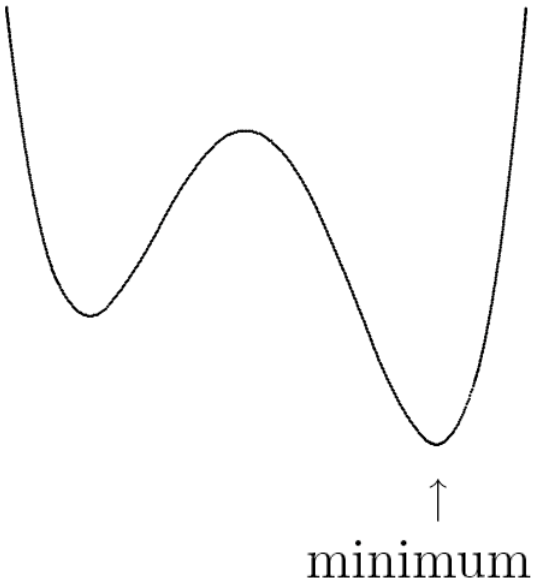
Outline



Optimization

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

Optimization Problems

- General continuous optimization problem:

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

  where $f\colon \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{g}\colon \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{h}\colon \mathbb{R}^n \to \mathbb{R}^p$

- *Linear programming*: f, $\boldsymbol{g}$, and $\boldsymbol{h}$ are all linear

- *Nonlinear programming*: at least one of f, $\boldsymbol{g}$, and $\boldsymbol{h}$ 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:

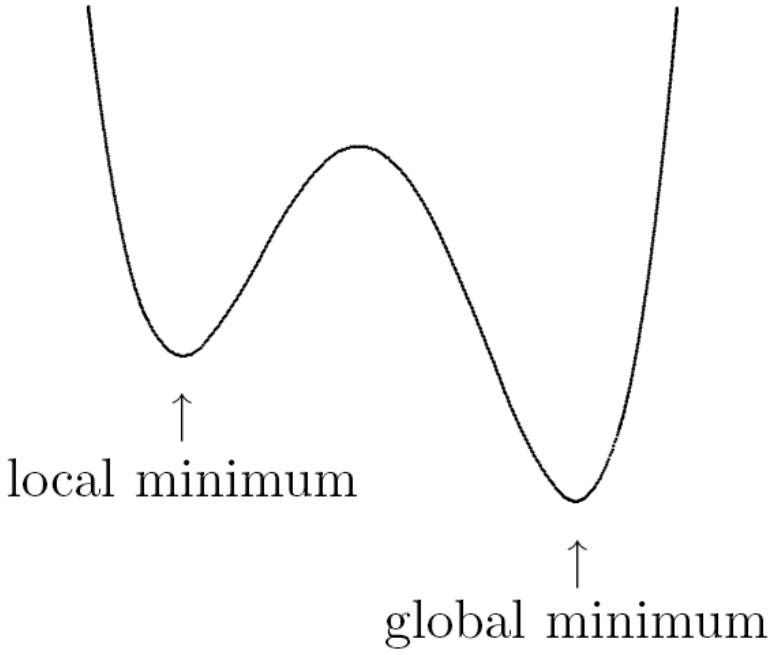
$$\min_{x_1,x_2} f(x_1, x_2) = 2\pi x_1(x_1 + x_2)$$

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

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

Local vs Global Optimization

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



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
- For some problems, such as linear programming, global optimization is more tractable

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

$$\lim_{\|\boldsymbol{x}\| \to \infty} f(\boldsymbol{x}) = +\infty$$

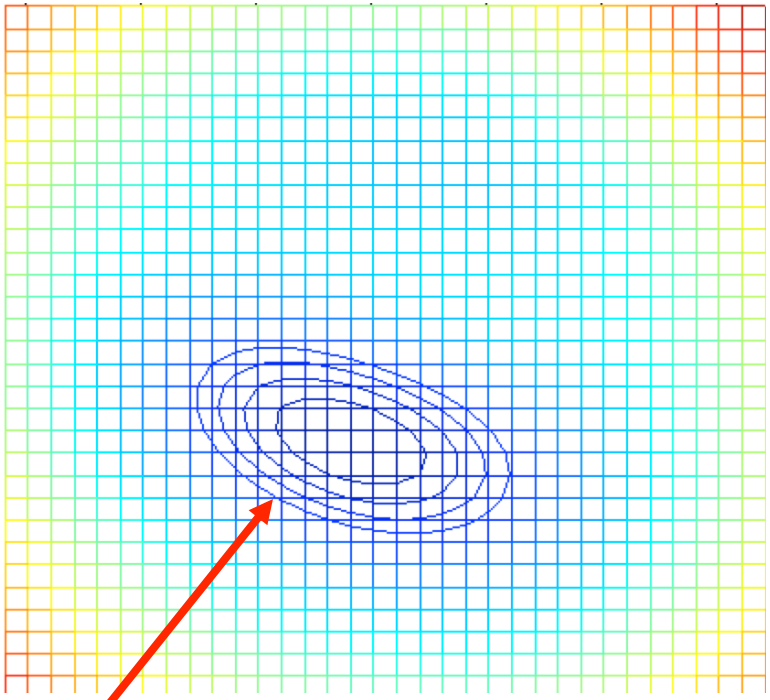
  i.e., $f(\boldsymbol{x})$ must be large whenever $\|\boldsymbol{x}\|$ is large
- If f is coercive on closed, unbounded set $S \subseteq \mathbb{R}^n$, then f has global minimum on S

