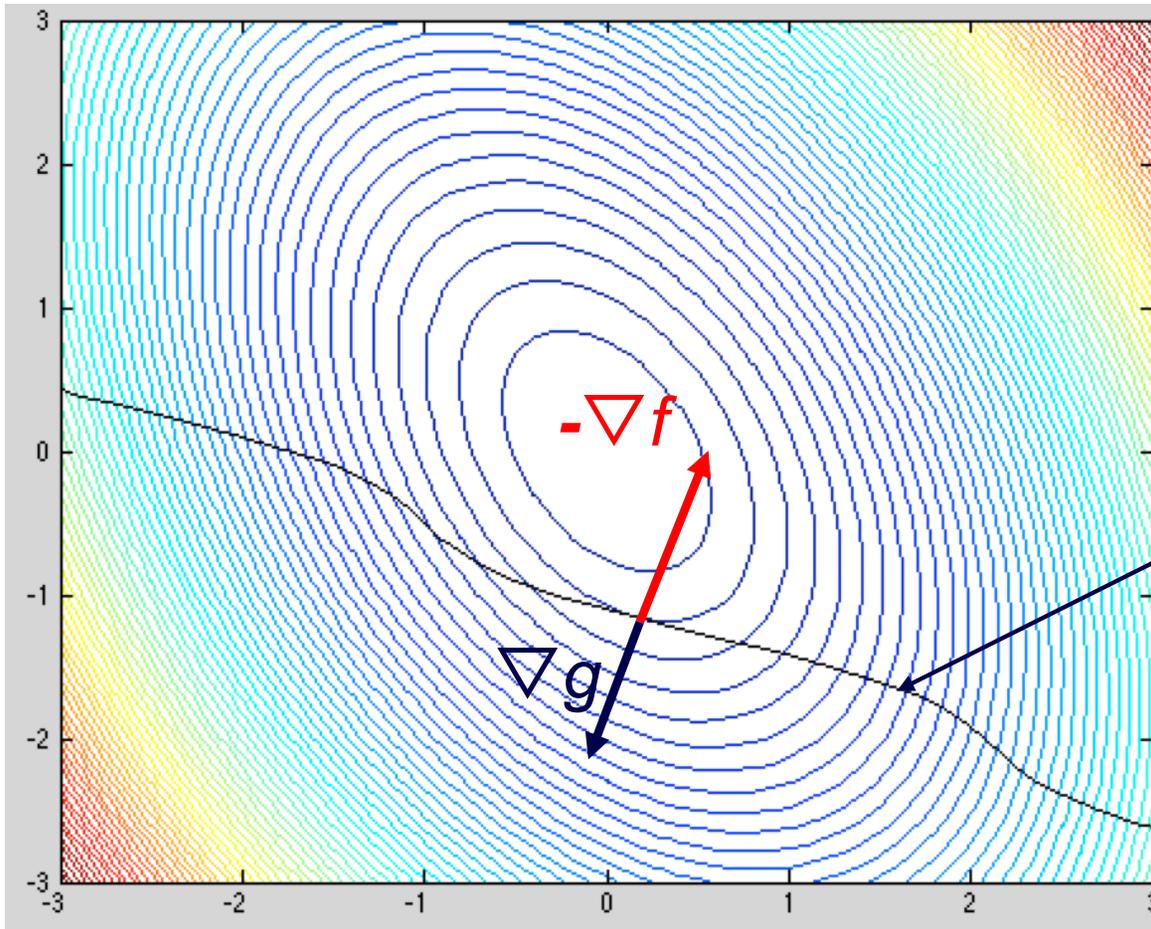
- Note, $\nabla f(\mathbf{x}^*) \neq 0$.

- Instead, $\nabla f(\mathbf{x}^*) = -\lambda^* \nabla g(\mathbf{x}^*)$.

- The gradient of f at x^* is a multiple of the gradient of g .

Constrained Optimality Example, $g(\mathbf{x})$ a Scalar.

- Comment on: *“This condition says we cannot reduce objective function without violating constraints.”*
- A key point is that, \mathbf{x}^* , ∇f is parallel to ∇g
- If there are multiple constraints, then ∇f in $span\{\nabla g_k\} = \mathbf{J}_g^T \boldsymbol{\lambda}$

