

# IFT 6112

## BACKGROUND: OPTIMIZATION

<http://www-labs.iro.umontreal.ca/~bmpix/teaching/6112/2018/>



Mikhail Bessmeltsev

# Motivation

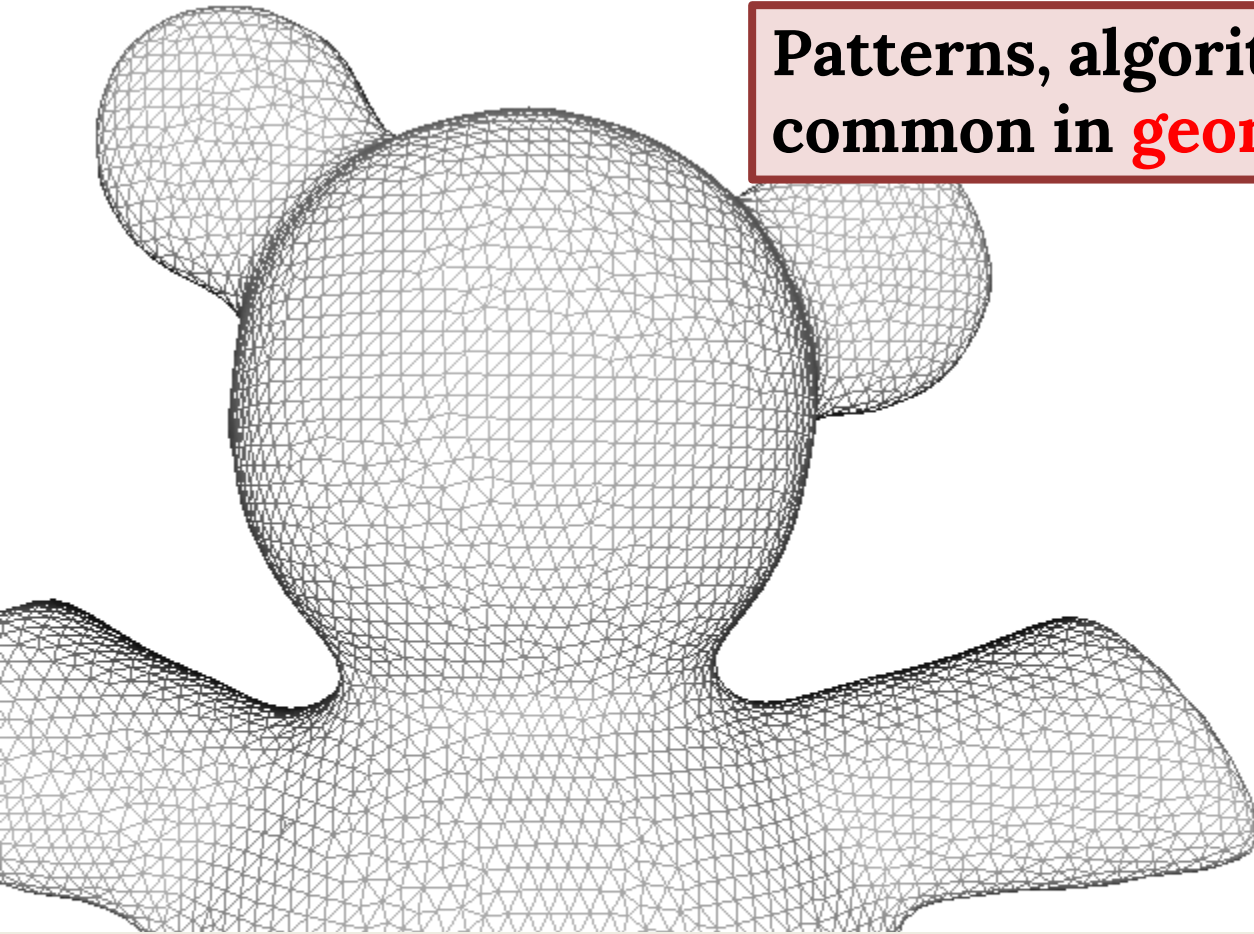
Numerical problems are everywhere  
in geometric modeling!

## Quick summary!

*Mostly for common ground: You may already know this material.  
First half is important; remainder summarizes interesting recent tools.*

# Our Bias

Patterns, algorithms, & examples  
common in **geometry**.



**Numerical analysis is a huge field.**

# Rough Plan

- Linear problems
- Unconstrained optimization
- Equality-constrained optimization
- Variational problems

# Rough Plan

- **Linear problems**
- Unconstrained optimization
- Equality-constrained optimization
- Variational problems

# Vector Spaces and Linear Operators

$$\mathcal{L}[\vec{x} + \vec{y}] = \mathcal{L}[\vec{x}] + \mathcal{L}[\vec{y}]$$

$$\mathcal{L}[c\vec{x}] = c\mathcal{L}[\vec{x}]$$

# Abstract Example

$$C^\infty(\mathbb{R})$$

$$\mathcal{L}[f] := df/dx$$

**Eigenvectors?**

# In Finite Dimensions

$A$        $\vec{x}$   
matrix    vector

$\vec{x} \mapsto A\vec{x}$   
linear operator



# Linear System of Equations

$$\begin{pmatrix} A \end{pmatrix} \begin{pmatrix} \vec{x} \end{pmatrix} = \begin{pmatrix} \vec{b} \end{pmatrix}$$