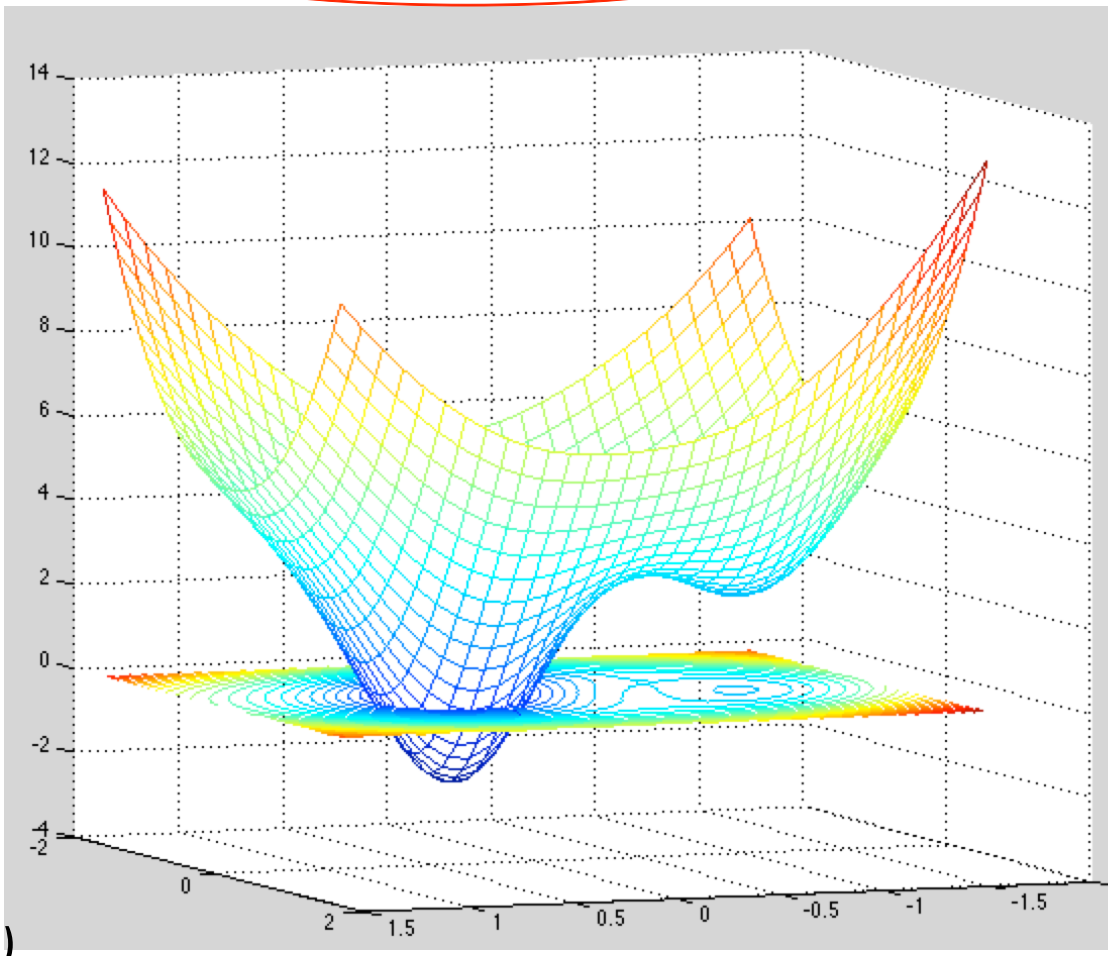
An Example of a Coercive Function

```
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 $||\boldsymbol{x}|| \rightarrow \infty$

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





Level Sets (contours, isosurfaces)

An Example of a Non-Coercive Function

- What's wrong with this case?

$$f(\mathbf{x}) = (x_1 - x_2)^2$$

noncoerce.m

Level Sets

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

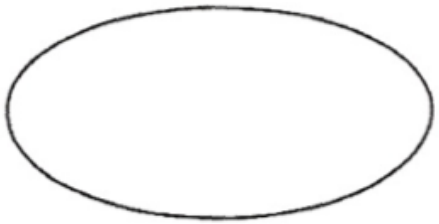
$$L_\gamma = \{\boldsymbol{x} \in S: \ f(\boldsymbol{x}) \leq \gamma\}$$

- 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
- If S is unbounded, then f is coercive on S if, and only if, *all* of its sublevel sets are bounded *(No open contours…)*

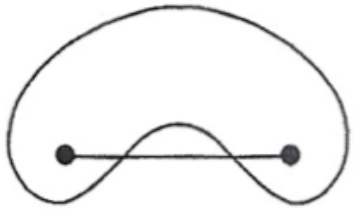
Uniqueness of Minimum

- Set $S \subseteq \mathbb{R}^n$ is *convex* if it contains 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

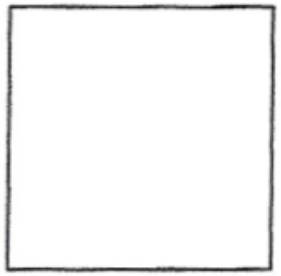
Convex Domains and Convex Functions



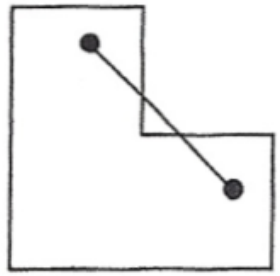
convex



nonconvex



convex



nonconvex

nonconvex

convex

strictly convex

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

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

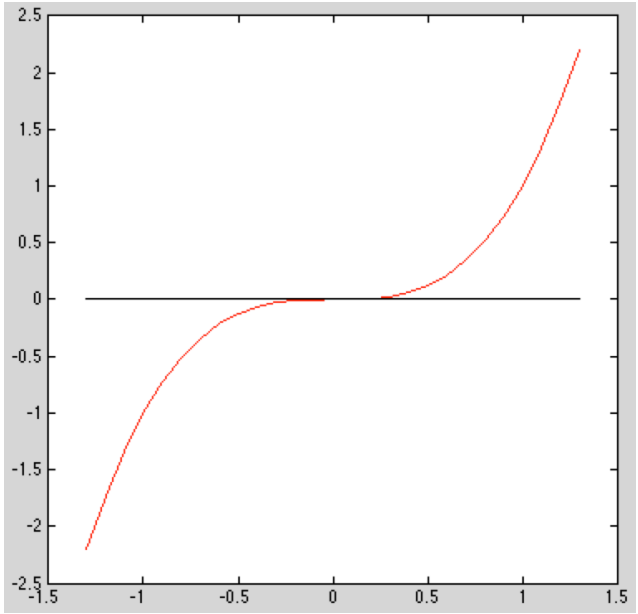
where $\nabla f(\boldsymbol{x})$ is *gradient* vector of f, whose ith component is $\partial f(\boldsymbol{x})/\partial x_i$

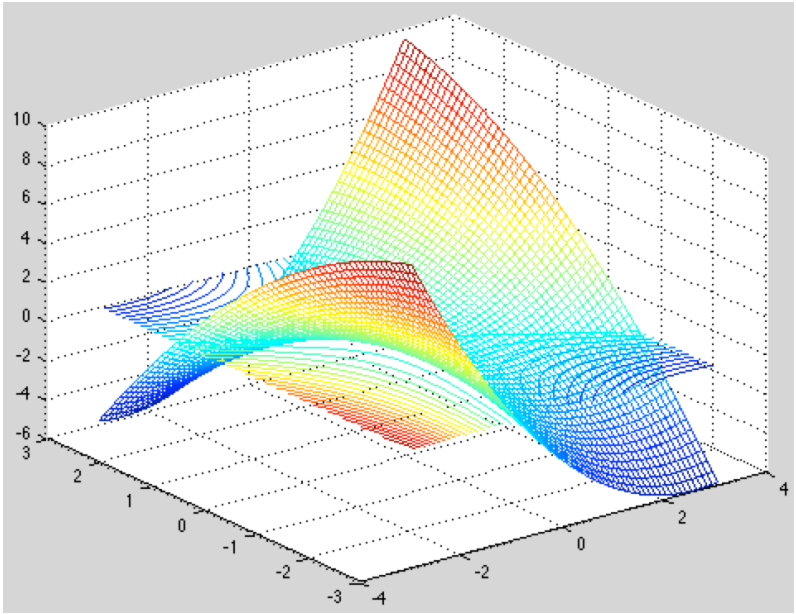
- For continuously differentiable $f : S \subseteq \mathbb{R}^n \to \mathbb{R}$, any interior point $\boldsymbol{x}^*$ of S at which f has local minimum must be critical point of f
- But not all critical points are minima: they can also be maxima or saddle points

First-Order Optimality Condition

- But 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

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

- ***We use the solution techniques of Chapter 5 to solve this nonlinear system.***

Second-Order Optimality Condition

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

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

  which is symmetric

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

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:

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

- Hessian is symmetric:

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

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

$$\begin{aligned} \mathbf{x}^* &= [1, -1]^T, \text{ and} \\ \mathbf{x}^* &= [2, -3]^T. \end{aligned}$$

- At first critical point, $x^* = [1, -1]^T$,

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

  does *not* have positive eigenvalues, so $[1, -1]^T$ is a saddle point.

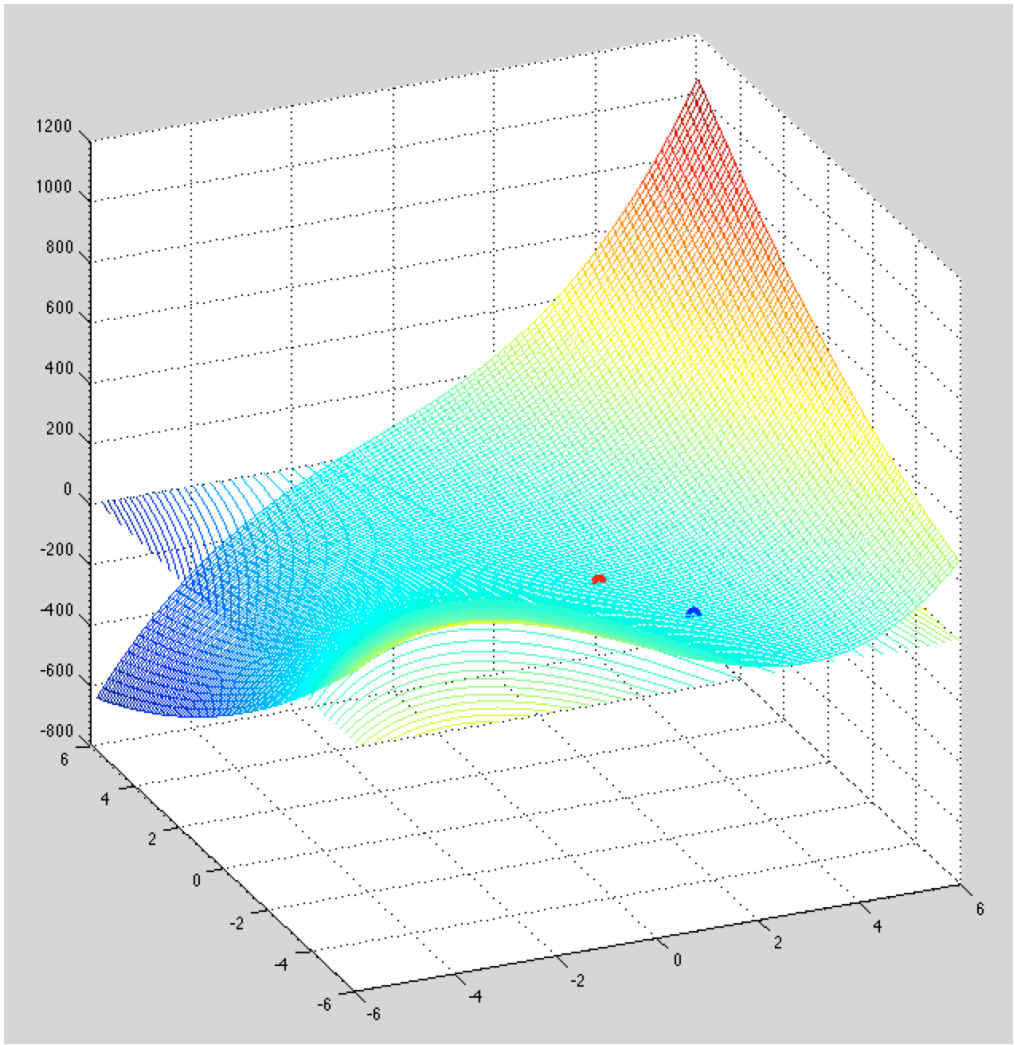
- At second critical point, $x^* = [2, -3]^T$,

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

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

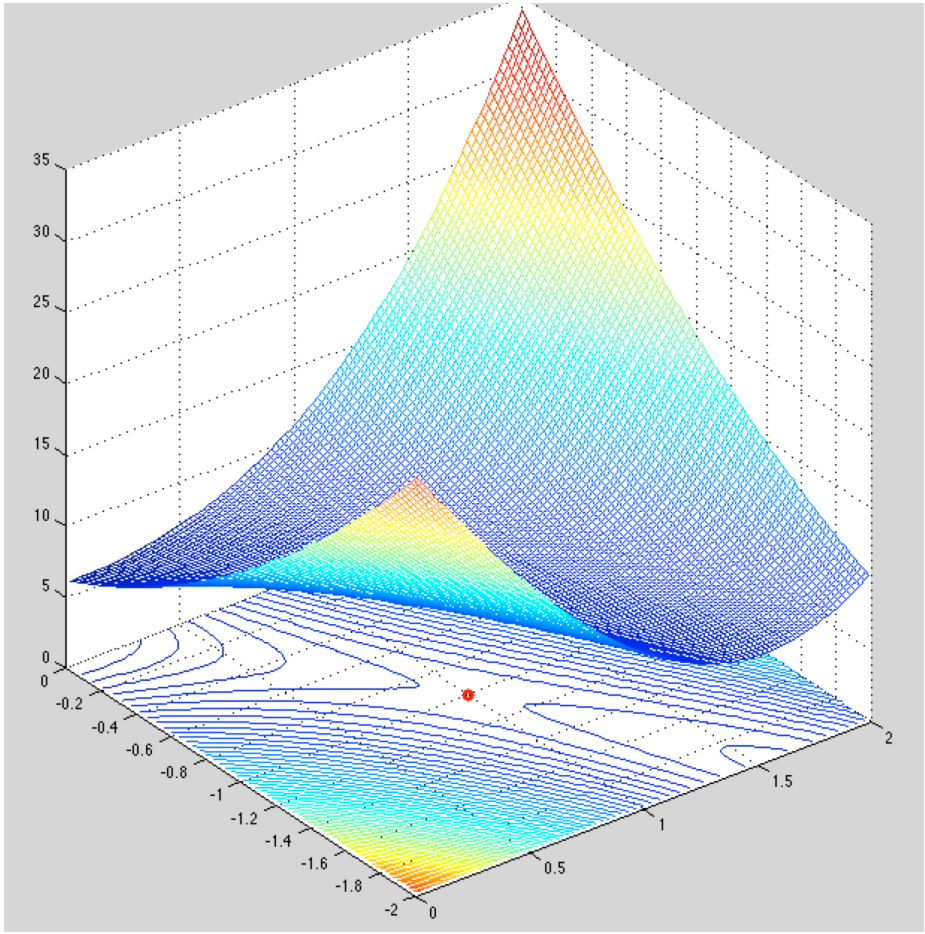
Example 6.5 from text

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



Example 6.5 from text

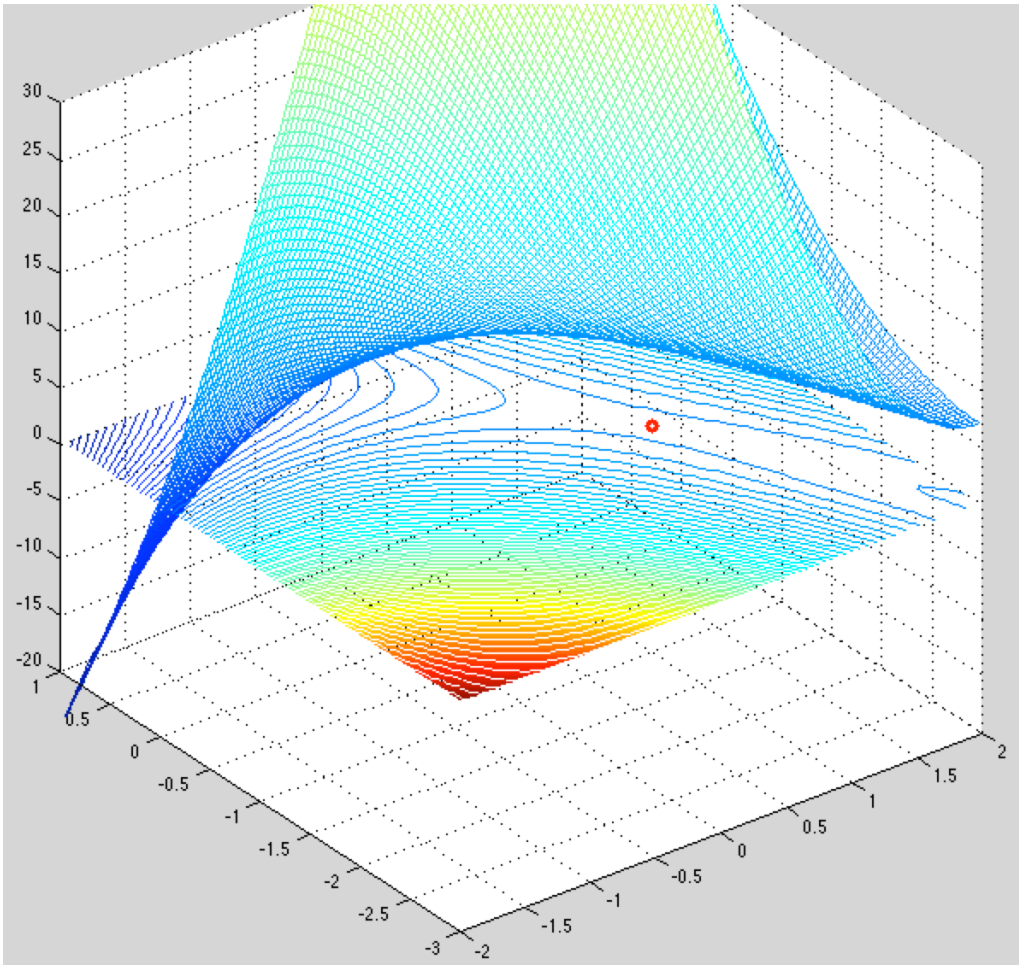
$$f(\mathbf{x}) \;=\; 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6.$$



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

Example 6.5 from text

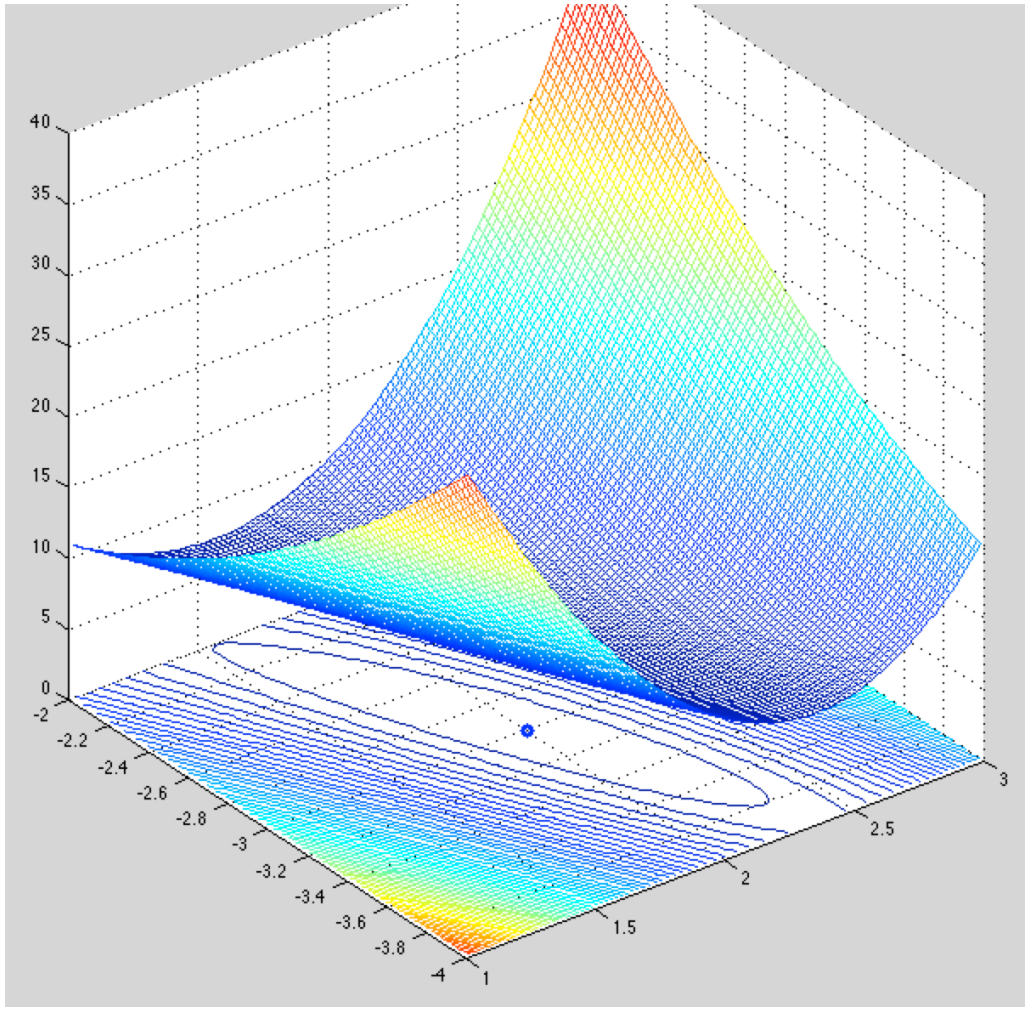
$$f(\mathbf{x}) \;=\; 2x_1^3 + 3x_1^2 + 12x_1x_2 + 3x_2^2 - 6x_2 + 6.$$



- 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 $f : S \subseteq \mathbb{R}^n \to \mathbb{R}$, we can distinguish among critical points by considering *Hessian matrix* $\boldsymbol{H}_f(x)$ defined by

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

  which is symmetric

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

Example of Singular Hessian in 1D

- Consider,

$$\begin{aligned} f(x) &= (x-1)^4 \\ \nabla f &= \frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial x} = 4(x-1)^3 = 0 \text{ when } x = x^* = 1. \\ H_{ij} &= \frac{\partial^2 f}{\partial x_i x_j} = \frac{\partial^2 f}{\partial x^2} = 12(x-1)^2 = 0. \end{aligned}$$

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

- However, if

$$\begin{aligned} f(x) &= (x-1)^3 \\ \nabla f &= \frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial x} = 3(x-1)^2 = 0 \text{ when } x = x^* = 1. \\ H_{ij} &= \frac{\partial^2 f}{\partial x_i x_j} = \frac{\partial^2 f}{\partial x^2} = 6(x-1) = 0. \end{aligned}$$

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

Constrained Optimality

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

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

where $f\colon \mathbb{R}^n \to \mathbb{R}$ and $\boldsymbol{g}\colon \mathbb{R}^n \to \mathbb{R}^m$, with $m \leq n$, necessary condition for feasible point $\boldsymbol{x}^*$ to be solution is that negative gradient of f lie in space spanned by constraint normals,

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

where $\boldsymbol{J}_g$ is Jacobian matrix of $\boldsymbol{g}$, and $\boldsymbol{\lambda}$ is vector of *Lagrange multipliers*

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

Constrained Optimality, continued

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

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

- Its gradient is given by

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

- Its Hessian is given by

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

  where

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

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

- Equality constraint:

$$\mathbf{g}(\mathbf{x}) \;=\; \begin{pmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_m(\mathbf{x}) \end{pmatrix} \;=\; \mathbf{0}.$$

- Lagrangian: $\mathcal{L}$:

$$\mathcal{L} \;=\; f(\mathbf{x}) \;+\; \mathbf{g}(\mathbf{x})^T \underline{\lambda} \;=\; f(\mathbf{x}) \;+\; \sum_{i=1}^{m} g_i(\mathbf{x})^T \lambda_i.$$

- Gradient:

$$\nabla \mathcal{L} \;=\; \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial x_1} \\ \frac{\partial \mathcal{L}}{\partial x_2} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial x_n} \\ \frac{\partial \mathcal{L}}{\partial \lambda_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \lambda_m} \end{pmatrix} \;=\; \begin{pmatrix} \nabla f \;+\; \sum_{i=1}^{m} \frac{\partial g_i}{\partial x_j} \lambda_i \\ g_1 \\ g_2 \\ \vdots \\ g_m \end{pmatrix} \;=\; \begin{pmatrix} \nabla f \;+\; J_g^T \underline{\lambda} \\ \mathbf{g}(\mathbf{x}) \end{pmatrix}.$$

- Hessian:

$$\nabla \mathbf{H}_{\mathcal{L}}(\mathbf{x}, \underline{\lambda}) \;=\; \begin{pmatrix} \mathbf{H}_f \;+\; \sum_{i=1}^{m} \mathbf{H}_{g_i} \lambda_i & J_g^T \\ J_g & 0 \end{pmatrix} \;=\; \begin{pmatrix} \mathbf{B}(\mathbf{x}, \underline{\lambda}) & J_g^T \\ J_g & 0 \end{pmatrix}$$

Constrained Optimality, continued

- Together, necessary condition and feasibility imply critical point of Lagrangian function,

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

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

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

- Lagrangian

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \lambda) \;&:=\; f(\mathbf{x}) \;+\; \lambda\, g(\mathbf{x}) \\ \lambda \;&:=\; \textit{Lagrange multiplier} \end{aligned}$$

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

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

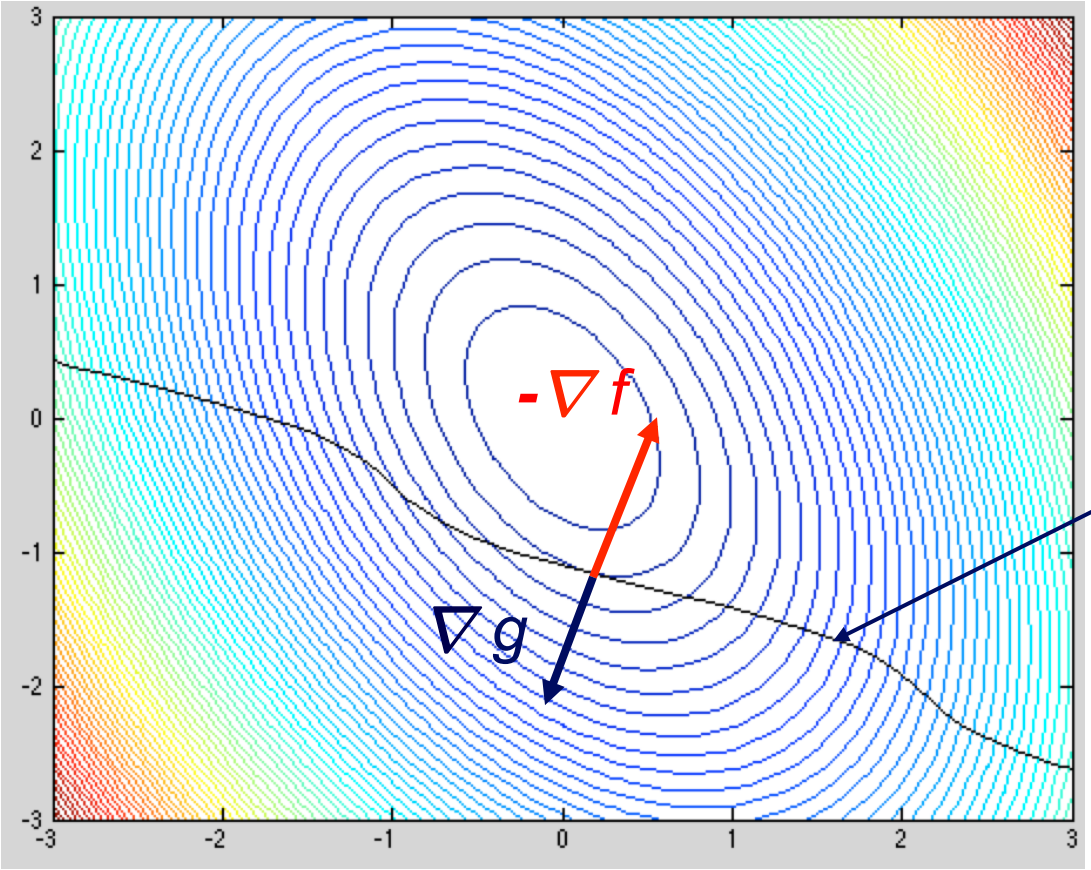
- Note, $\nabla f(\mathbf{x}^*) \neq 0$.

- 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(**x**)* a Scalar.

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

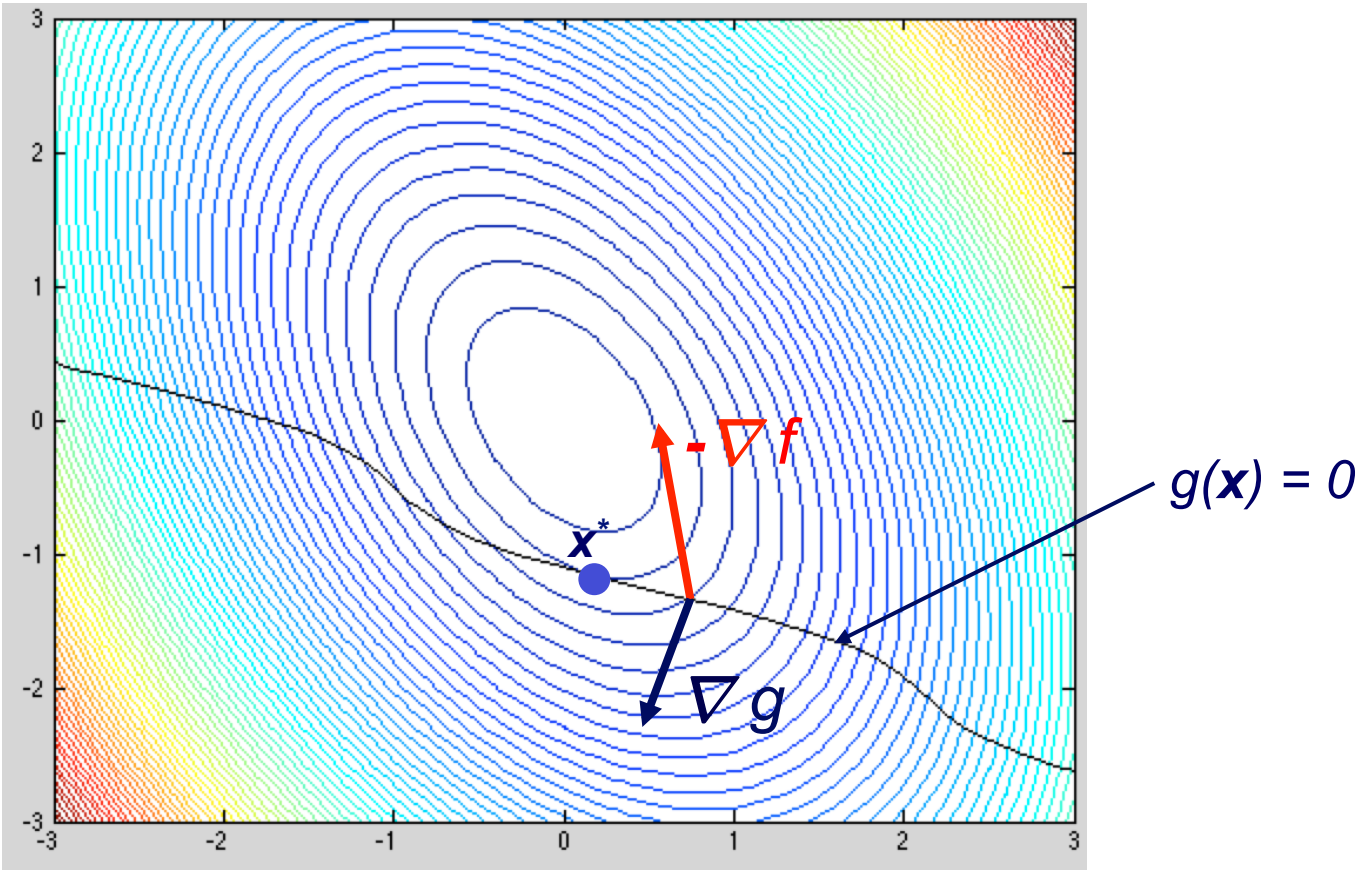


$$-\nabla f(\mathbf{x}^*) = \mathbf{J}_g^T(\mathbf{x}^*)\lambda$$

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

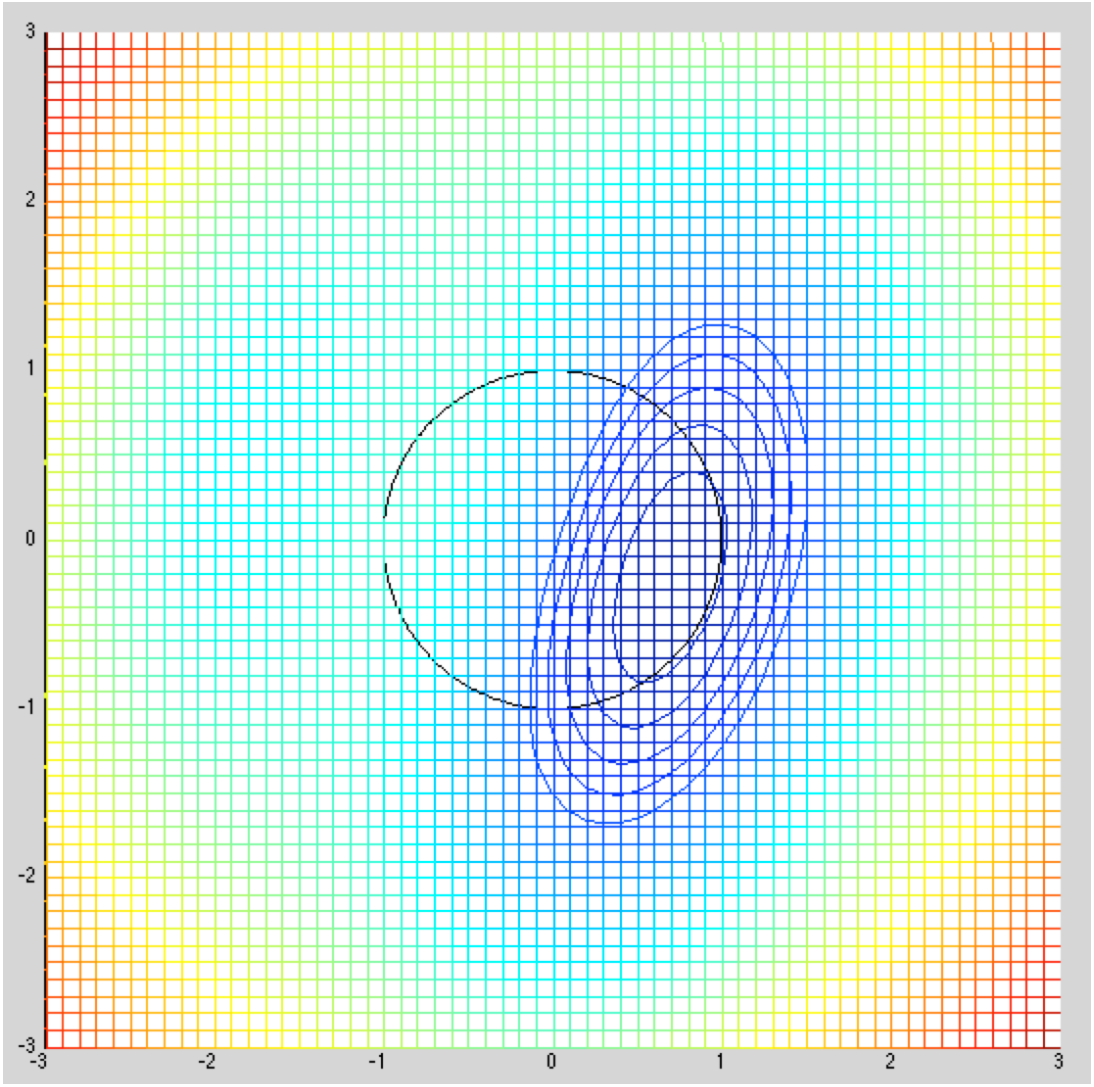
Constrained Optimality Example, $g(\boldsymbol{x})$ a Scalar.

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



Matlab examples

- const_examp.m
- co_const.m



Example: Two Equality Constraints

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

- Lagrangian

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

- First-order conditions