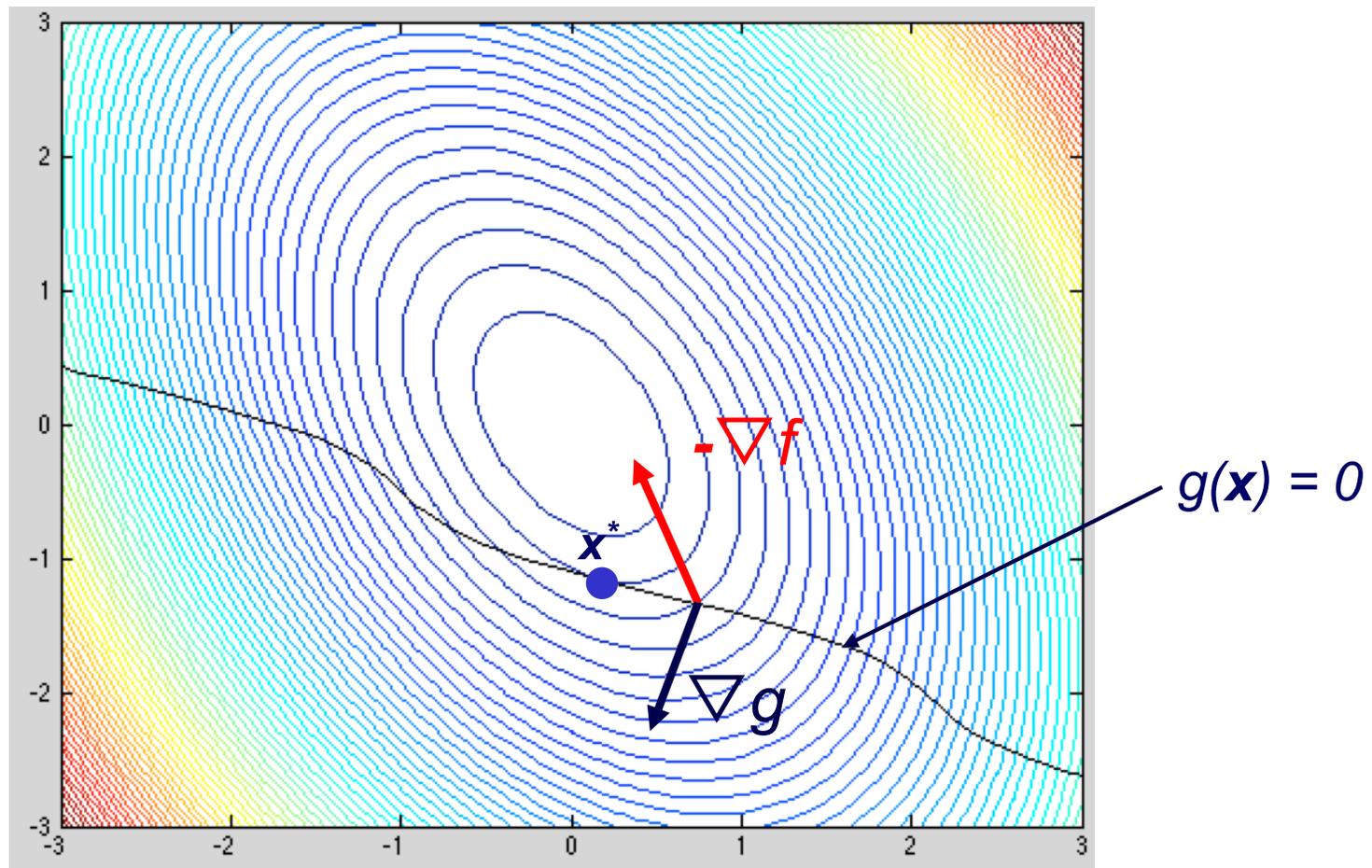


$$-\nabla f(\mathbf{x}^*) = \mathbf{J}_g^T(\mathbf{x}^*)\boldsymbol{\lambda}$$

$$-\frac{\partial f}{\partial x_i} = \sum_{k=1}^m \frac{\partial g_k}{\partial x_i} \lambda_k$$

Constrained Optimality Example, $g(\mathbf{x})$ a Scalar.

- ❑ **Here**, we see the gradients of f and g at a point that is not \mathbf{x}^* .
- ❑ Clearly, we can make more progress in reducing $f(\mathbf{x})$ by moving along the $g(\mathbf{x})=0$ contour until ∇f is parallel to ∇g .



Example: Two Equality Constraints

$\min f(\mathbf{x})$ subject to $g_1(\mathbf{x}) = 0$ and $g_2(\mathbf{x}) = 0$.

- Lagrangian

$$\mathcal{L}(\mathbf{x}, \lambda) := f(\mathbf{x}) + \lambda_1 g_1(\mathbf{x}) + \lambda_2 g_2(\mathbf{x}).$$

- First-order conditions

$$\begin{aligned} \nabla \mathcal{L} &= \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial x_i} \\ \frac{\partial \mathcal{L}}{\partial \lambda_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x_i} + \sum_k \lambda_k \frac{\partial g_k}{\partial x_i} \\ \mathbf{0} + \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} \end{bmatrix} \\ &= \begin{bmatrix} \nabla f + \mathbf{J}_g^T \lambda \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \end{aligned}$$

Constrained Optimality, continued

- Lagrangian is designed so that the critical point satisfies necessary and feasibility conditions for constrained minimization

$$\nabla \mathcal{L}(\mathbf{x}, \lambda) = \begin{bmatrix} \nabla f(\mathbf{x}) + \mathbf{J}_g^T \lambda \\ \mathbf{g}(\mathbf{x}) \end{bmatrix} = \mathbf{0}$$