$$\nabla \mathcal{L} \;=\; \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial x_i} \\ \frac{\partial \mathcal{L}}{\partial \lambda_k} \end{pmatrix} \;=\; \begin{pmatrix} \frac{\partial f}{\partial x_i} \;+\; \sum_k \lambda_k \frac{\partial g_k}{\partial x_i} \\ 0 \;+\; \begin{pmatrix} g_1 \\ g_2 \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} \nabla f \;+\; \mathcal{J}_g^T \lambda \\ \mathbf{g} \end{pmatrix} \;=\; \mathbf{0}.$$

Constrained Optimality, continued

- Together, necessary condition and feasibility imply critical point of Lagrangian function,

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

- 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 $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ of $\mathcal{L}$ is constrained minimum of f if $\boldsymbol{B}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ is positive definite on *null space* of $\boldsymbol{J}_g(\boldsymbol{x}^*)$
- If columns of $\boldsymbol{Z}$ form basis for null space, then test *projected* Hessian $\boldsymbol{Z}^T\boldsymbol{B}\boldsymbol{Z}$ for positive definiteness

Constrained Optimality, continued

- Together, necessary condition and feasibility imply critical point of Lagrangian function,

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

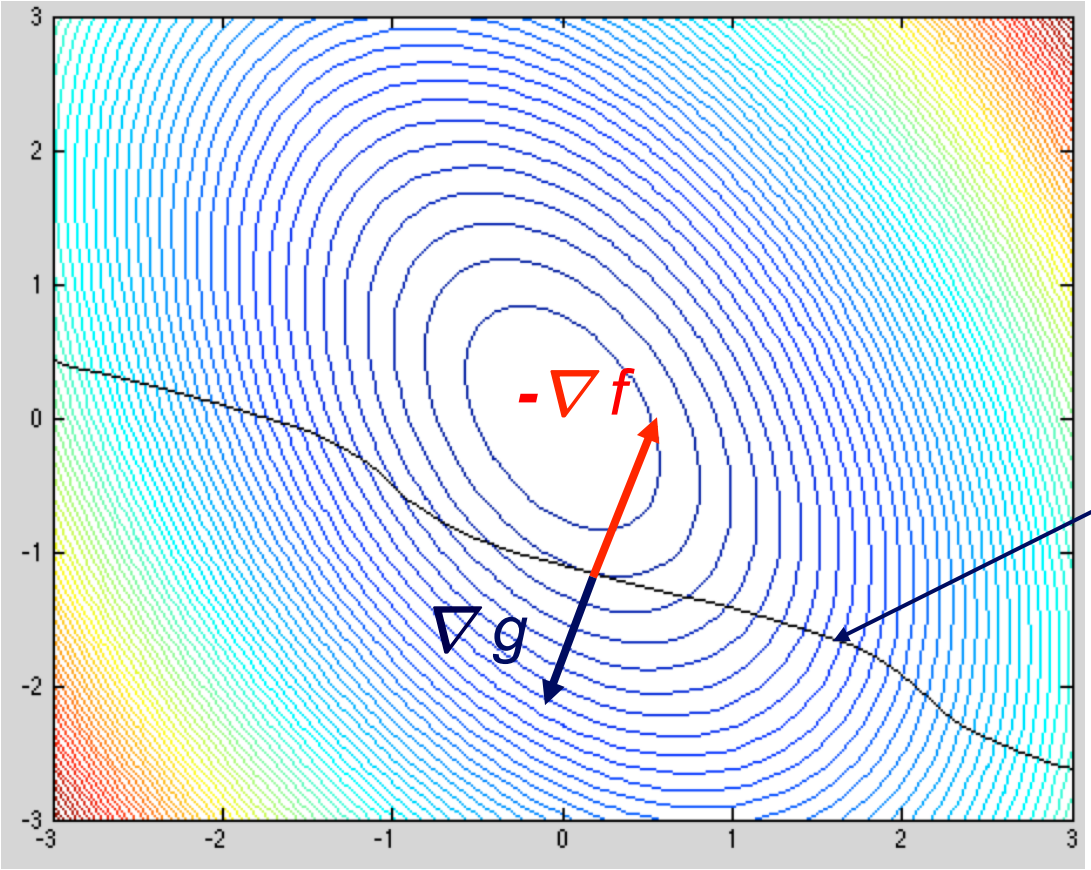
- Hessian of Lagrangian is symmetric, but not positive definite, so critical point of $\mathcal{L}$ is saddle point rather than minimum or maximum

- Critical point $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ of $\mathcal{L}$ is constrained minimum of f if $\boldsymbol{B}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ is positive definite on *null space* of $\boldsymbol{J}_g(\boldsymbol{x}^*)$

- ***Note: null space of $J_g(x^*)$ is the set of vectors tangent to the constraint surface (or surfaces, if more than one g present).***

Constrained Optimality Example, *g(**x**)* a Scalar.

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



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

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

Constrained Optimality, continued

- If inequalities are present, then KKT optimality conditions also require nonnegativity of Lagrange multipliers corresponding to inequalities, and complementarity condition

Karush-Kuhn-Tucker

$$\nabla_x \mathcal{L}(\underline{x}^*, \underline{\lambda}^*) = \underline{0},$$

$$\underline{g}(\underline{x}^*) = \underline{0},$$

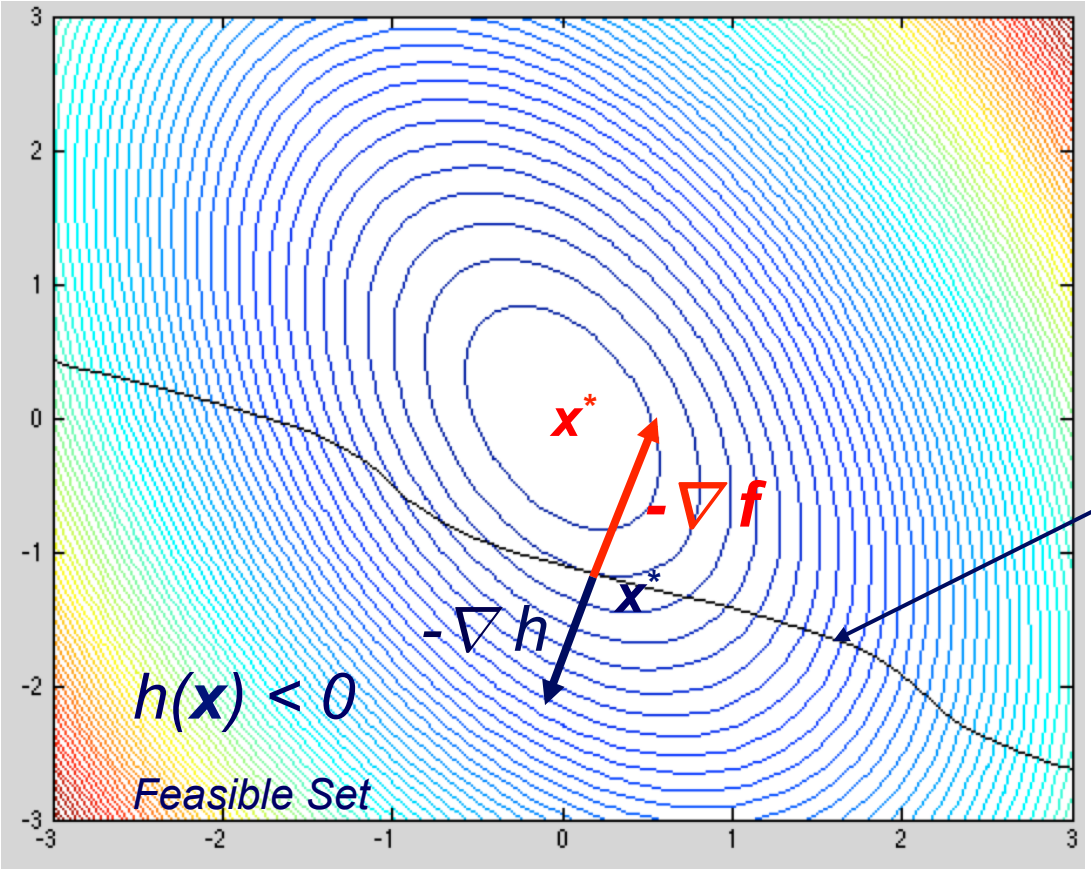
$$\underline{h}(\underline{x}^*) \leq \underline{0},$$

$$\lambda_i^* \geq 0, \ i = m+1, \ldots, m+p \ \text{ inequality Lagrange multipliers}$$

$$\lambda_i^* h_i(\underline{x}^*) = 0, \ i = m+1, \ldots, m+p \ \text{ complementarity condition.}$$

Inequality Constraint and Sign of λ

- A key point is that, at $\boldsymbol{x}^*$, ∇f is parallel to ∇h
- If there are multiple constraints, then ∇f in $span\{\nabla h_k\} = J_h^T \lambda$

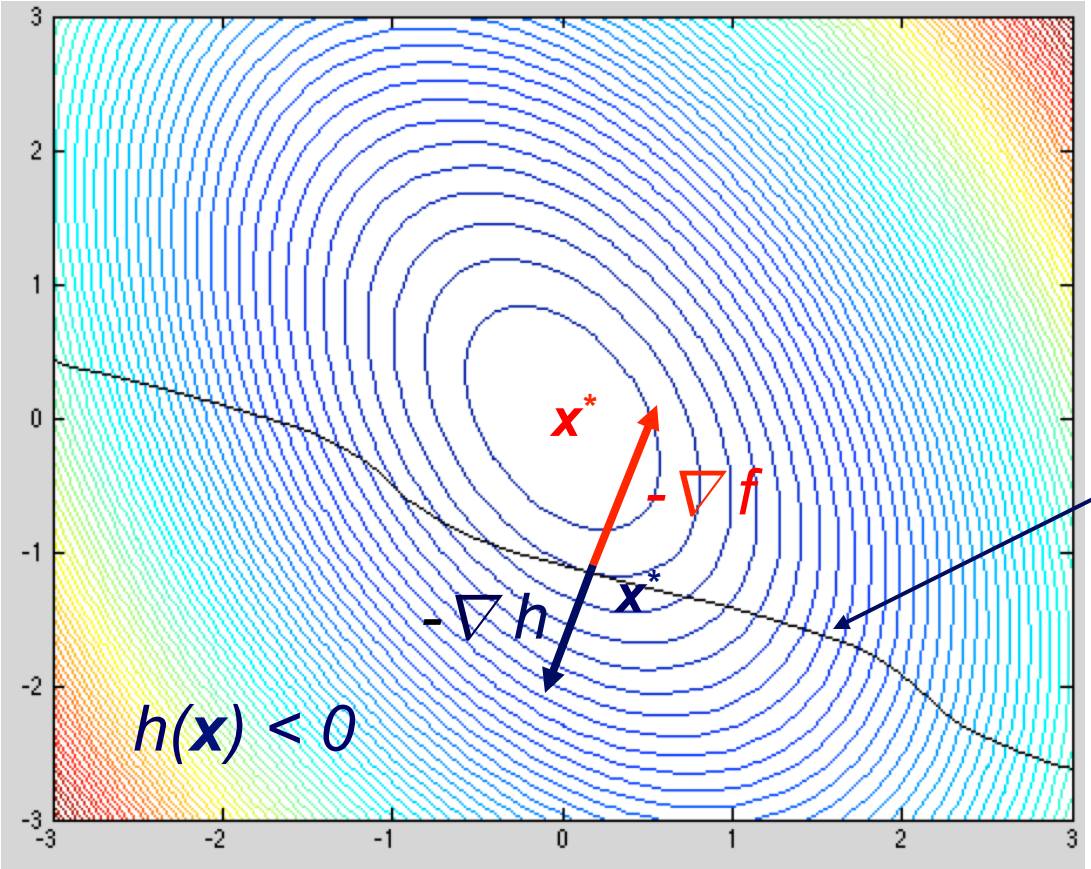


$$-\nabla f(\mathbf{x}^*) = \mathbf{J}_h^T(\mathbf{x}^*)\lambda$$

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

Inequality Constraint and Sign of λ

- If constraint is active $(h(x^*) = 0)$ then ∇f and ∇h point in opposite directions.
- $\nabla f + \lambda^* \nabla h = 0.$
- $\lambda^* > 0.$



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

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

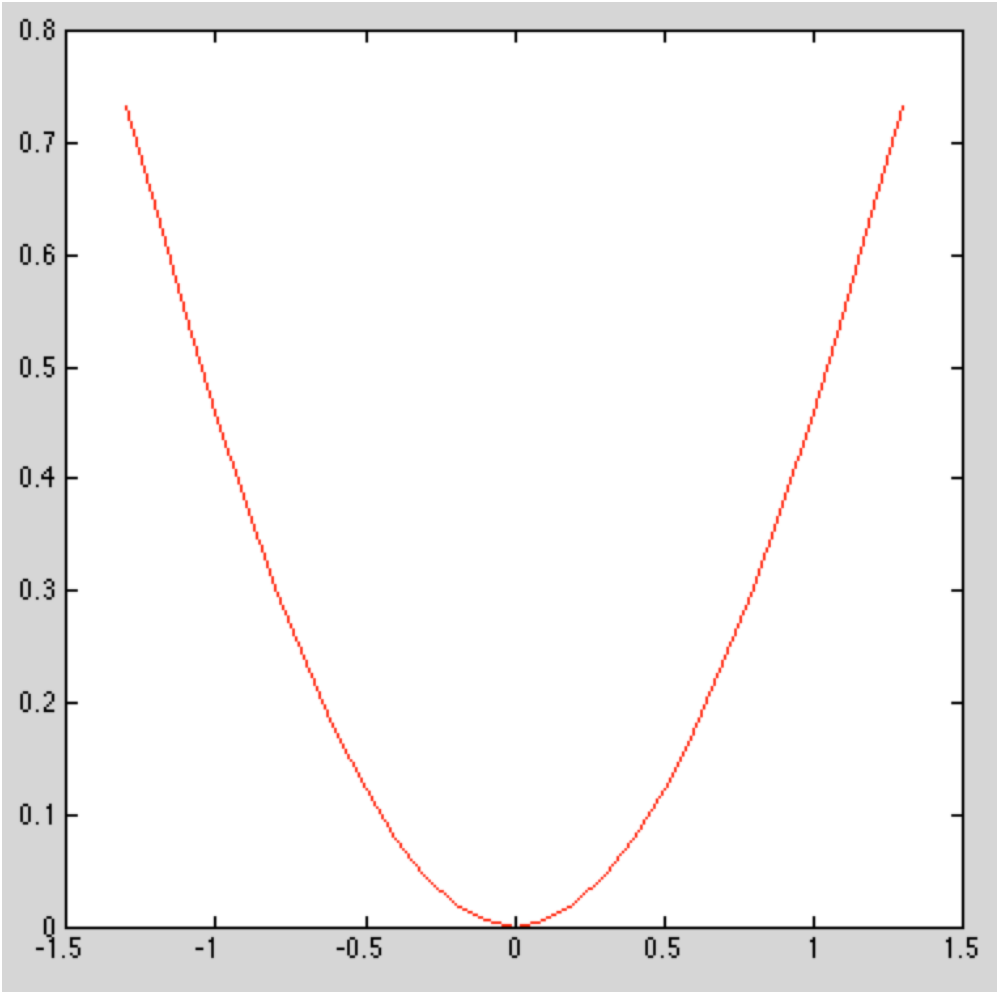
Sensitivity and Conditioning

- 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 $\epsilon/|f'(x^*)|$
- For minimizing f, Taylor series expansion

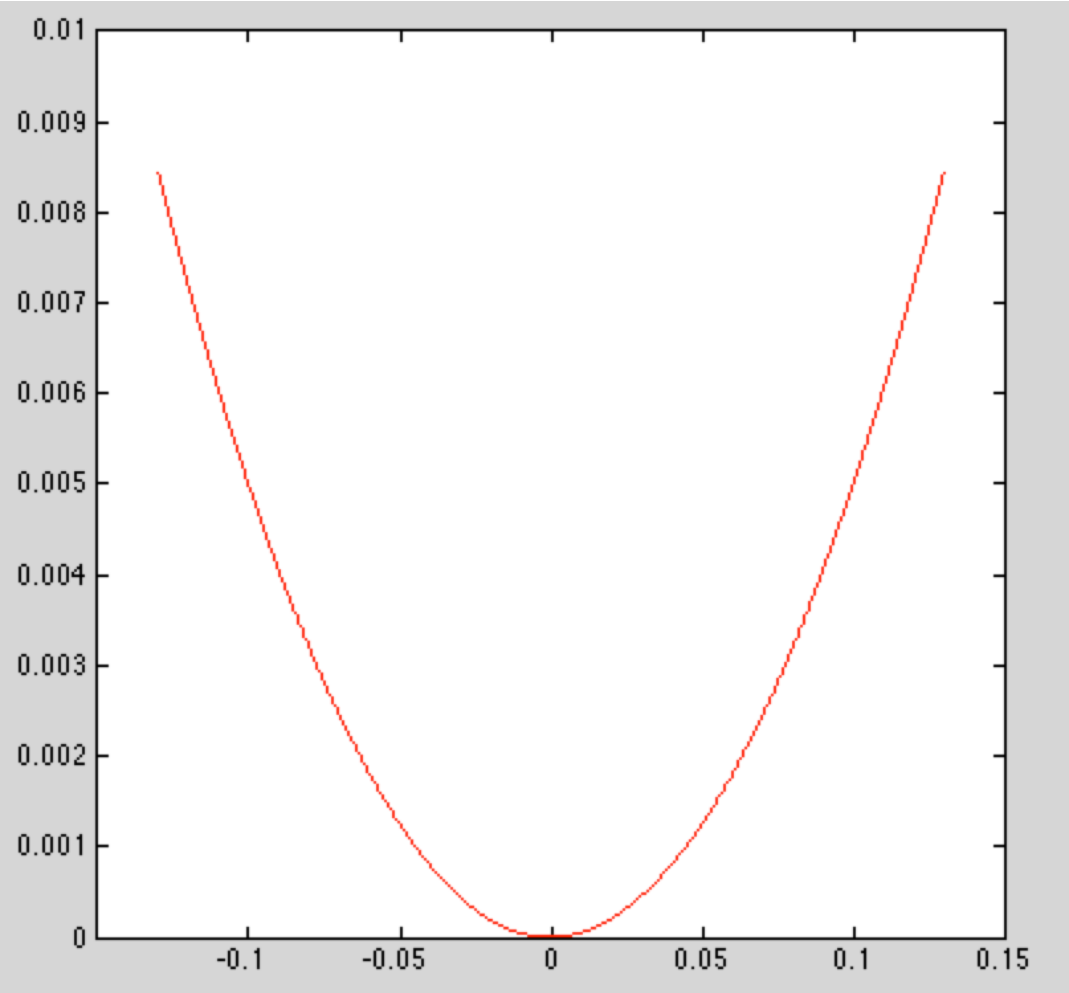
$$\begin{aligned} f(\hat{x}) &= f(x^* + h) \\ &= f(x^*) + f'(x^*)h + \tfrac{1}{2} f''(x^*)h^2 + \mathcal{O}(h^3) \end{aligned}$$

  shows that, since $f'(x^*) = 0$, if $|f(\hat{x}) - f(x^*)| \leq \epsilon$, then $|\hat{x} - x^*|$ may be as large as $\sqrt{2\epsilon/|f''(x^*)|}$
- Thus, based on function values alone, minima can be computed to only about half precision

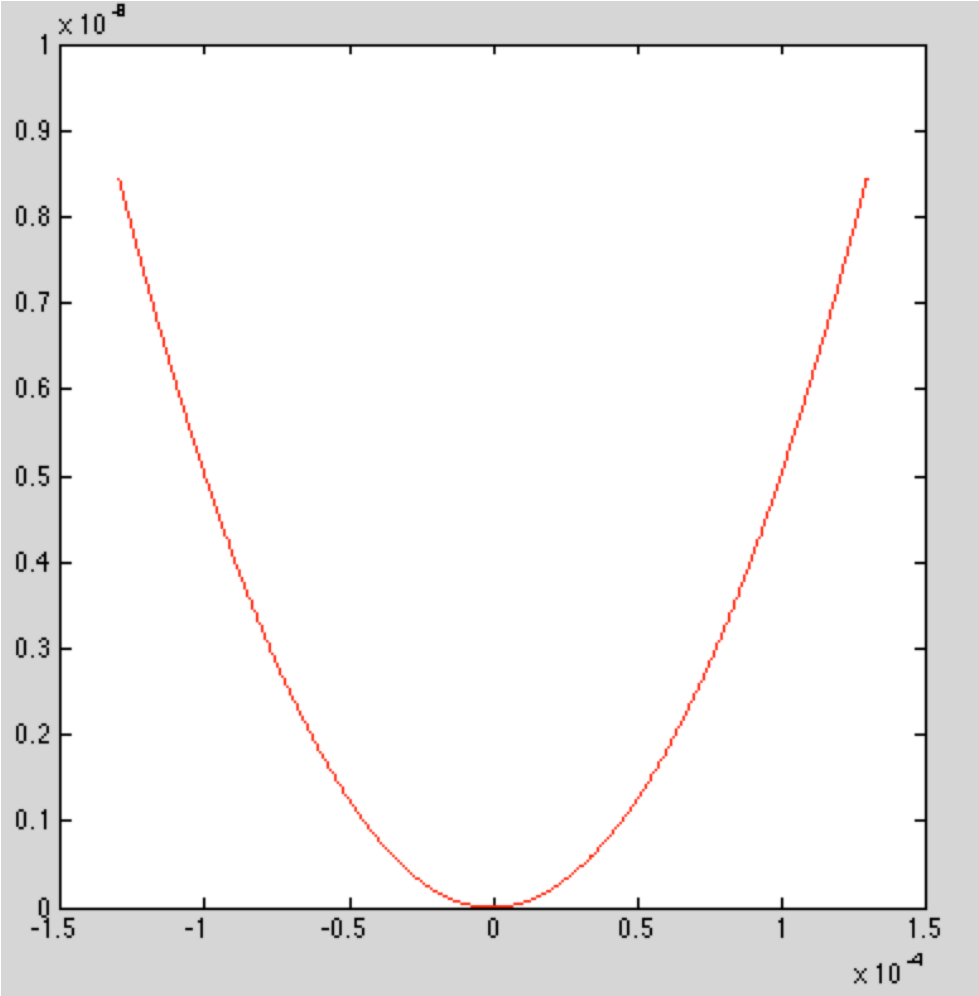
Consider f=1-cos(x)



Consider f=1-cos(x)

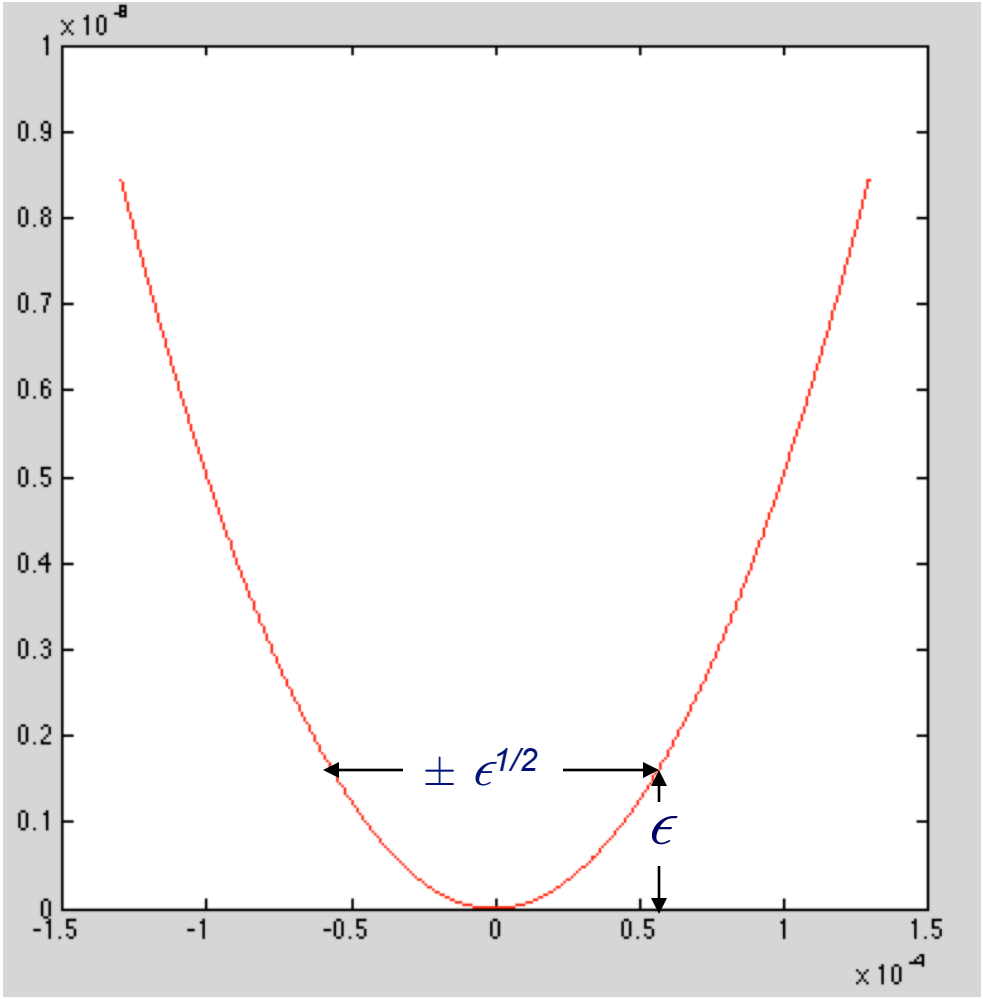


Consider f=1-cos(x)



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

Consider f=1-cos(x)



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

Sensitivity of Minimization

Terminate search when

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

Taylor series:

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

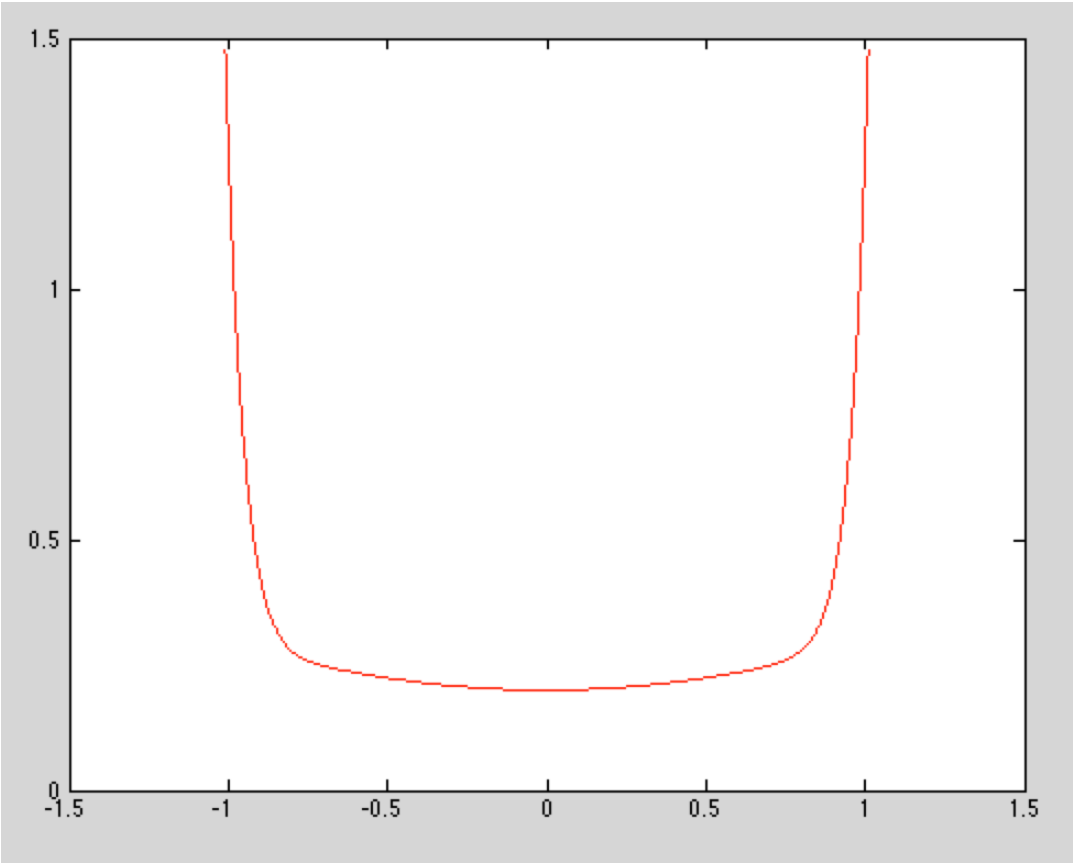
$$\frac{\Delta x^2}{2} \approx \frac{f(x^* + \Delta x) - f(x^*)}{f''(x^*)}$$

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

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

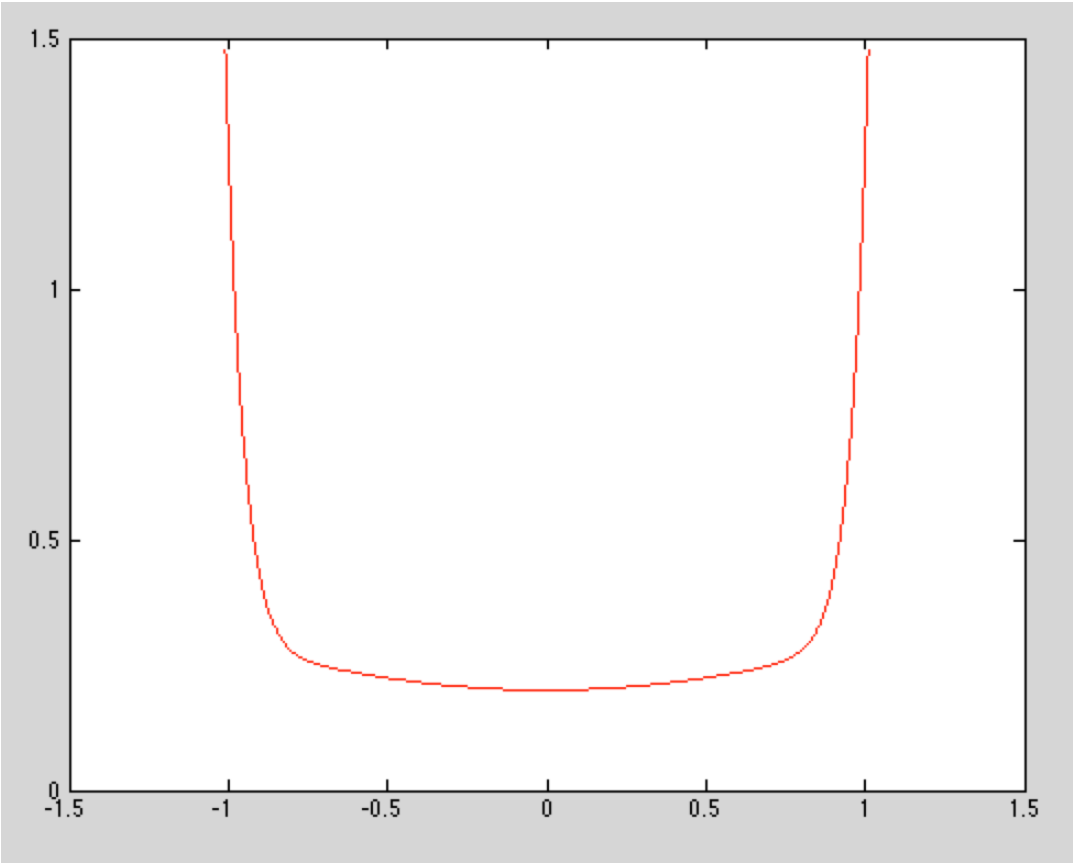
Example: Minimize Cost over Some Design Parameter, x

Cost



Design Parameter

Example: Minimize Cost over Some Design Parameter, x



- While having f''(x*) small 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 in practice is ***good.***

Methods for One-Dimensional Problems

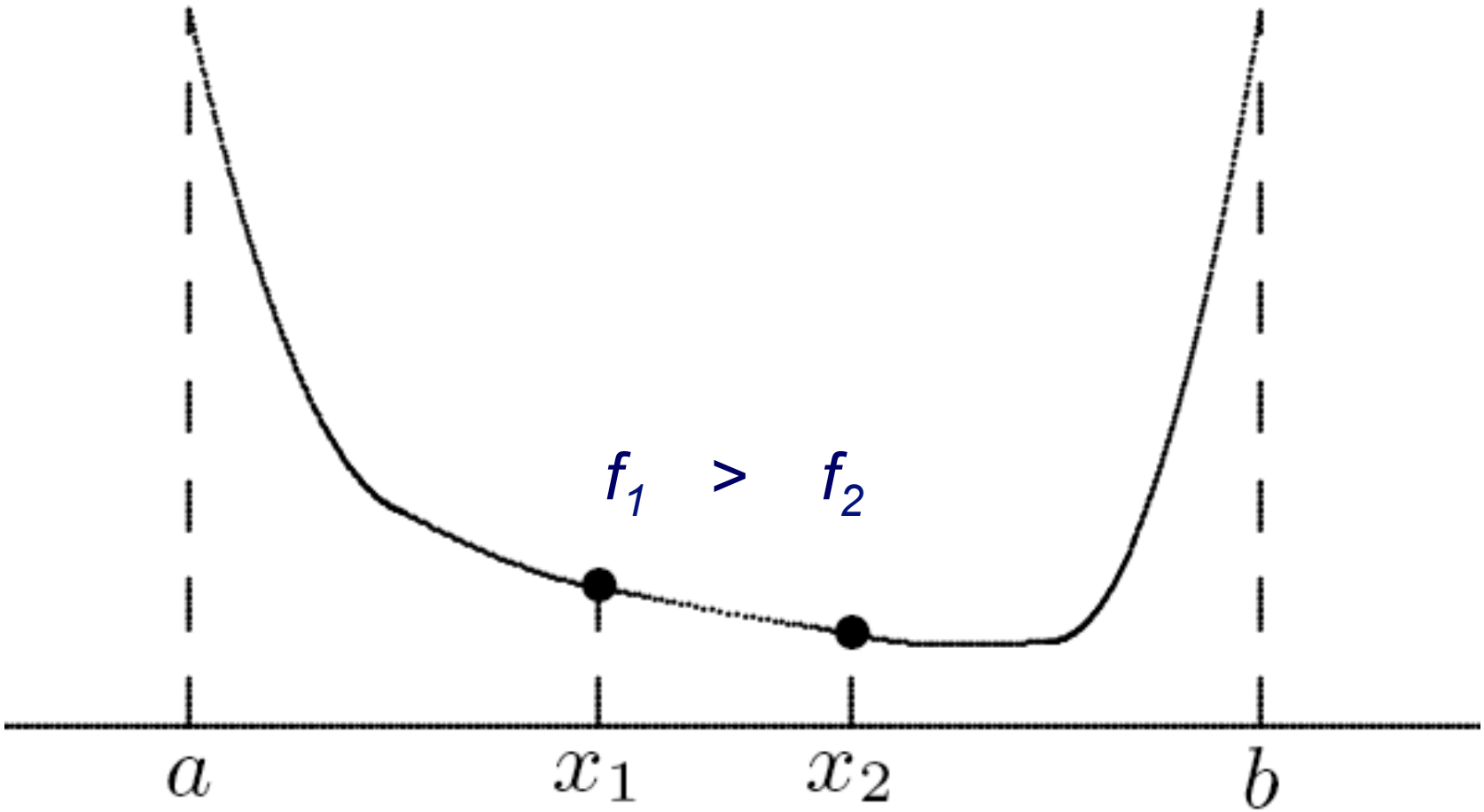
- Demonstrate:
  - basic techniques
  - bracketing
  - convergence rates

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

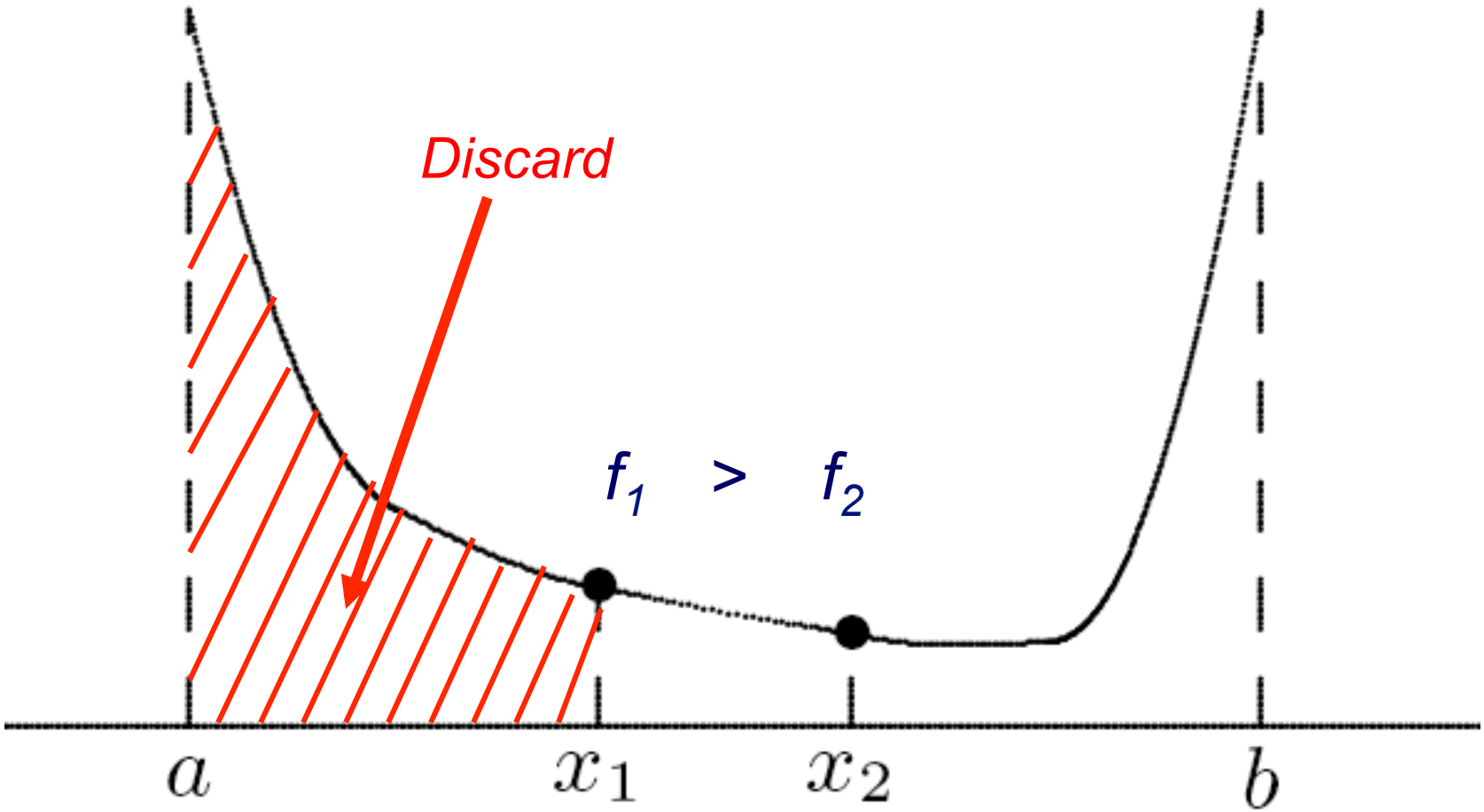
Unimodality

- 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 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

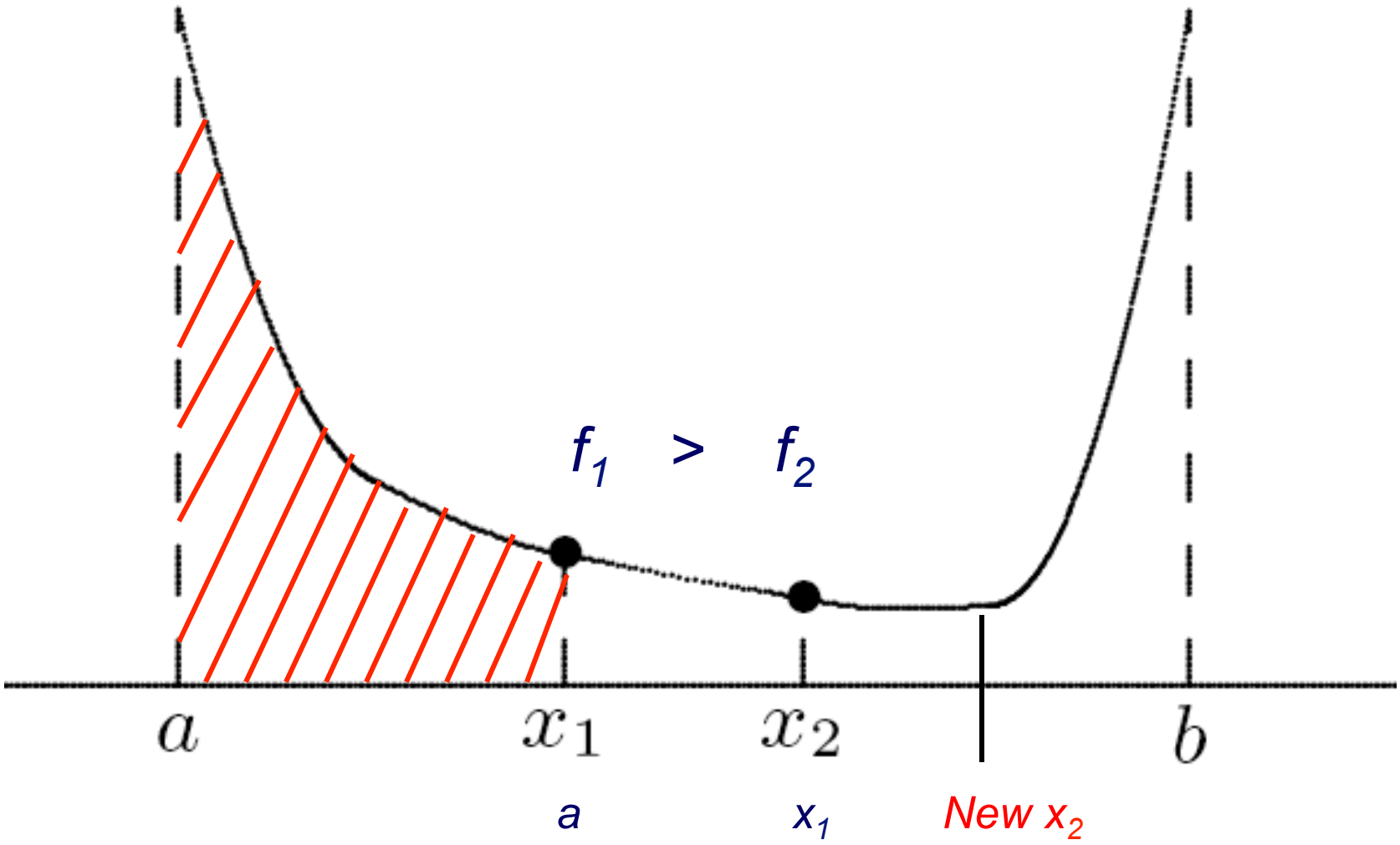
Golden Section Search



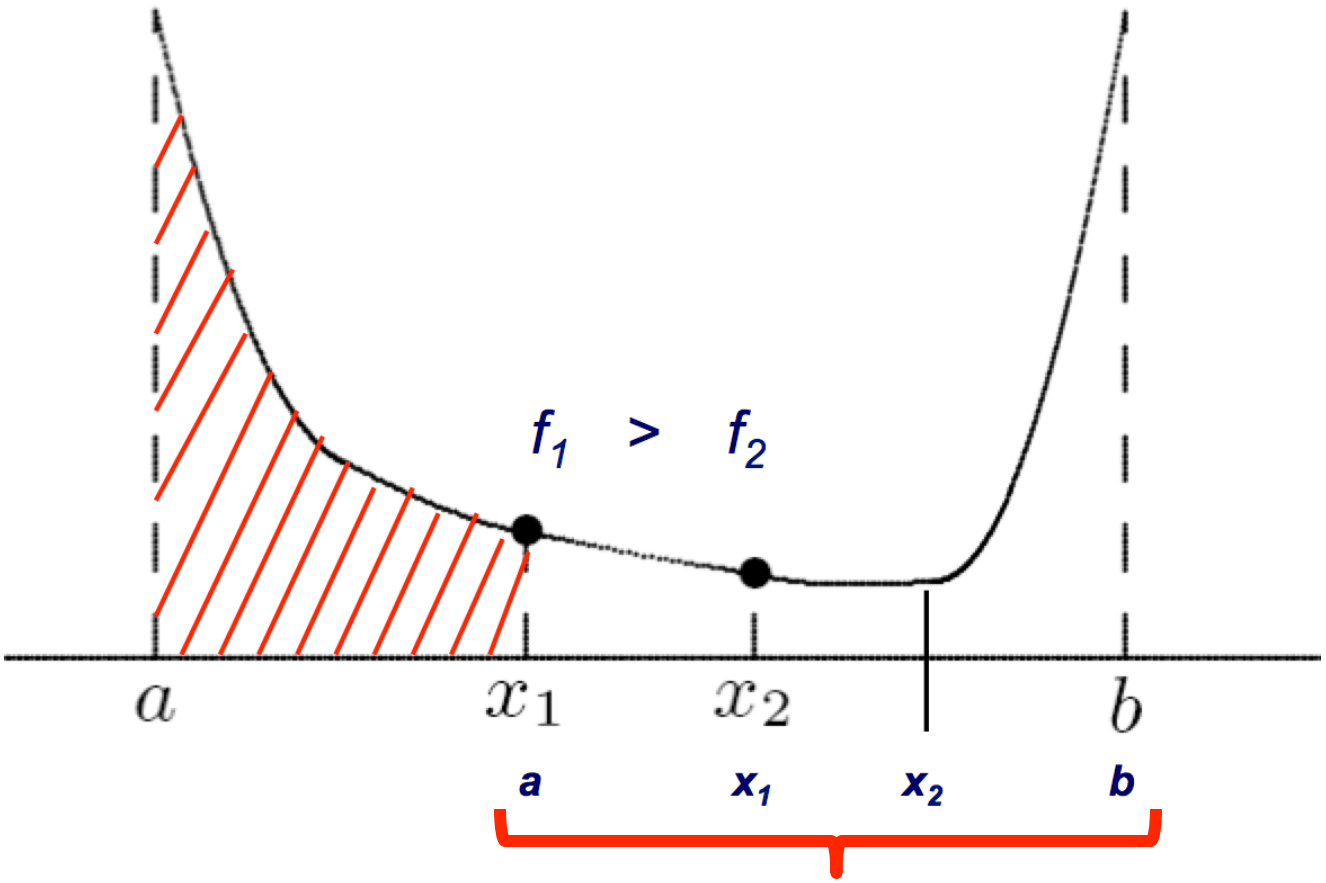
Golden Section Search



Golden Section Search



Golden Section Search



New sample pattern, **[a,x_1,x_2,b] is similar to original.**

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 $(x_2, b]$ or $[a, x_1)$, with minimum known to lie in remaining subinterval
- To repeat process, we need 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

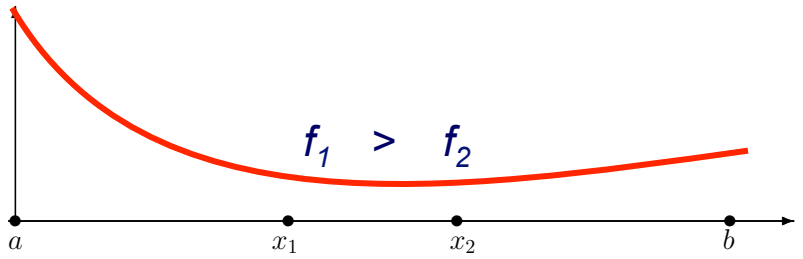
Golden Section Search, continued

- To accomplish this, we 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 previous interval, and interior point retained 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 $C \approx 0.618$

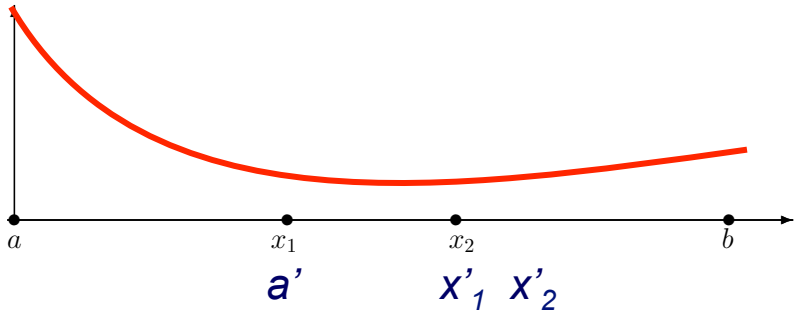
Requires Unimodality?

Golden Section Search

- f(x) unimodal on [a,b].
- Subdivide [a,b] into 3 parts.
- If $f_1 > f_2$ discard (a,x_1)

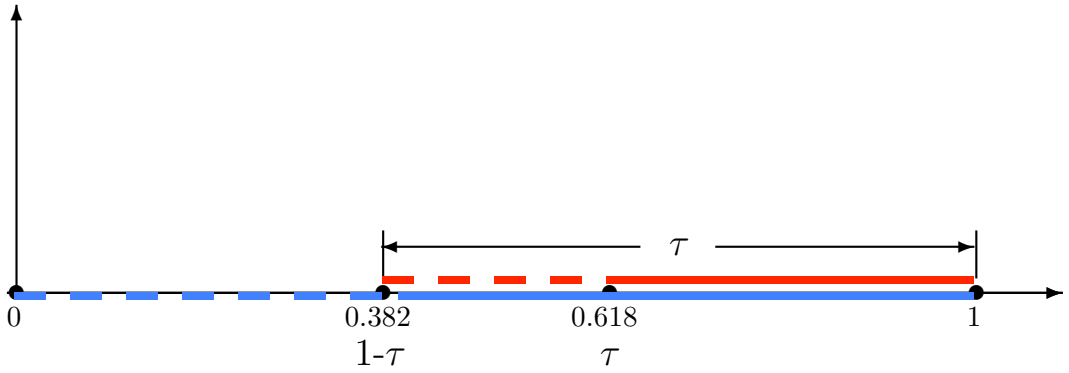


- Choose new point in larger of remaining two intervals: (x_1,x_2) or (x_2,b).
  - → x_1 and x_2 should be closer to center and not at 1/3 , 2/3.



Golden Section Geometry

- Want new section [1-τ, τ] to have same relation as [0,1- τ] to [0,1].



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

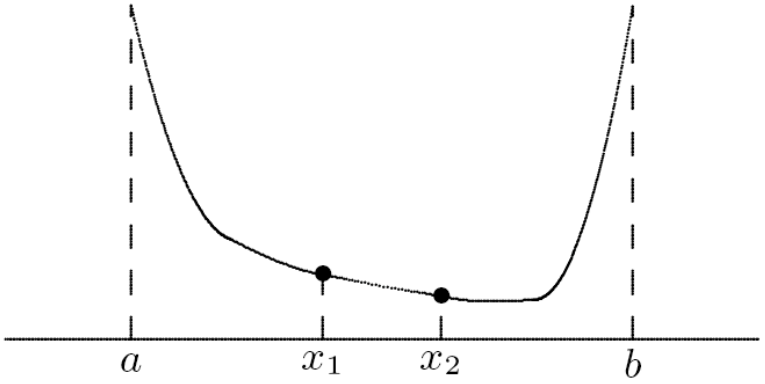
$$2\tau - 1 = \tau - \tau^2$$

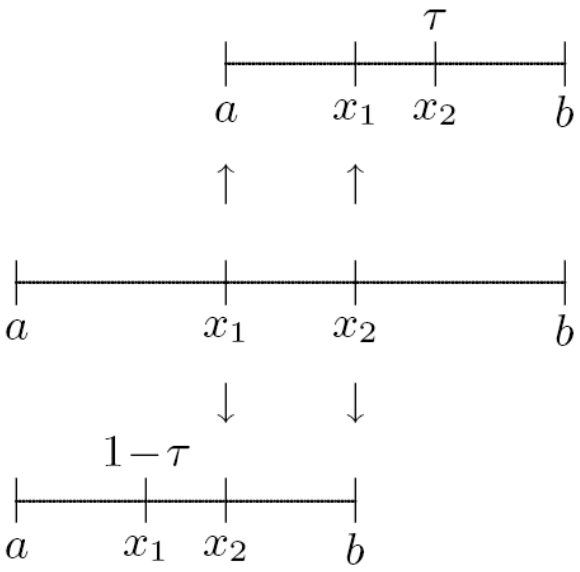
$$\tau^2 + \tau - 1 = 0$$

$$\tau = \frac{-1+\sqrt{1+4}}{2} = \frac{\sqrt{5}-1}{2} = 0.618$$

Golden Section Search, continued

$\tau = (\sqrt{5}-1)/2$
$x_1 = a + (1-\tau)(b-a)$; $f_1 = f(x_1)$
$x_2 = a + \tau(b-a)$; $f_2 = f(x_2)$
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

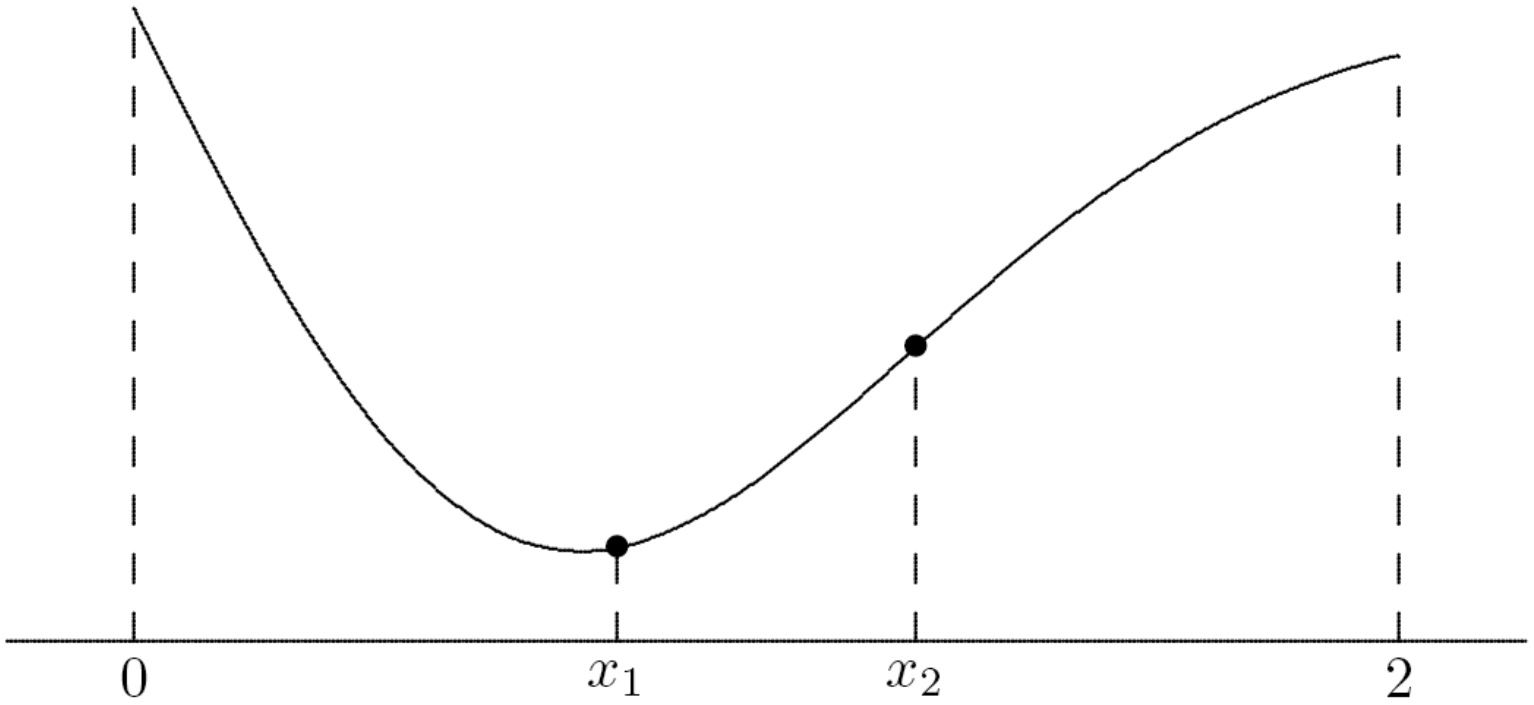




Example: Golden Section Search

Use golden section search to minimize

$$f(x) = 0.5 - x\exp(-x^2)$$



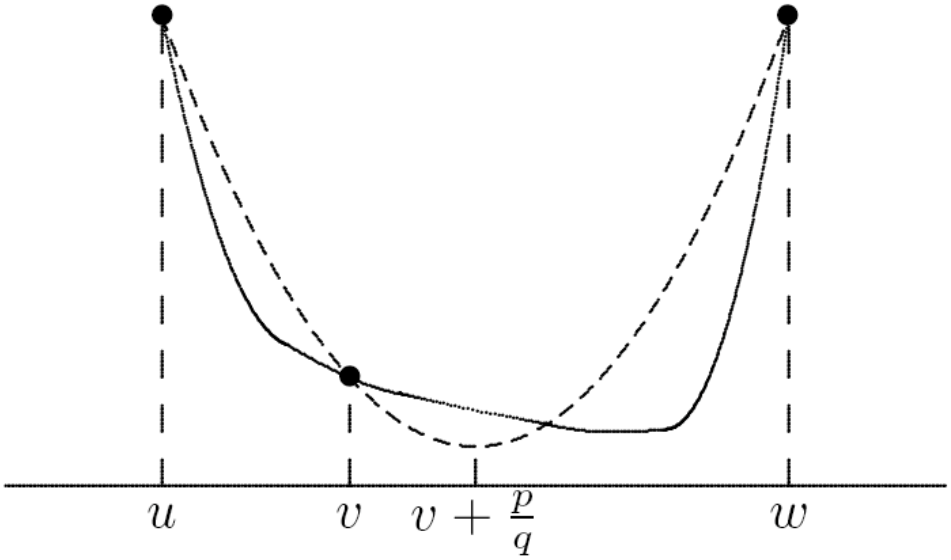
Example, continued

x_1	f_1	x_2	f_2
0.764	0.074	1.236	0.232
0.472	0.122	0.764	0.074
0.764	0.074	0.944	0.113
0.652	0.074	0.764	0.074
0.584	0.085	0.652	0.074
0.652	0.074	0.695	0.071
0.695	0.071	0.721	0.071
0.679	0.072	0.695	0.071
0.695	0.071	0.705	0.071
0.705	0.071	0.711	0.071

golden.m

Successive Parabolic Interpolation

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

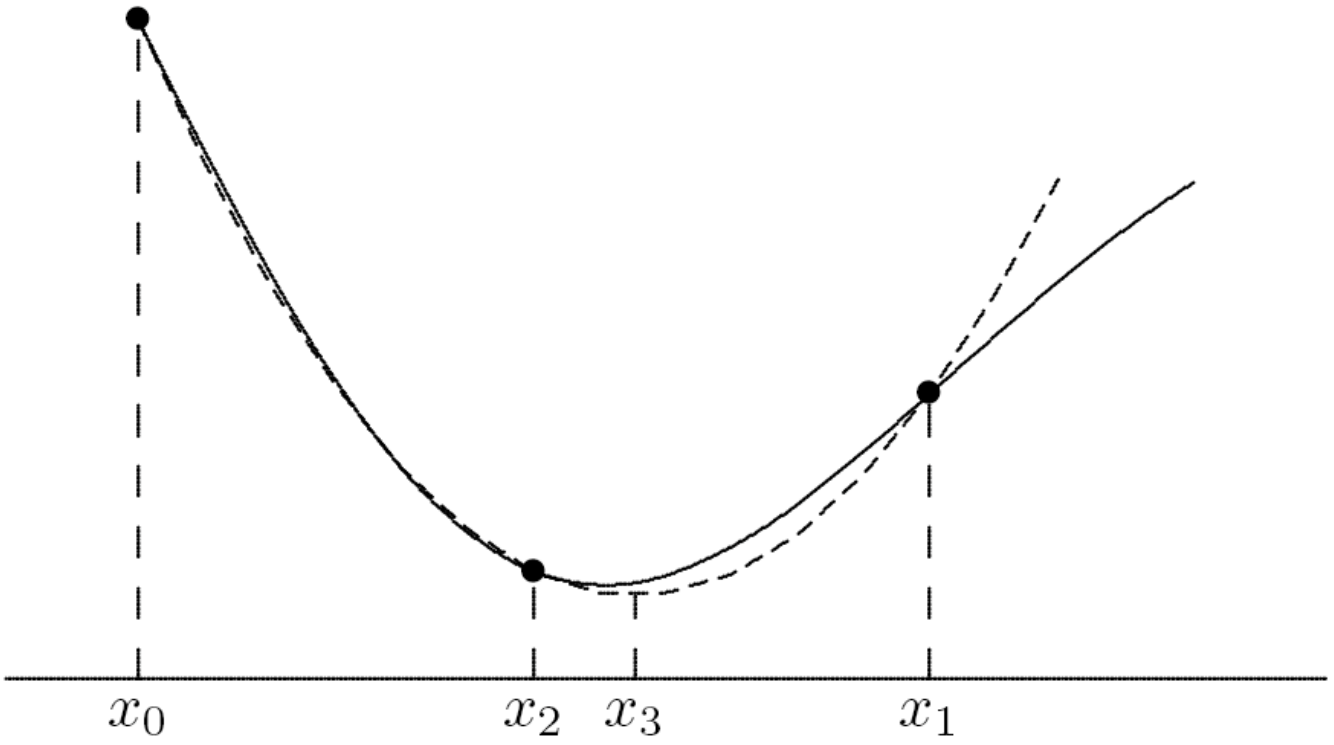


- New point replaces oldest of three previous points and process is repeated until convergence
- Convergence rate of successive parabolic interpolation is superlinear, with $r \approx 1.324$

Example: Successive Parabolic Interpolation

Use successive parabolic interpolation to minimize

$$f(x) = 0.5 - x\exp(-x^2)$$



Example, continued

x_k	$f(x_k)$
0.000	0.500
0.600	0.081
1.200	0.216
0.754	0.073
0.721	0.071
0.692	0.071
0.707	0.071

Not monotone

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.

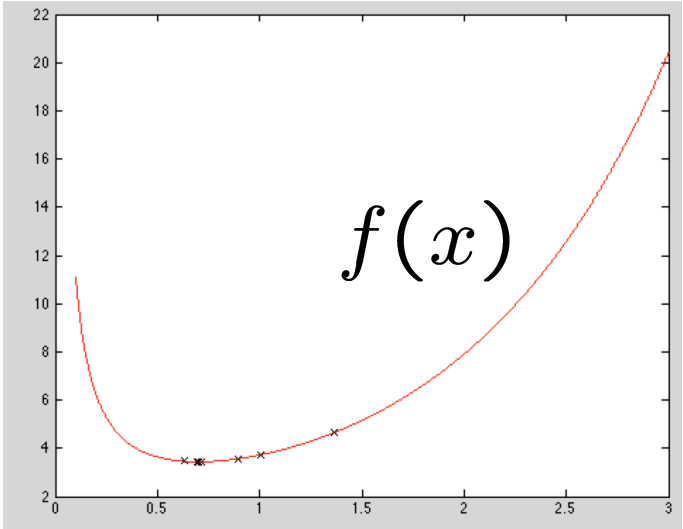
*Matlab: Successive Parabolic Interpolation – **Replace Oldest***

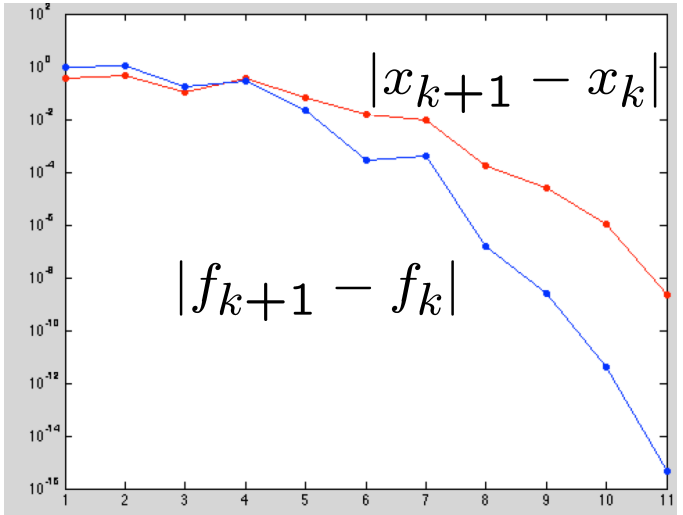
```
function x=parab(a,b,c);

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;
```

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

parabolic.m

```
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;
```





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 this quadratic function of h is given by $h = -f'(x)/f''(x)$
- Suggests iteration scheme

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

  which is *Newton's method* for solving nonlinear equation $f'(x) = 0$
- Newton's method for finding minimum normally has quadratic convergence rate, but must be started close enough to solution to converge **newton1d.m**

Example: Newton's Method

- Use Newton's method to minimize $f(x) = 0.5 - x\exp(-x^2)$
- First and second derivatives of f are given by

$$f'(x) = (2x^2 - 1)\exp(-x^2)$$

and

$$f''(x) = 2x(3 - 2x^2)\exp(-x^2)$$

- 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

x_k	$f(x_k)$
1.000	0.132
0.500	0.111
0.700	0.071
0.707	0.071

Matlab Example: newton.m

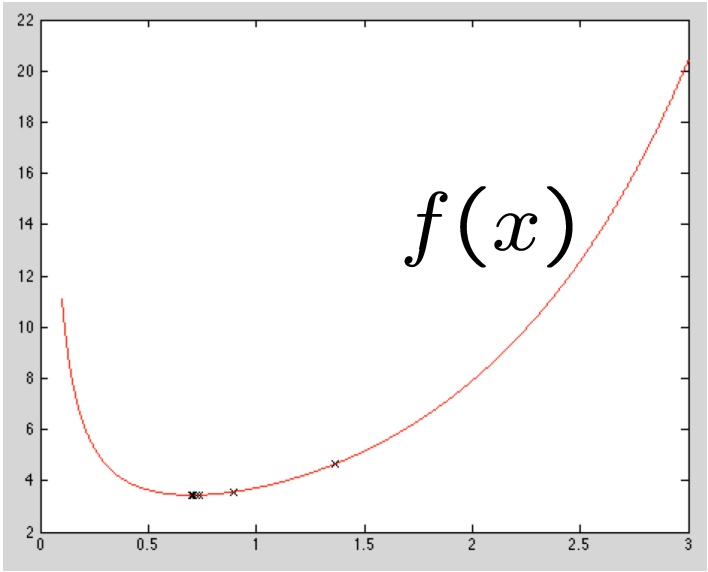
```
format compact; format longe;