- Hessian of Lagrangian is symmetric, but not positive definite, so critical point of \mathcal{L} is saddle point rather than minimum or maximum
- Critical point $(\mathbf{x}^*, \lambda^*)$ of \mathcal{L} is constrained minimum of f if $\mathbf{B}(\mathbf{x}^*, \lambda^*)$ is positive definite on *null space* of $\mathbf{J}_g(\mathbf{x}^*)$
- If columns of \mathbf{Z} form basis for null space, then test *projected* Hessian, $\mathbf{Z}^T \mathbf{B} \mathbf{Z}$, for positive definiteness
- Note: null space of $\mathbf{J}_g(\mathbf{x}^*)$ is the set of vectors tangent to the constraint surface

Solving Constrained Optimization Problems

- For equality-constrained optimization problem

$$\min f(\mathbf{x}) \text{ subject to } \mathbf{g}(\mathbf{x}) = 0,$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, with $m \leq n$, we seek *critical point of Lagrangian*

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda^T \mathbf{g}(\mathbf{x})$$

- Applying Newton's method to nonlinear system

$$\nabla \mathcal{L}(\mathbf{x}, \lambda) = \begin{bmatrix} \nabla f(\mathbf{x}) + \mathbf{J}_g^T(\mathbf{x})\lambda \\ \mathbf{g}(\mathbf{x}) \end{bmatrix}$$

we obtain linear system

$$\begin{bmatrix} \mathbf{B}(\mathbf{x}, \lambda) & \mathbf{J}_g^T(\mathbf{x}) \\ \mathbf{J}_g(\mathbf{x}) & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{s} \\ \delta \end{bmatrix} = \begin{bmatrix} \nabla f(\mathbf{x}) + \mathbf{J}_g^T(\mathbf{x})\lambda \\ \mathbf{g}(\mathbf{x}) \end{bmatrix}$$

for Newton step (\mathbf{s}, δ) in (\mathbf{x}, λ) at each iteration, where \mathbf{B} is the matrix introduced earlier involving the Hessians

Example: Constrained Optimization

- Consider quadratic objective function

$$\min_{\mathbf{x}} f(\mathbf{x}) = 0.5 x_1^2 + 2.5 x_2^2$$

subject to

$$g(\mathbf{x}) = x_1 - x_2 - 1 = 0$$

- Lagrangian function is given by

$$\mathcal{L}(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}) = 0.5 x_1^2 + 2.5 x_2^2 + \lambda(x_1 - x_2 - 1)$$

- Since

$$\nabla f(\mathbf{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix} \quad \text{and} \quad \mathbf{J}_g(\mathbf{x}) = [1 \quad -1]$$

we have

$$\nabla_x \mathcal{L}(\mathbf{x}, \lambda) = \nabla f(\mathbf{x}) + \mathbf{J}_g^T \lambda = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix} + \lambda \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Example, continued

- System to be solved for critical point of Lagrangian is

$$x_1 + \lambda = 0$$

$$5x_2 - \lambda = 0$$

$$x_1 - x_2 = 1$$

which in this case is linear system

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 5 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- Solving this system, we obtain