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;
```

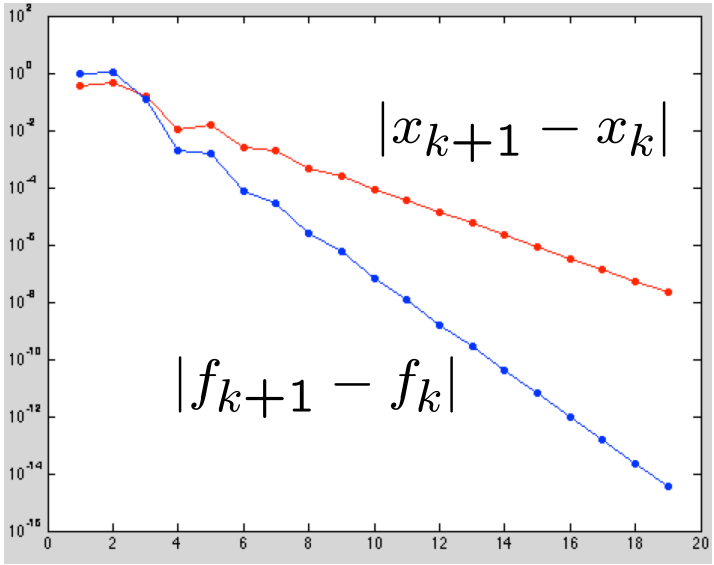
Safeguarded Methods

- As with nonlinear equations in one dimension, slow-but-sure and fast-but-risky optimization methods can be combined to provide both safety 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

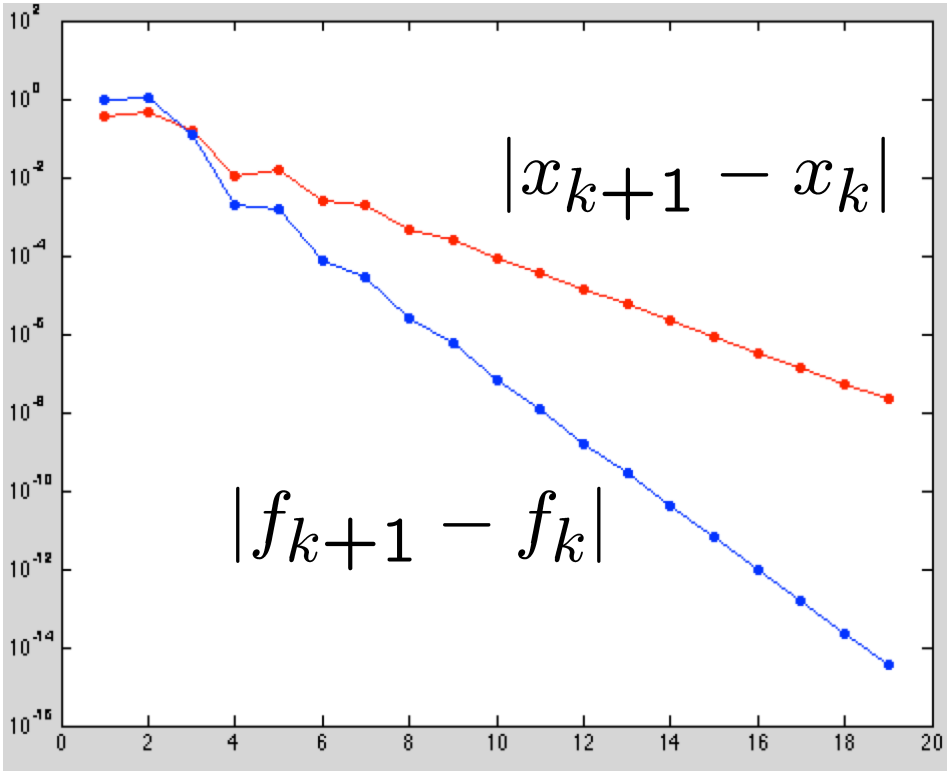
Matlab: Successive Parabolic Interpolation: ***Naive Bracketing***

```
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 Multi-Dimensional Problems

Steepest Descent Method

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

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

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

Steepest Descent, continued

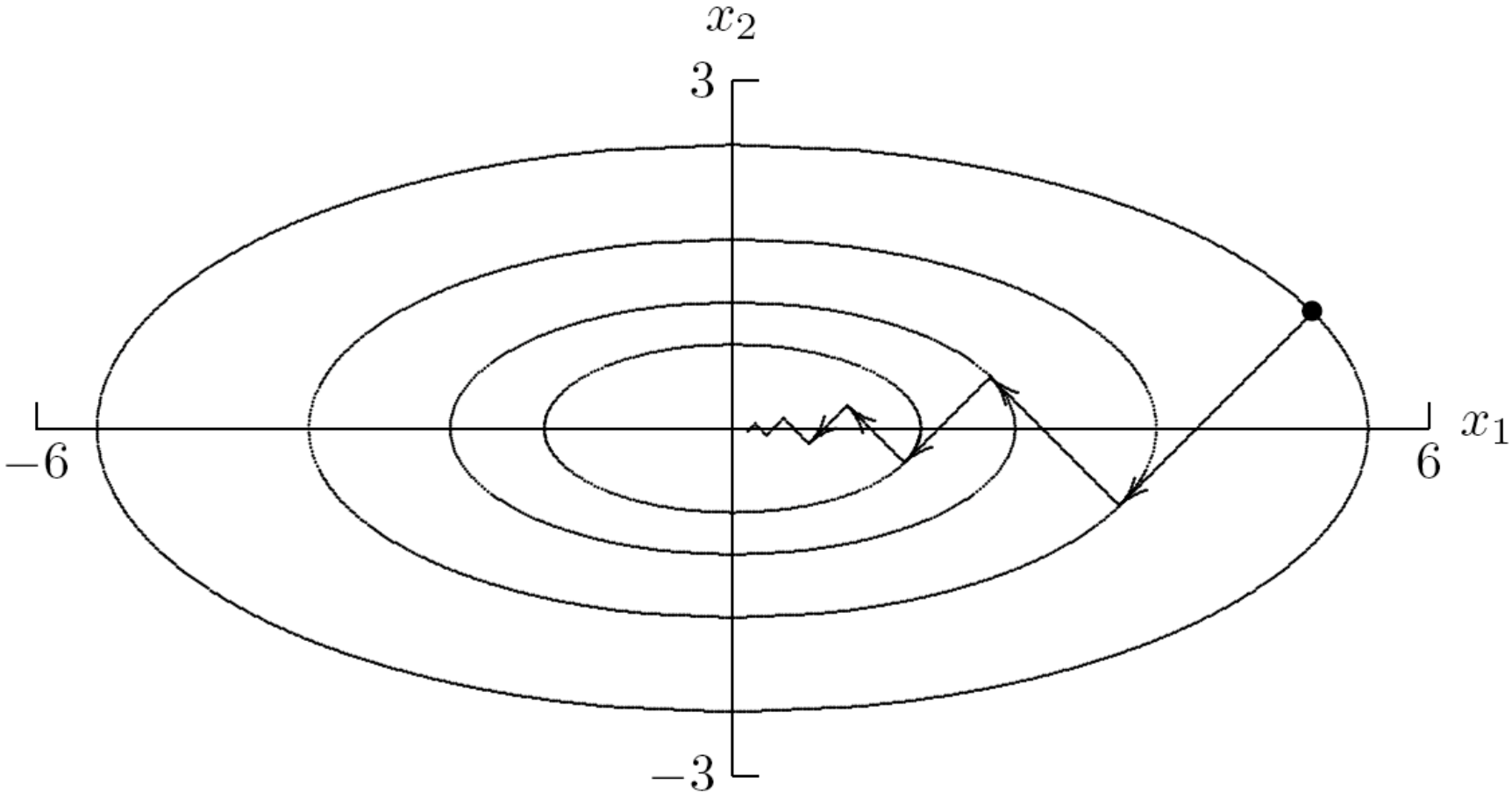
- Given descent direction, such as negative gradient, determining appropriate value for α_k at each iteration is one-dimensional minimization problem

$$\min_{\alpha_k} f(\boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k))$$

  that can be solved by methods already discussed

- Steepest descent method is very reliable: it can always make progress provided gradient is nonzero
- But method is myopic in its view of function's behavior, and resulting iterates can zigzag back and forth, making very slow progress toward solution
- In general, convergence rate of steepest descent is only linear, with constant factor that can be arbitrarily close to 1

Example, continued



Example: Steepest Descent

- Use steepest descent method to minimize

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

- Gradient is given by $\nabla f(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$
- Taking $\boldsymbol{x}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$, we have $\nabla f(\boldsymbol{x}_0) = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$
- Performing line search along negative gradient direction,

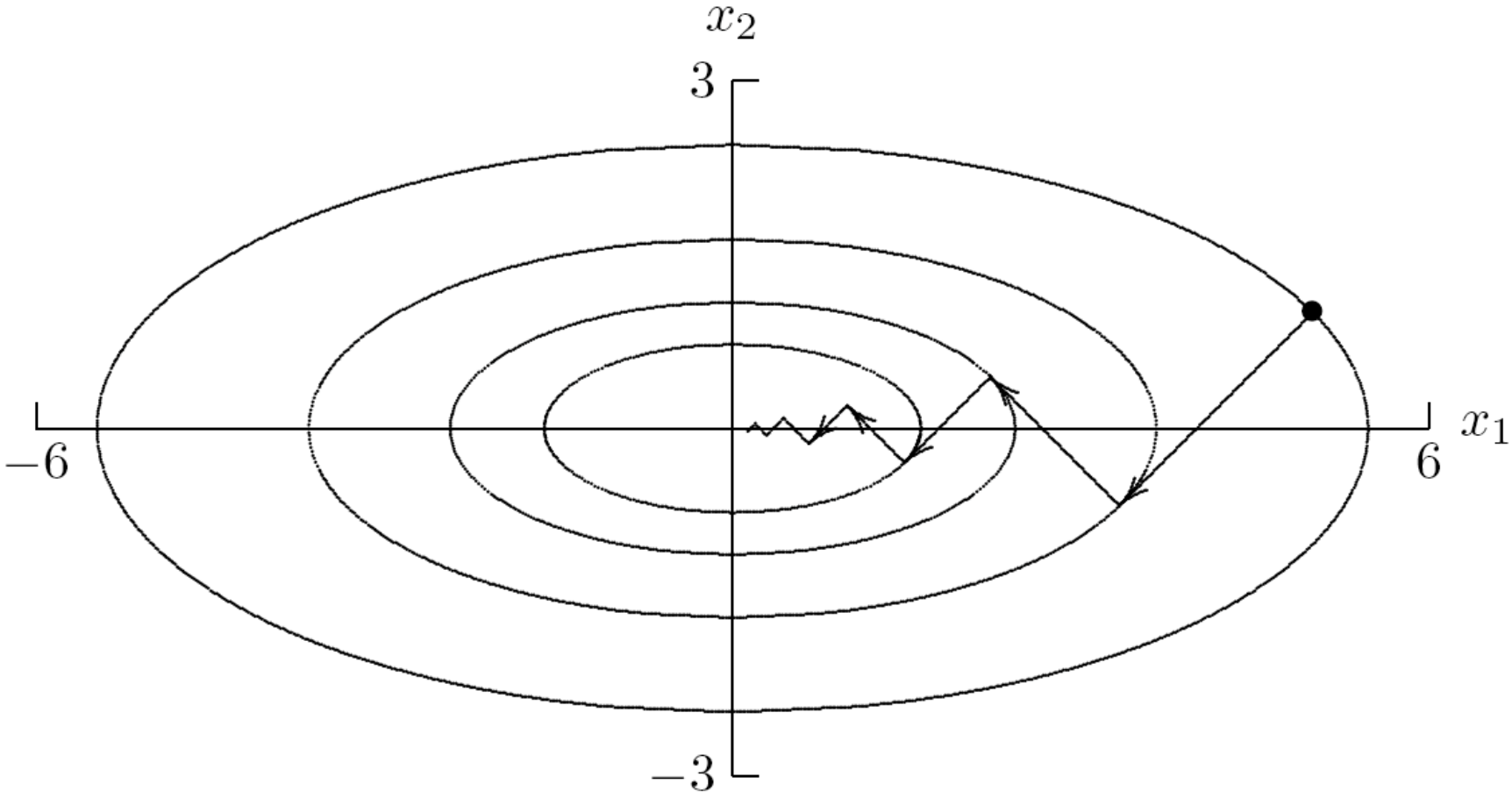
$$\min_{\alpha_0} f(\boldsymbol{x}_0 - \alpha_0 \nabla f(\boldsymbol{x}_0))$$

exact minimum along line is given by $\alpha_0 = 1/3$, so next approximation is $\boldsymbol{x}_1 = \begin{bmatrix} 3.333 \\ -0.667 \end{bmatrix}$