$$x_1 = 0.833, \quad x_2 = -0.167, \quad \lambda = -0.833$$

Example from Text

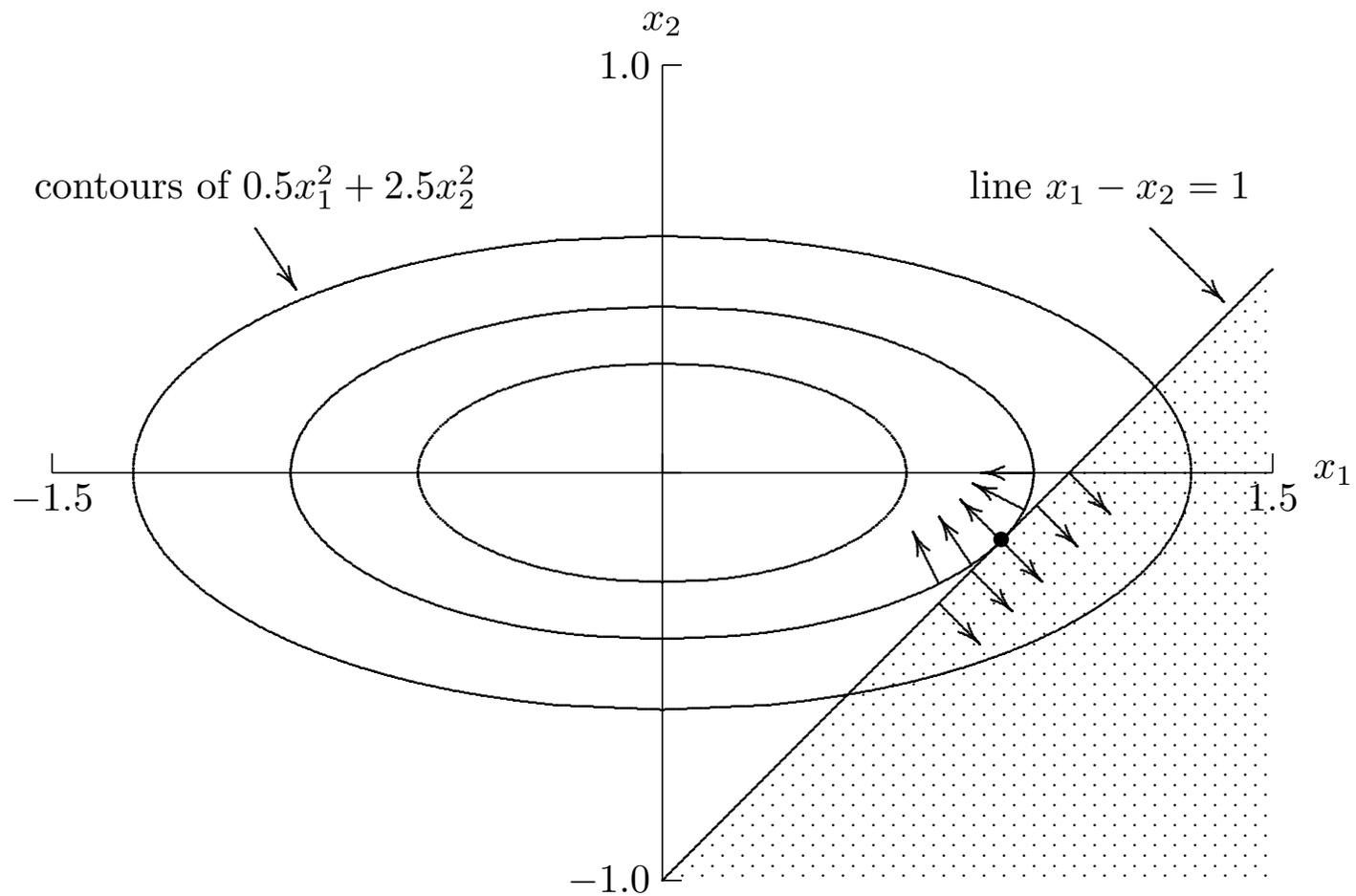


Figure 6.5: Solution to constrained optimization problem. Feasible region is shaded.

Weighted Mean Example

- Suppose we have a set of data $\tilde{\mathbf{x}} = [\tilde{x}_1 \cdots \tilde{x}_n]^T$ and we want to find a nearby set,

$$\mathbf{x} = [x_1 \cdots x_n]^T \approx \tilde{\mathbf{x}},$$

such that the *weighted-average*,

$$g(\mathbf{x}) := \frac{\sum_{i=1}^n w_i \mathbf{x}_i}{\sum_{i=1}^n w_i} = \frac{\mathbf{w}^T \mathbf{x}}{\mathbf{w}^T \mathbf{e}} = 0.$$

- We will assume that $w_i > 0$ and define the W -norm as

$$\|\mathbf{x}\|_W^2 = \mathbf{x}^T W \mathbf{x},$$

where $W = \text{diag}(w_i)$ is an SPD matrix.

- For our approximation, we elect to minimize in the W -norm:

$$\min_{\mathbf{x} \text{ s.t. } g(\mathbf{x})=0} \|\mathbf{x} - \tilde{\mathbf{x}}\|_W^2$$

Weighted Mean Example

Constraint, $g(\mathbf{x})=0$

- The Lagrangian for this case is

$$\begin{aligned}\mathcal{L}(\mathbf{x}, \lambda) &= \sum w_i (x_i - \tilde{x}_i)^2 + \lambda \sum w_i x_i \\ &= (\mathbf{x} - \tilde{\mathbf{x}})^T W (\mathbf{x} - \tilde{\mathbf{x}}) + \lambda \mathbf{w}^T \mathbf{x}.\end{aligned}$$

- The first-order condition on \mathcal{L} leads to

$$J_{\mathcal{L}} = \begin{bmatrix} 2W & \mathbf{w} \\ \mathbf{w}^T & 0 \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} - \begin{bmatrix} 2W & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} \tilde{\mathbf{x}} \\ 0 \end{pmatrix} = 0,$$

or

$$\begin{bmatrix} 2W & \mathbf{w} \\ \mathbf{w}^T & 0 \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} = \begin{bmatrix} 2W & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} \tilde{\mathbf{x}} \\ 0 \end{pmatrix}.$$

Weighted Mean Example

$$\begin{bmatrix} 2W & \mathbf{w} \\ \mathbf{w}^T & 0 \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} = \begin{bmatrix} 2W & 0 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} \tilde{\mathbf{x}} \\ 0 \end{pmatrix}.$$

- Block Gaussian elimination for λ :

$$\begin{bmatrix} 2W & \mathbf{w} \\ 0 & -S \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} 2W\tilde{\mathbf{x}} \\ -\mathbf{w}^T\tilde{\mathbf{x}} \end{pmatrix}$$

- Here, S is the (1×1) Schur-complement matrix,

$$S = \frac{1}{2}\mathbf{w}^T W^{-1}\mathbf{w} = \frac{1}{2}\sum w_i = \frac{1}{2}\mathbf{w}^T \mathbf{e}.$$

with $\mathbf{e} = [1 \ 1 \ \dots \ 1]^T$

- Solve for λ :

$$\lambda = \mathbf{S}^{-1}\mathbf{w}^T\tilde{\mathbf{x}} = 2\frac{\mathbf{w}^T\tilde{\mathbf{x}}}{\mathbf{w}^T\mathbf{e}} = 2g(\tilde{\mathbf{x}})$$