Example, continued

$\boldsymbol{x}_k$		$f(\boldsymbol{x}_k)$	$\nabla f(\boldsymbol{x}_k)$	
5.000	1.000	15.000	5.000	5.000
3.333	−0.667	6.667	3.333	−3.333
2.222	0.444	2.963	2.222	2.222
1.481	−0.296	1.317	1.481	−1.481
0.988	0.198	0.585	0.988	0.988
0.658	−0.132	0.260	0.658	−0.658
0.439	0.088	0.116	0.439	0.439
0.293	−0.059	0.051	0.293	−0.293
0.195	0.039	0.023	0.195	0.195
0.130	−0.026	0.010	0.130	−0.130

Example, continued



< interactive example >

Newton's Method

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

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

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

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

Newton's Method, continued

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

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

  for Newton step $\boldsymbol{s}_k$, then take as next iterate

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

- 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

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

- Gradient and Hessian are given by

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

- Taking $\boldsymbol{x}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$, we have $\nabla f(\boldsymbol{x}_0) = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$
- Linear system for Newton step is $\begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} \boldsymbol{s}_0 = \begin{bmatrix} -5 \\ -5 \end{bmatrix}$, so $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix} + \begin{bmatrix} -5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, which is exact solution for this problem, as expected for quadratic function

Newton's Method, continued

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

Newton's Method, continued

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

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

- In this case, alternative descent direction can be computed, such as negative gradient or direction of negative curvature, and then perform line search

cg4.m demo

Quasi-Newton Methods

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

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

  where α_k is line search parameter and $\boldsymbol{B}_k$ is approximation to Hessian matrix
- Many quasi-Newton methods are more robust than Newton's method, are superlinearly convergent, and have lower overhead per iteration, which often 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 symmetry of Hessian matrix
- Several secant updating formulas have been developed for minimization that not only preserve symmetry in approximate Hessian matrix, but also preserve positive definiteness
- Symmetry reduces amount of work required by about half, while positive definiteness guarantees that quasi-Newton step will be descent direction

BFGS Method

One of most effective secant updating methods for minimization is *BFGS*

$\boldsymbol{x}_0 =$ initial guess
$\boldsymbol{B}_0 =$ initial Hessian approximation
for $k = 0, 1, 2, \ldots$
  Solve $\boldsymbol{B}_k \, \boldsymbol{s}_k = -\nabla f(\boldsymbol{x}_k)$ for $\boldsymbol{s}_k$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$
  $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$
  $\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + (\boldsymbol{y}_k \boldsymbol{y}_k^T)/(\boldsymbol{y}_k^T \boldsymbol{s}_k) - (\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k)/(\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k)$
end

BFGS Method, continued

- In practice, factorization of $\boldsymbol{B}_k$ is updated rather than $\boldsymbol{B}_k$ itself, so linear system for $\boldsymbol{s}_k$ can be solved at cost of $\mathcal{O}(n^2)$ rather than $\mathcal{O}(n^3)$ work
- Unlike Newton's method for minimization, no second derivatives are required
- Can start with $\boldsymbol{B}_0 = \boldsymbol{I}$, so initial step is along negative gradient, and then second derivative information is gradually built up in approximate Hessian matrix 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

Example: BFGS Method

- Use BFGS to minimize $f(\boldsymbol{x}) = 0.5x_1^2 + 2.5x_2^2$
- Gradient is given by $\nabla f(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$
- Taking $\boldsymbol{x}_0 = \begin{bmatrix} 5 & 1 \end{bmatrix}^T$ and $\boldsymbol{B}_0 = \boldsymbol{I}$, initial step is negative gradient, so

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

- Updating approximate Hessian using BFGS formula, we obtain

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

- Then new step is computed and process is repeated

Example: BFGS Method

$\boldsymbol{x}_k$		$f(\boldsymbol{x}_k)$	$\nabla f(\boldsymbol{x}_k)$	
5.000	1.000	15.000	5.000	5.000
0.000	-4.000	40.000	0.000	-20.000
-2.222	0.444	2.963	-2.222	2.222
0.816	0.082	0.350	0.816	0.408
-0.009	-0.015	0.001	-0.009	-0.077
-0.001	0.001	0.000	-0.001	0.005

- 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, where n is dimension of problem < interactive example >

Conjugate Gradient Method

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

Conjugate Gradient Method, continued

$\boldsymbol{x}_0 =$ initial guess
$\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0)$
$\boldsymbol{s}_0 = -\boldsymbol{g}_0$
for $k = 0, 1, 2, \ldots$
  Choose α_k to minimize $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k)$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k$
  $\boldsymbol{g}_{k+1} = \nabla f(\boldsymbol{x}_{k+1})$
  $\beta_{k+1} = (\boldsymbol{g}_{k+1}^T \boldsymbol{g}_{k+1})/(\boldsymbol{g}_k^T \boldsymbol{g}_k)$
  $\boldsymbol{s}_{k+1} = -\boldsymbol{g}_{k+1} + \beta_{k+1} \boldsymbol{s}_k$
end

- Alternative formula for β_{k+1} is

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

Conjugate Gradient Method for Optimization

$$f(\underline{x}) = \frac{1}{2}\underline{x}^T H \underline{x} - \underline{x}^T \underline{b} + c$$

$$\nabla f = H\underline{x} - \underline{b}$$

Find $\underline{x}^*$ s.t. $\nabla f = 0 \Rightarrow \boxed{H\underline{x}^* = \underline{b}}$ H - SPD

Let

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

$$= \underline{x}_0 + \alpha_0 \underline{s}_0 + \alpha_1 \underline{s}_1 + \cdots + \alpha_{k-1}\underline{s}_{k-1}$$

$$= \begin{bmatrix} \underline{s}_0 & \underline{s}_1 & \cdots & \underline{s}_{k-1} \end{bmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{k-1} \end{pmatrix} \quad \text{If } \underline{x}_0 = 0$$

$$= S_k \underline{\alpha}_k$$

Idea!

Find $\underline{\alpha}_k$ s.t.

$$S_k \underline{\alpha}_k \simeq \underline{x}^*$$

} Least Squares Problem — with a hitch.

We don't know x^*, we only know!

$$H\underline{x}^* = \underline{b}$$

Recall -

Linear least squares:

$$\min_{\underline{\alpha}} \|S\underline{\alpha} - \underline{x}^*\|^2 \rightarrow \underline{s}_i^T(S\underline{\alpha} - \underline{x}^*) = 0 \qquad i = 0, \ldots, k-1$$

$$\underline{s}_i^T S\alpha = \underline{s}_i^T \underline{x}^* = (s_i, x^*) \qquad i = 0, \ldots, k-1$$

We need a computable rhs.: (s_i, x^*).

Use the H inner-product!

$$\min_{\underline{\alpha}} \|S\underline{\alpha} - \underline{x}^*\|_H^2 \rightarrow (s_i, S\alpha)_H = (s_i, \underline{x}^*)_H \qquad i = 0, \ldots, k-1$$

Conjugate Gradient Method for Optimization

$$(\underline{s}_i, \underline{x}^\#)_H = \underline{s}_i^T H \underline{x}^\# = \underline{s}_i^T \underline{b}$$

$$(\underline{s}_i, S\underline{\alpha})_H = \underline{s}_i^T H \left(\sum_{j=0}^{k-1} \alpha_j \underline{s}_j\right)$$

$$= \underline{s}_i^T H \underline{x}_{k-1}$$

As with linear least squares, it helps if

$$\boxed{(s_i, s_j)_H = \delta_{ij}}$$ - orthonormal basis.

- in H-inner product,

Then we have:

$$(\underline{s}_i, S\underline{\alpha})_H = \alpha_i = \underline{s}_i^T \underline{b} \iff \underline{\alpha} = S^T \underline{b}$$

So:

$$\underline{x}_k = \sum_{j=0}^{k-1} \underline{s}_j \alpha_j = S\underline{\alpha} = SS^T \underline{b}$$

$$= \sum_{j=0}^{k-2} \underline{s}_j \alpha_j + \underline{s}_{k-1} \alpha_{k-1}$$

$$= \underline{x}_{k-1} + \alpha_{k-1} \underline{s}_{k-1} \quad \left\{ \begin{array}{l} \alpha_{k-1} \text{ chosen to minimize} \\ f(x_{k-1} + \alpha_{k-1} s_{k-1}) \end{array} \right.$$

- Thus, as we augment the search space,

  if $\underline{s}_{k-1} \perp_H \{\underline{s}_0, \underline{s}_1 \cdots \underline{s}_{k-2}\}$

  we have

  $$\underline{x}_k = \underline{x}_{k-1} + \alpha_{k-1} \underline{s}_{k-1} \qquad \alpha_{k-1} = \underset{\alpha}{\operatorname{argmin}}\, f(x_{k-1} + \alpha s_{k-1})$$

  is the best-fit in $(\underline{s}_0 \underline{s}_1 \cdots \underline{s}_{k-1})$ in H-norm

- If H is $n \times n$, then $\underline{x}_n = \underline{x}^\#$

Conjugate Gradient Method for Optimization

To enforce $\underline{s}_k \perp \{\underline{s}_0, \underline{s}_1, \cdots \underline{s}_{k-1}\}$

Let $\underline{s}_k = -\underline{g}_k + P_{k-1}\, \underline{g}_k$

$= -\underline{g}_k + \sum_{j=1}^{k-1} \underline{s}_j \beta_j$ ← Use Gram Schmidt in H-inner product. (H-SPD)

$\underline{s}_k^T H \underline{s}_j = 0 \quad j = 1, \cdots, k-1$

$\underline{s}_i^T H \left(\sum_{j=1}^{k-1} \underline{s}_j \beta_j \right) = -\underline{s}_i^T H \underline{g}_k$

$\beta_i = -\dfrac{\underline{s}_i^T H \underline{g}_k}{\underline{s}_i^T H \underline{s}_i} = \dfrac{\underline{g}_k^T H \underline{s}_i}{\underline{s}_i^T H \underline{s}_i}$

Recall $\underline{g}_k = H \underline{x}_k - H \underline{x}^* = H \underline{e}_k$

$\underline{e}_k^T H \underline{v} = 0 \quad \underline{v} \in (\underline{s}_0, \underline{s}_1 \cdots \underline{s}_{k-1})$

$\underline{g}_0 = H \underline{x}_0 - \underline{b} = \nabla F(\underline{x}_0)$

$\underline{s}_0 = -\underline{g}_0 \in P_0(H)\, \underline{g}_0$

$\underline{x}_1 = \underline{x}_0 + \alpha_0 \underline{s}_0$

$\underline{g}_1 = \nabla F(\underline{x}_1) = H \underline{x}_1 - \underline{b}$

$= H \alpha_0 \underline{s}_0 + \underline{g}_0$

$= -\alpha_0 H \underline{g}_0 + \underline{g}_0 \in P_1(H)\, \underline{g}_0$

$\underline{s}_1 = -\underline{g}_1 + \beta_0 \underline{s}_0 \in P_1(H)\, \underline{g}_0$

$\underline{g}_2 = H(\underline{x}_1 + \alpha_1 \underline{s}_1) - \underline{b}$

$= \underline{g}_1 + \alpha_1 H \underline{s}_1 \in P_2(H)\, \underline{g}_0$

$\underline{s}_2 = -\underline{g}_2 + \sum_{j=0}^{1} \beta_j \underline{s}_j \in P_2(H)\, \underline{g}_0$

Conjugate Gradient Method for Optimization

$\therefore H s_i \in P_{i+1}(H) g_0$

$e_k \perp_H (H s_i) \quad i = 1, \cdots k-2$

$\therefore g_k^T H s_i = 0 \quad i = 1, \cdots, k-2$

Only $\beta_{k-1} = \dfrac{g_k^T H s_{k-1}}{s_{k-1}^T H s_{k-1}}$ survives

So, H-orthogonal search direction is

$$s_k = -g_k + \sum_{j=0}^{k-1} \beta_j s_j$$

$$= -g_k + \beta_{k-1} s_{k-1}$$

$$\boxed{\beta_{k-1} = \frac{g_k^T H s_{k-1}}{s_{k-1}^T H s_{k-1}}}$$

$g_k = g_{k-1} + \alpha_{k-1} H s_{k-1}$

$g_k^T g_k = g_k^T g_{k-1} + \alpha_{k-1} g_k^T H s_{k-1}$

$g_k^T g_{k-1} = 0$

$(s_i^T H s_i) \alpha_i = s_i^T g_{i-1}$

$s_{k-1}^T H s_{k-1} \alpha_{k-1} = s_{k-1}^T g_{k-1}$

$\alpha_{k-1} = \dfrac{s_{k-1} g_{k-1}}{s_{k-1}^T H s_{k-1}}$

$g_k^T H s_{k-1} = \dfrac{g_k^T g_k}{\alpha_{k-1}} = \dfrac{s_{k-1}^T H s_{k-1}}{s_{k-1}^T g_{k-1}} \cdot g_k^T g_k$

$\dfrac{g_k^T H s_{k-1}}{s_{k-1}^T H s_{k-1}} = \dfrac{g_k^T g_k}{g_{k-1}^T s_{k-1}^T} = \dfrac{g_k^T g_k}{g_{k-1}^T g_{k-1} - g_{k-1}^T s_{k-2} \beta}$

Conjugate Gradient Iteration for Optimization

- Consider the objective function

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

- Let's take the case H=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

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

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

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

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

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

- That is, $\mathbf{x}_{k+1}$ is a linear combination of the preceeding $k+1$ search directions $\mathbf{s}_j$.
- **Main Idea:** *Find* $\mathbf{a}_k = [\alpha_0\, \alpha_1\, \cdots\, \alpha_k]^T$ *such that*

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

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

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

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

Linear Least Squares in H Inner-Product.

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

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

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

$$\left(S_k^T H S_k\right) \mathbf{a}_k = S_k^T \mathbf{b}.$$

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

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

- Solving for α_k becomes trivial:

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

  - Note that, for optimization problems, we don't know 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 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: $\mathbf{s}_k \perp_H S_{k-1}$.

- Thus, set:

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

with

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

- As we saw in Chapter 3 notes, $\gamma_j = 0$ for $j < k$ because $\mathbf{g}_{k+1} = H\mathbf{e}_{k+1}$ and $\mathbf{e}_{k+1} \;\perp_H\; HS_j$, $j < k$.

- 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

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

Summary of CG for Optimization

- Conjugate gradient iteration produces the projection of x^* onto span $(s_0\ s_1\ \ldots s_k)$
- For constant SPD Hessian, ***H***, CG iteration will be exact after:
  - n iterations, or m < n iterations if H has only m distinct eigenvalues.
- For nonconstant ***H***, CG is not exact. However ***H*** → constant as $x \rightarrow x^*$
- **Important** to restart orthogonalization process after n iterations.
  - Otherwise, $\boldsymbol{S^T H S}$ will be k x k, with k > n, and will be singular.
- Method does not need ***H***, nor $\underline{\boldsymbol{b}}$, only $\nabla \boldsymbol{f}$, plus line search in the direction $\underline{\boldsymbol{s}}_k$ (not $\underline{\boldsymbol{g}}_k := \nabla \boldsymbol{f}$, which would be steepest descent).

Conjugate Gradient Method, continued

$\boldsymbol{x}_0 =$ initial guess
$\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0)$
$\boldsymbol{s}_0 = -\boldsymbol{g}_0$
for $k = 0, 1, 2, \ldots$
  Choose α_k to minimize $f(\boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k)$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k$
  $\boldsymbol{g}_{k+1} = \nabla f(\boldsymbol{x}_{k+1})$
  $\beta_{k+1} = (\boldsymbol{g}_{k+1}^T \boldsymbol{g}_{k+1})/(\boldsymbol{g}_k^T \boldsymbol{g}_k)$
  $\boldsymbol{s}_{k+1} = -\boldsymbol{g}_{k+1} + \beta_{k+1} \boldsymbol{s}_k$
end

- Alternative formula for β_{k+1} is

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

Example: Conjugate Gradient Method

- Use CG method to minimize $f(\boldsymbol{x}) = 0.5x_1^2 + 2.5x_2^2$
- Gradient is given by $\nabla f(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$
- Taking $\boldsymbol{x}_0 = \begin{bmatrix} 5 & 1 \end{bmatrix}^T$, initial search direction is negative gradient,

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

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

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

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

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

  which gives as next search direction

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

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

cg4.m demo

Truncated Newton Methods

- Another way to reduce work in Newton-like methods is to solve linear system for Newton step by iterative method
- Small number of iterations may suffice to produce step as useful as true Newton step, especially far from overall solution, where true Newton step may be unreliable anyway
- Good choice for linear iterative solver is CG method, which gives step intermediate between steepest descent and Newton-like step
- Since only matrix-vector products are required, explicit formation of Hessian matrix can be avoided by using finite difference of gradient along given vector

Nonlinear Least Squares

Nonlinear Least Squares

A bit of notation...

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

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

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

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

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

- Here, $A = A(\mathbf{x})$ is a function of the unknown parameters $\mathbf{x} = [x_1\, x_2\, \ldots\, x_n]^T$.

- In many cases, cannot readily write residual evaluation as $\mathbf{r} = \mathbf{b} - A\mathbf{x}$.

- Instead, seek parameters x_j, $j = 1, \ldots, n$ that minimize the 2-norm of the residual vector with components

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

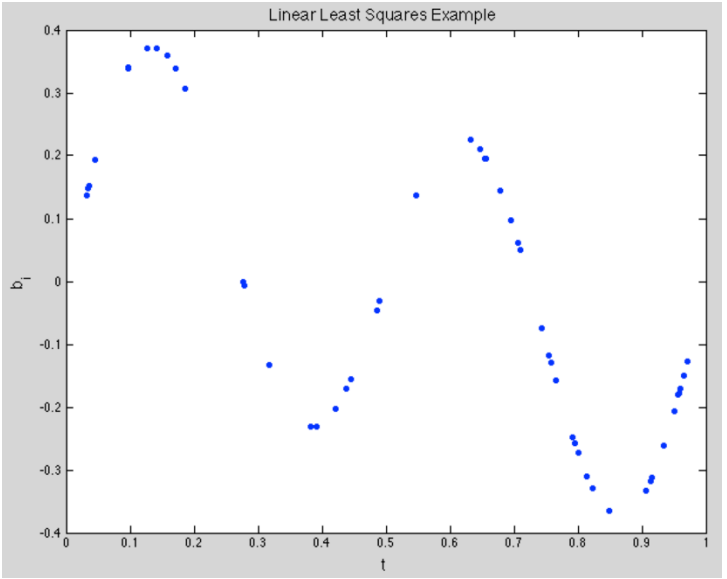
  with $f(t, \mathbf{x})$ a nonlinear function of $\mathbf{x}$.

Examples: 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

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



Examples: 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*

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

- Note: The text uses $\mathbf{y} := \mathbf{b}$ (which is different from the role of $\mathbf{y}$ in Chapter 3.)

Nonlinear Least Squares

- Given data (t_i, y_i), find vector $\boldsymbol{x}$ of parameters that gives "best fit" in least squares sense to model function $f(t, \boldsymbol{x})$, where f is nonlinear function of $\boldsymbol{x}$
- Define components of *residual function*

$$r_i(\boldsymbol{x}) = y_i - f(t_i, \boldsymbol{x}), \quad i = 1, \ldots, m$$

so we want to minimize $\phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{r}^T(\boldsymbol{x})\boldsymbol{r}(\boldsymbol{x})$

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

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

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

Nonlinear Least Squares

- Given data (t_i, y_i), find vector $\boldsymbol{x}$ of parameters that gives "best fit" in least squares sense to model function $f(t, \boldsymbol{x})$, where f is nonlinear function of $\boldsymbol{x}$
- Define components of *residual function*

$$r_i(\boldsymbol{x}) = y_i - f(t_i, \boldsymbol{x}), \quad i = 1, \ldots, m$$

so we want to minimize $\phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{r}^T(\boldsymbol{x})\boldsymbol{r}(\boldsymbol{x})$

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

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

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

Nonlinear Least Squares, continued

- Linear system for Newton step is

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

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

Gauss-Newton Method

- This motivates Gauss-Newton method for nonlinear least squares, in which second-order term is dropped and linear system

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

  is solved for approximate Newton step $\boldsymbol{s}_k$ at each iteration

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

$$\boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{s}_k \cong -\boldsymbol{r}(\boldsymbol{x}_k)$$

  which can be solved better by QR factorization

- Next approximate solution is then given by

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

  and process is repeated until convergence

Nonlinear Least Squares

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

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

  with

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

- Hessian:

$$
\begin{aligned}
H_\phi(\mathbf{x}) &= \frac{\partial}{\partial x_k} \nabla \phi \\
&= \frac{\partial}{\partial x_k} \left(\sum_{i=1}^{m} \frac{\partial r_i}{\partial x_j} r_i \right) \\
H_{\phi,jk} &= \sum_{i=1}^{m} \frac{\partial r_i}{\partial x_j} \frac{\partial r_i}{\partial x_k} + \sum_{i=1}^{m} r_i \frac{\partial^2 r_i}{\partial x_j x_k} \\
H_\phi(\mathbf{x}) &= J^T(\mathbf{x}) J(\mathbf{x}) + \sum_{i=1}^{m} r_i(\mathbf{x}) H_i(\mathbf{x}).
\end{aligned}
$$

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

- **Newton Step:**

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{s}_k \\ H_\phi \mathbf{s}_k &= -J_k^T \mathbf{r}_k \end{aligned}$$

- Search direction via approximate Hessian:

$$\begin{aligned} \left(J_k^T J_k + \sum_{i=1}^{m} r_i H_{r_i} \right) \mathbf{s}_k &= -J(\mathbf{x})^T \mathbf{r}_k \\ \left(J_k^T J_k \right) \mathbf{s}_k &= -J(\mathbf{x})^T \mathbf{r}_k \end{aligned}$$

- Equivalent to normal equations, linear least squares:

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

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

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

nllsq_test.m

Nonlinear Least Squares

$r_i = y_i - f(t_i, \vec{x})$

t parameters

Ex: $f(t, \vec{x}) = x_1 e^{x_2 t}$

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

$$\nabla \phi \stackrel{?}{=} J^T \underline{r}$$

$$\nabla \phi = \frac{\partial}{\partial x_j} \phi(\underline{r}(x))$$

$$= \frac{\partial}{\partial x_j} \left[\frac{1}{2} \sum_{i=1}^{m} r_i^2 \right]$$

$$= \sum_{i=1}^{m} r_i \frac{\partial r_i}{\partial x_j} = \sum_{i=1}^{m} \frac{\partial r_i}{\partial x_j} r_i = J^T \underline{r}$$

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

$$H_\phi(x) \stackrel{?}{=} \frac{\partial}{\partial x_k} \nabla \phi$$

$$= \frac{\partial}{\partial x_k} \left(\sum_{i=1}^{m} \frac{\partial r_i}{\partial x_j} r_i \right)$$

$$H_{jk} = \sum_{i=1}^{m} \frac{\partial r_i}{\partial x_j} \frac{\partial r_i}{\partial x_k} + \sum_{i=1}^{m} r_i \frac{\partial^2 r_i}{\partial x_j \partial x_k}$$

$$H = J^T J + \sum_{i=1}^{m} r_i H_{r_i}$$

Nonlinear Least Squares

Newton Step!

$$x_{k+1} = x_k + s_k$$

$$H_\phi s_k = -J_k^T r_k$$

$$\left(J_k^T J_k + \sum_{i=1}^{m} r_i H_{r_i}\right) s_k = -J_k^T r_k$$

(Ignore)

$$\left(J_k^T J_k\right) s_k = -J_k r_k$$

$J_k s_k \cong -r_k$ Solve via $QR = J$

$$Q_1^T Q_1 R s_k = -Q_1^T r_k$$

$$s_k = -R^{-1} Q_1^T r_k$$

$$x_{k+1} = x_k + s_k$$

Example: Gauss-Newton Method

- Use Gauss-Newton method to fit nonlinear model function

$$f(t, \boldsymbol{x}) = x_1 \exp(x_2 t)$$

to data

t	0.0	1.0	2.0	3.0
y	2.0	0.7	0.3	0.1

- For this model function, entries of Jacobian matrix of residual function $\boldsymbol{r}$ are given by

$$\{\boldsymbol{J}(\boldsymbol{x})\}_{i,1} = \frac{\partial r_i(\boldsymbol{x})}{\partial x_1} = -\exp(x_2 t_i)$$

$$\{\boldsymbol{J}(\boldsymbol{x})\}_{i,2} = \frac{\partial r_i(\boldsymbol{x})}{\partial x_2} = -x_1 t_i \exp(x_2 t_i)$$

Example, continued

- If we take $\boldsymbol{x}_0 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$, then Gauss-Newton step $\boldsymbol{s}_0$ is given by linear least squares problem

$$\begin{bmatrix} -1 & 0 \\ -1 & -1 \\ -1 & -2 \\ -1 & -3 \end{bmatrix} \boldsymbol{s}_0 \cong \begin{bmatrix} -1 \\ 0.3 \\ 0.7 \\ 0.9 \end{bmatrix}$$

  whose solution is $\boldsymbol{s}_0 = \begin{bmatrix} 0.69 \\ -0.61 \end{bmatrix}$

- Then next approximate solution is given by $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s}_0$, and process is repeated until convergence

Example, continued

$\boldsymbol{x}_k$		$\|\boldsymbol{r}(\boldsymbol{x}_k)\|_2^2$
1.000	0.000	2.390
1.690	−0.610	0.212
1.975	−0.930	0.007
1.994	−1.004	0.002
1.995	−1.009	0.002
1.995	−1.010	0.002

< interactive example >

Gauss-Newton Method, continued

- Gauss-Newton method replaces nonlinear least squares problem by sequence of linear least squares 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 method that takes into account true full Hessian matrix

Levenberg-Marquardt Method

- Levenberg-Marquardt method is another useful alternative when Gauss-Newton approximation is inadequate or yields rank deficient linear least squares subproblem
- In this method, linear system at each iteration is of form

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

  where μ_k is scalar parameter chosen by some strategy
- Corresponding linear least squares problem is

$$\begin{bmatrix} \boldsymbol{J}(\boldsymbol{x}_k) \\ \sqrt{\mu_k}\boldsymbol{I} \end{bmatrix} \boldsymbol{s}_k \cong \begin{bmatrix} -\boldsymbol{r}(\boldsymbol{x}_k) \\ \boldsymbol{0} \end{bmatrix}$$

- With suitable strategy for choosing μ_k, this method can be very robust in practice, and it forms basis for several effective software packages < interactive example >

Levenberg-Marquardt

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

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

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

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

- Levenberg-Marquardt is a combination of these two:

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

  which, for large μ_k, corresponds to steepest descent.

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

levenberg.m

Equality-Constrained Optimization

- For equality-constrained minimization problem

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

where $f\colon \mathbb{R}^n \to \mathbb{R}$ and $\boldsymbol{g}\colon \mathbb{R}^n \to \mathbb{R}^m$, with $m \leq n$, we seek critical point of Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{g}(\boldsymbol{x})$

- Applying Newton's method to nonlinear system

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

we obtain linear system

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

for Newton step $(\boldsymbol{s}, \boldsymbol{\delta})$ in $(\boldsymbol{x}, \boldsymbol{\lambda})$ at each iteration

Constrained Optimality, continued

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

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

- Its gradient is given by

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

- Its Hessian is given by

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

  where

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

Sequential Quadratic Programming

- Foregoing block 2×2 linear system is equivalent to quadratic programming problem, so this approach is known as *sequential quadratic programming*
- Types of solution methods include
  - *Direct solution* methods, in which entire block 2×2 system is solved directly
  - *Range space* methods, based on block elimination in block 2×2 linear system
  - *Null space* methods, based on orthogonal factorization of matrix of constraint normals, $\boldsymbol{J}_g^T(\boldsymbol{x})$

Direct Method:

Solve 2 x 2 Block System via Gaussian Elimination

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

Merit Function

- Once Newton step $(\boldsymbol{s}, \boldsymbol{\delta})$ determined, we need *merit function* to measure progress toward overall solution for use in line search or trust region
- Popular choices include *penalty function*

$$\phi_\rho(\boldsymbol{x}) = f(\boldsymbol{x}) + \tfrac{1}{2}\,\rho\,\boldsymbol{g}(\boldsymbol{x})^T\boldsymbol{g}(\boldsymbol{x})$$

and *augmented Lagrangian function*

$$\mathcal{L}_\rho(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T\boldsymbol{g}(\boldsymbol{x}) + \tfrac{1}{2}\,\rho\,\boldsymbol{g}(\boldsymbol{x})^T\boldsymbol{g}(\boldsymbol{x})$$

where parameter $\rho > 0$ determines relative weighting of optimality vs feasibility

- Given starting guess $\boldsymbol{x}_0$, good starting guess for $\boldsymbol{\lambda}_0$ can be obtained from least squares problem

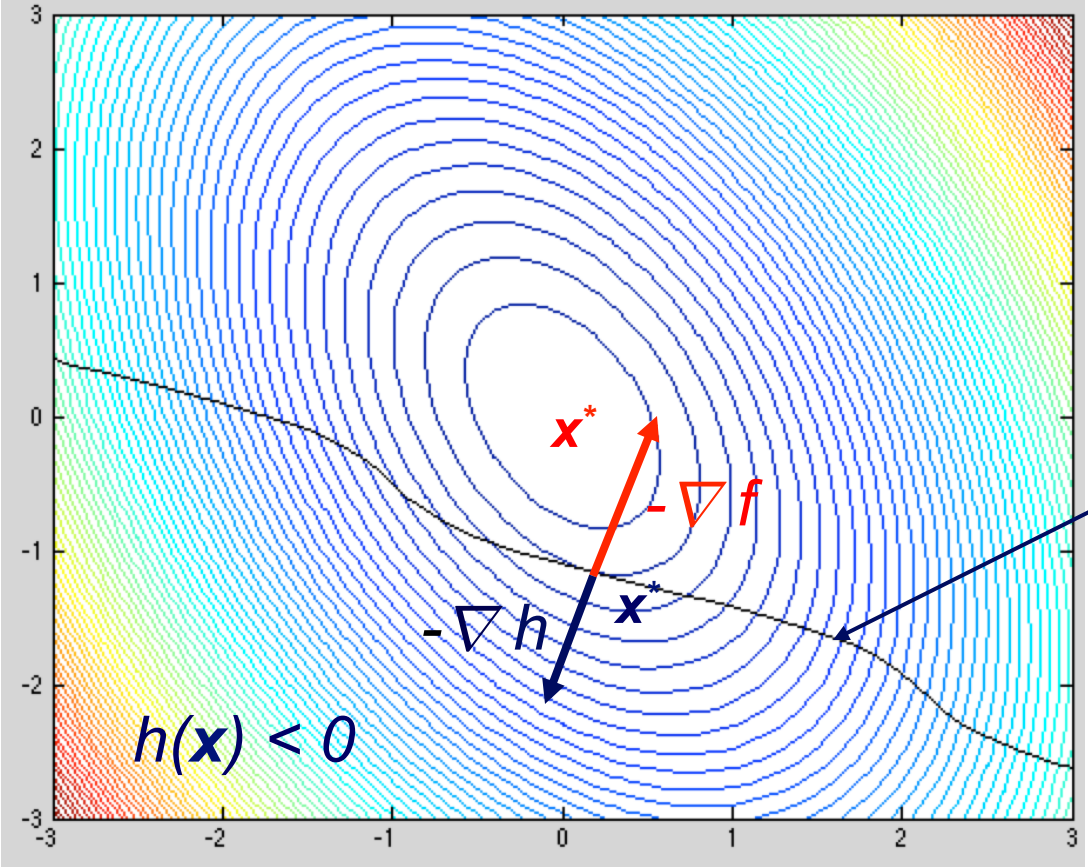
$$\boldsymbol{J}_g^T(\boldsymbol{x}_0)\,\boldsymbol{\lambda}_0 \cong -\nabla f(\boldsymbol{x}_0)$$

Inequality-Constrained Optimization

- Methods just outlined for equality constraints can be extended to handle inequality constraints by using *active set* strategy
- Inequality constraints are provisionally divided into those that are satisfied already (and can therefore be temporarily disregarded) and those that are violated (and are therefore temporarily treated as equality constraints)
- This division of constraints is revised as iterations proceed until eventually correct constraints are identified that are binding at solution

Active Constraint

- If constraint is active $(h(x^*) = 0)$ then ∇f and ∇h point in opposite directions.
- $\nabla f + \lambda^* \nabla h = 0.$
- $\lambda^* > 0.$



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

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

Penalty Methods

- Merit function can also be used to convert equality-constrained problem into sequence of unconstrained problems
- If $\boldsymbol{x}_\rho^*$ is solution to

$$\min_{\boldsymbol{x}} \phi_\rho(\boldsymbol{x}) = f(\boldsymbol{x}) + \tfrac{1}{2}\,\rho\,\boldsymbol{g}(\boldsymbol{x})^T\boldsymbol{g}(\boldsymbol{x})$$

  then under appropriate conditions

$$\lim_{\rho\to\infty} \boldsymbol{x}_\rho^* = \boldsymbol{x}^*$$

- This enables use of unconstrained optimization methods, but problem becomes ill-conditioned for large ρ, so we solve sequence of problems with gradually increasing values of ρ, with minimum for each problem used as starting point for next problem < interactive example >

Barrier Methods

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

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

or

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

which increasingly penalize feasible points as they approach boundary of feasible region

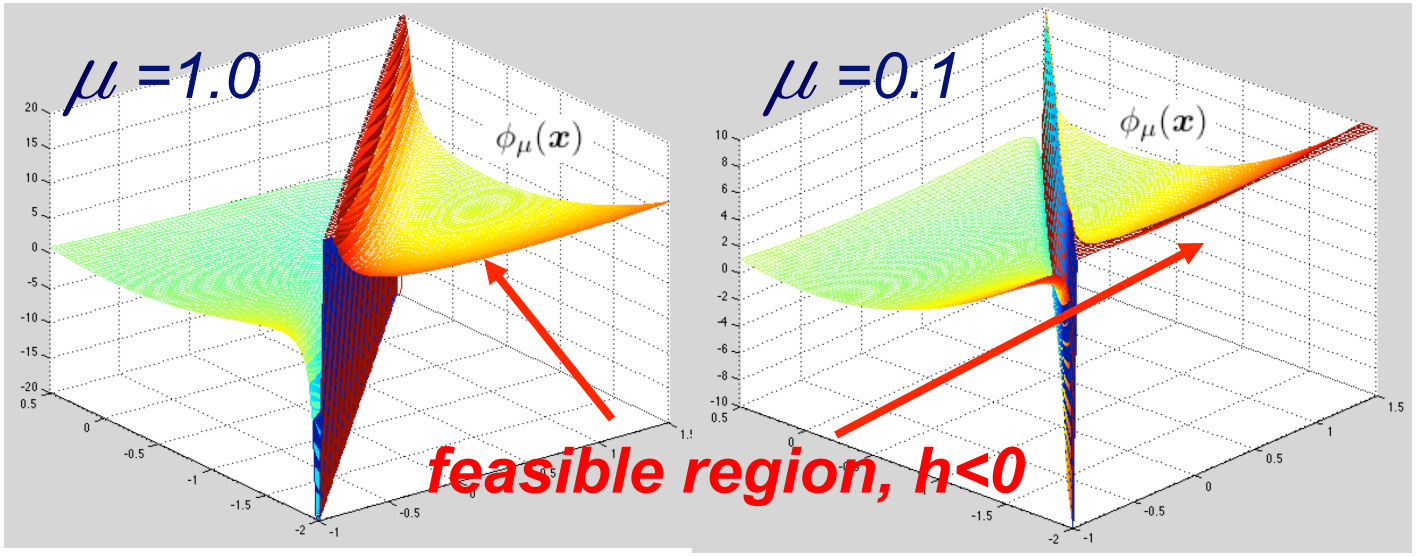
- Again, solutions of unconstrained problem approach $\boldsymbol{x}^*$ as $\mu \to 0$, but problems are increasingly ill-conditioned, so solve sequence of problems with decreasing values of μ
- Barrier functions are basis for *interior point* methods for linear programming

Barrier Functions for Inequality Constraints

- Solve modified unconstrained minimization problem,

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

- ***Starting in the feasible region!***
- As $\mu \rightarrow 0$ it appears that the constraint is weaker.
- However, the constraint goes to ∞ for any $\mu \neq 0$, so a small μ simply means the fence is *steeper,* which makes the minimization problem more challenging.



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

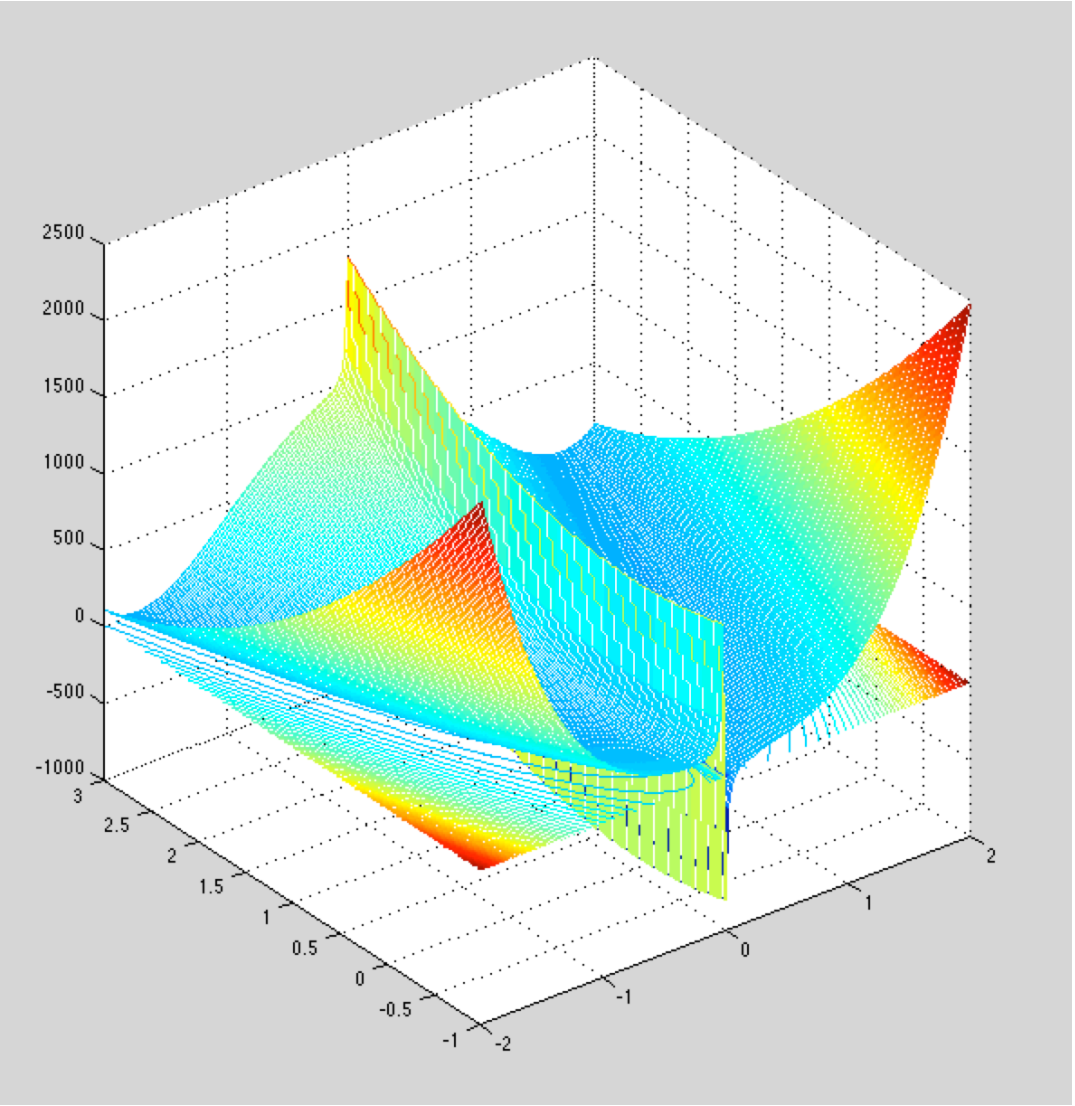
$$h(\mathbf{x}) = 1 - x_1 + x_2 < 0$$

Barrier Demo

- *With barrier methods, simply augment the original (unconstrained) objective function, f(**x**), with the penalty function.*

- *Then, start in the feasible region.*

cg_barrier.m



Example: Constrained Optimization

- Consider quadratic programming problem

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

subject to

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

- Lagrangian function is given by

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

- Since

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

we have

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

Example, continued

- So system to be solved for critical point of Lagrangian is

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

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 solution

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

Example, continued