Weighted Mean Example

- Solve for \mathbf{x} :
$$\begin{aligned}\mathbf{x} &= (2W)^{-1} \left(2W\tilde{\mathbf{x}} - 2\frac{\mathbf{w}^T\tilde{\mathbf{x}}}{\mathbf{w}^T\mathbf{e}}\mathbf{w} \right) \\ &= \tilde{\mathbf{x}} - \frac{\mathbf{w}^T\tilde{\mathbf{x}}}{\mathbf{w}^T\mathbf{e}}W^{-1}\mathbf{w} \\ &= \tilde{\mathbf{x}} - \frac{\mathbf{w}^T\tilde{\mathbf{x}}}{\mathbf{w}^T\mathbf{e}}\mathbf{e} \\ &= \tilde{\mathbf{x}} - g(\tilde{\mathbf{x}})\mathbf{e}.\end{aligned}$$

- So, we have simply:

$$x_i = \tilde{x}_i - \text{constant},$$

where the constant is the weighted average of the \tilde{x}_i s.

- While this is the expected result, it's nice to be able to cast it into a constrained minimization problem because it allows us to assert that it is the *best possible* result, in the given measure, subject to the constraints.

Penalty Methods

- Merit function can be used to convert equality-constrained problem into a sequence of unconstrained problems
- If \mathbf{x}_ρ^* is solution to

$$\min_{\mathbf{x}} \phi_\rho(\mathbf{x}) = f(\mathbf{x}) + \frac{1}{2}\rho\mathbf{g}(\mathbf{x})^T\mathbf{g}(\mathbf{x})$$

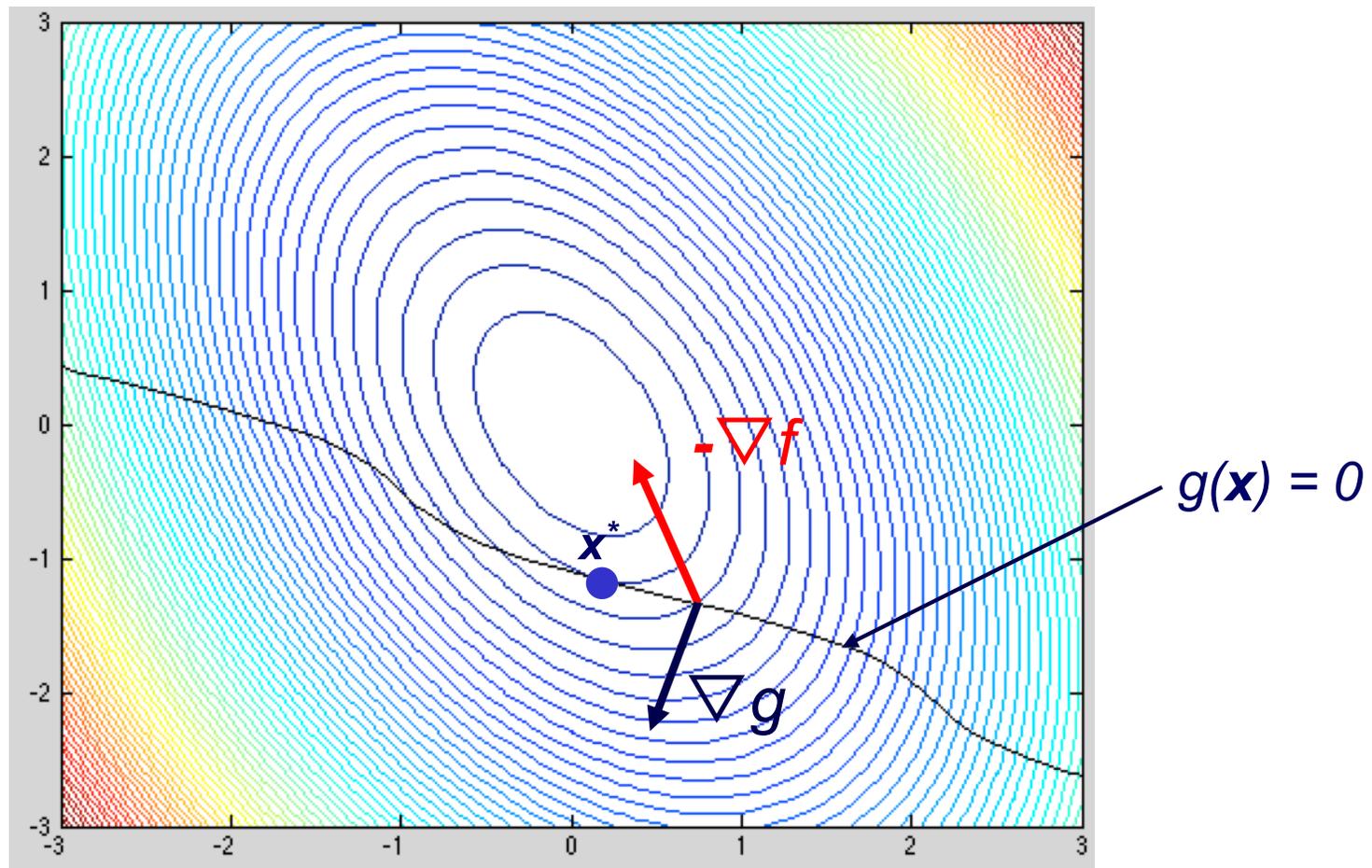
then under appropriate conditions,

$$\lim_{\rho \rightarrow \infty} \mathbf{x}_\rho^* = \mathbf{x}^*$$

- This enables use of unconstrained optimization methods, but problem becomes ill-conditioned for large ρ , so we solve a sequence of problems with gradually increasing values of ρ
- Minimum of each problem is used as the starting point for the next problem
- With this approach, for large ρ , \mathbf{x} will tend to be on the constraint surface and will tend towards the minimum of objective function f , subject to the constraint

Constrained Optimality Example, $g(\mathbf{x})$ a Scalar.

- **Here**, we see the gradients of f and g at a point that is not \mathbf{x}^* .
- Clearly, we can make more progress in reducing $f(\mathbf{x})$ by moving along the $g(\mathbf{x})=0$ contour until ∇f is parallel to ∇g .



Barrier Methods

- Recall optimization problem with *inequality constraints*,

$$\min f(\mathbf{x}) \text{ subject to } \mathbf{h}(\mathbf{x}) \leq 0$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $\mathbf{h} : \mathbb{R}^n \rightarrow \mathbb{R}^p$

- For inequality-constrained problems, alternative is *barrier function*, such as

$$\phi_\mu(\mathbf{x}) = f(\mathbf{x}) - \mu \sum_{i=1}^p \frac{1}{h_i(\mathbf{x})}$$

or

$$\phi_\mu(\mathbf{x}) = f(\mathbf{x}) - \mu \sum_{i=1}^p \log(-h_i(\mathbf{x}))$$

- Again, solutions of unconstrained problem approach \mathbf{x}^* as $\mu \rightarrow 0$, but problems are increasingly ill-conditioned, so solve sequence of problems with decreasing μ
- Barrier functions are basis for *interior point* methods

Barrier Functions for Inequality Constraints

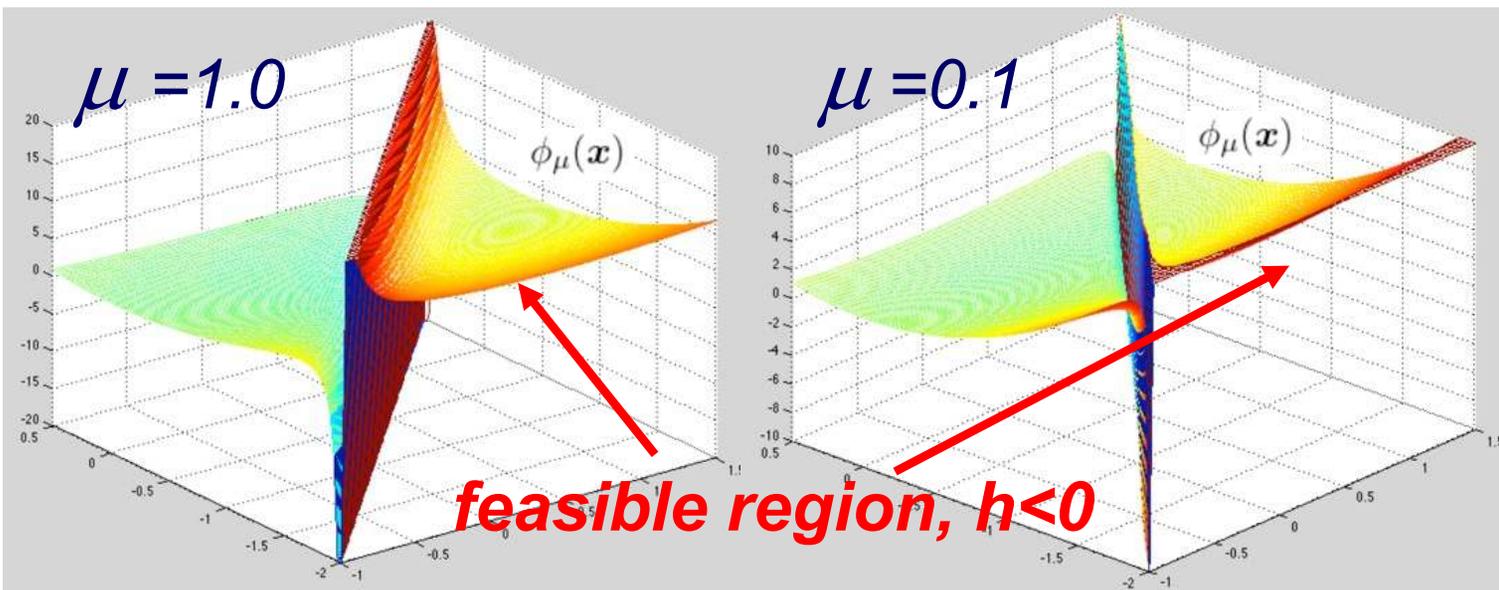
- Solve modified unconstrained minimization problem,

$$\phi_{\mu}(\mathbf{x}) = f(\mathbf{x}) - \mu \sum_{i=1}^p \frac{1}{h_i(\mathbf{x})}$$

- **Starting in the feasible region!**

- As $\mu \rightarrow 0$ it appears that the constraint is weaker.

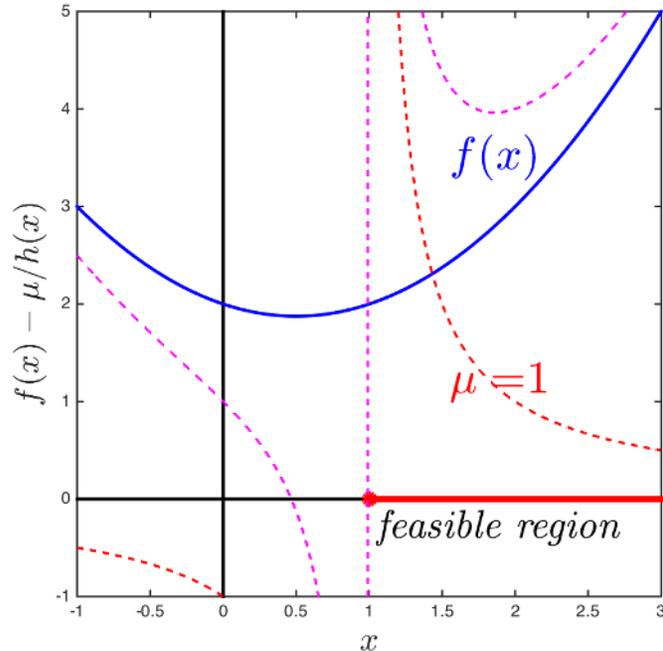
- However, the constraint goes to infinity for any $\mu \neq 0$, so a small μ simply means the fence is *steeper*, which makes the minimization problem more challenging.



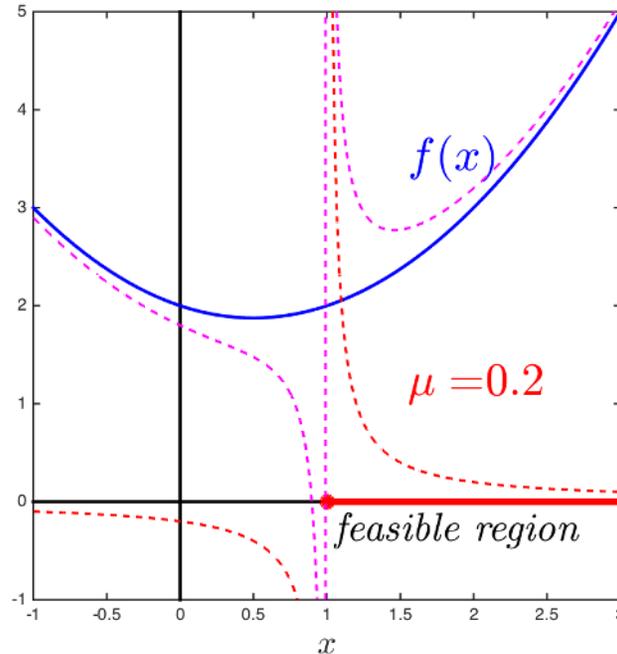
$$f(\mathbf{x}) = \frac{1}{2}x_1^2 + \frac{5}{2}x_2^2$$
$$h(\mathbf{x}) = 1 - x_1 + x_2 < 0$$

1D Barrier Functions

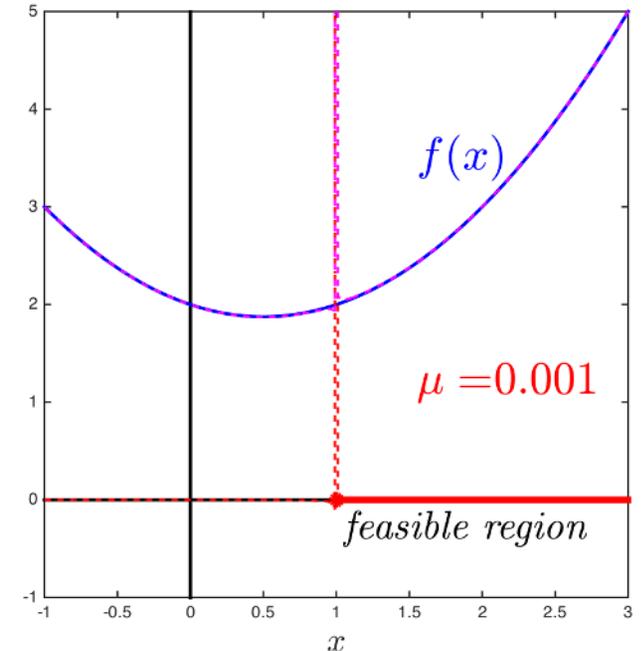
1D Barrier Function



1D Barrier Function



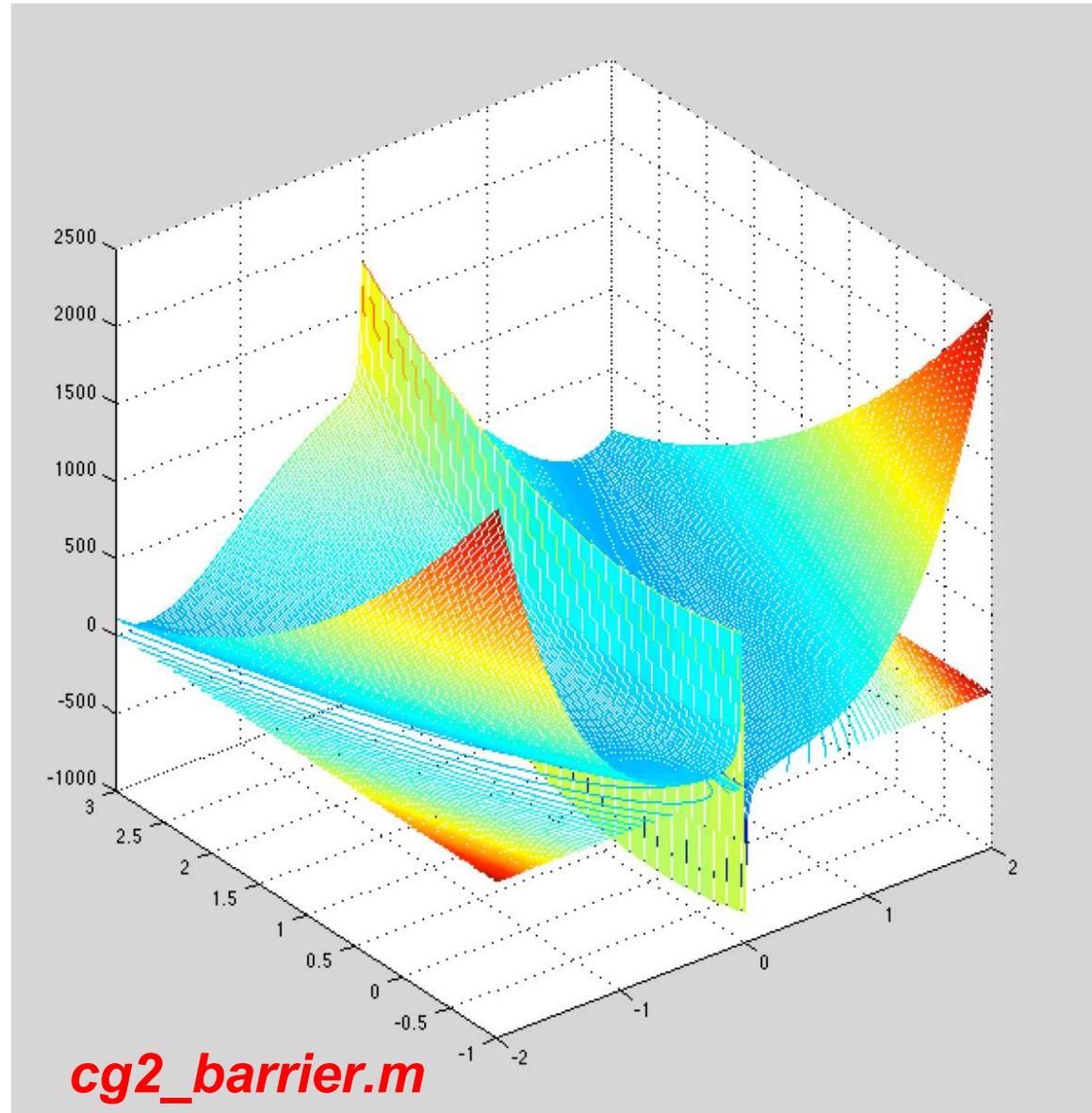
1D Barrier Function



- ❑ As $\mu \rightarrow 0$ it appears that the constraint is weaker.
- ❑ The constraint goes to infinity for any $\mu \neq 0$, so a small μ simply means the fence is *steeper*, which makes the minimization problem more challenging.

Barrier Demo

- ❑ *With barrier methods, simply augment the original (unconstrained) objective function, $f(\mathbf{x})$, with the penalty function.*
- ❑ *Then, start in the feasible region.*
- ❑ **Careful** that iteration doesn't jump over the barrier—particularly Newton's method.
- ❑ **Careful line search...**



Optimization Summary

- 1-D and n -D minimization, with and without constraints
- Sensitivity is greater than for root-finding ($\|\mathbf{x} - \mathbf{x}^*\| = O(\sqrt{\epsilon_M})$)
- 1-D: bracketing and successive parabolic interpolation (no gradients needed)
- n -D: Newton and quasi-Newton (e.g., BFGS, CG)
 - Typically require line search (i.e., 1-D minimization along \mathbf{s})
- Nonlinear least squares: Gauss-Newton and Levenberg-Marquardt
- Constrained optimization: Lagrangian and Lagrange multipliers, barrier methods