**Simple “inverse problem”**

# Common Strategies

- **Gaussian elimination**
  - $O(n^3)$  time to solve  $Ax=b$  or to invert
- **But:** Inversion is unstable and slower!
- **Never ever compute  $A^{-1}$  if you can avoid it.**

# Simple Example

$$\frac{d^2 f}{dx^2} = g, f(0) = f(1) = 0$$

$$\begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & & \ddots & & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_n \end{pmatrix}$$



# Linear Solver Considerations

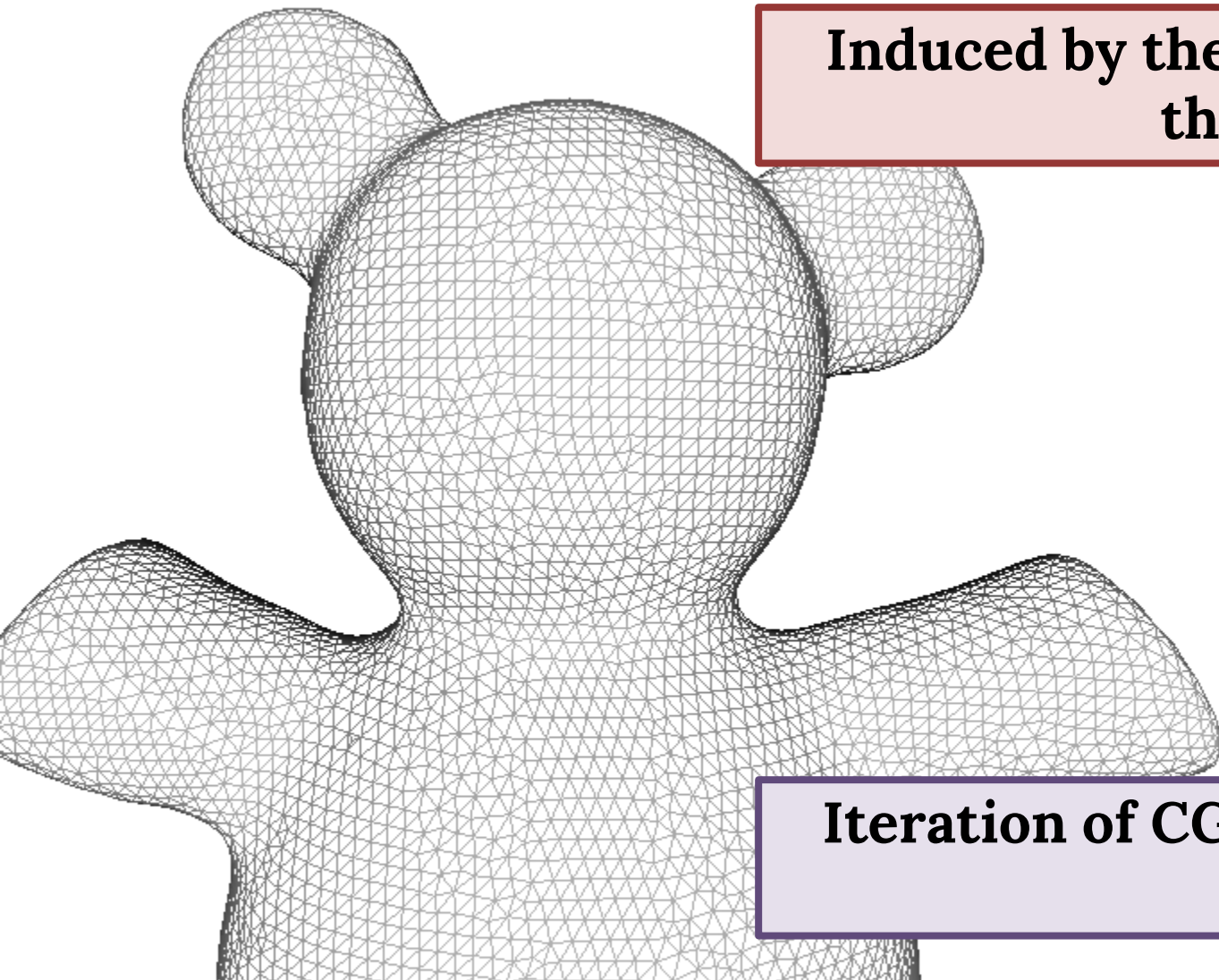
- **Never construct  $A^{-1}$  explicitly**  
*(if you can avoid it)*
- **Added structure helps**  
Sparsity, symmetry, positive definiteness,  
bandedness

$$\text{inv}(A) * b \ll (A' * A) \setminus (A' * b) \ll A \setminus b$$

# Two Classes of Solvers

- **Direct** (*explicit matrix*)
  - **Dense**: Gaussian elimination/LU, QR for least-squares
  - **Sparse**: Reordering (SuiteSparse, Eigen)
- **Iterative** (*apply matrix repeatedly*)
  - **Positive definite**: Conjugate gradients
  - **Symmetric**: MINRES, GMRES
  - **Generic**: LSQR

# Very Common: Sparsity

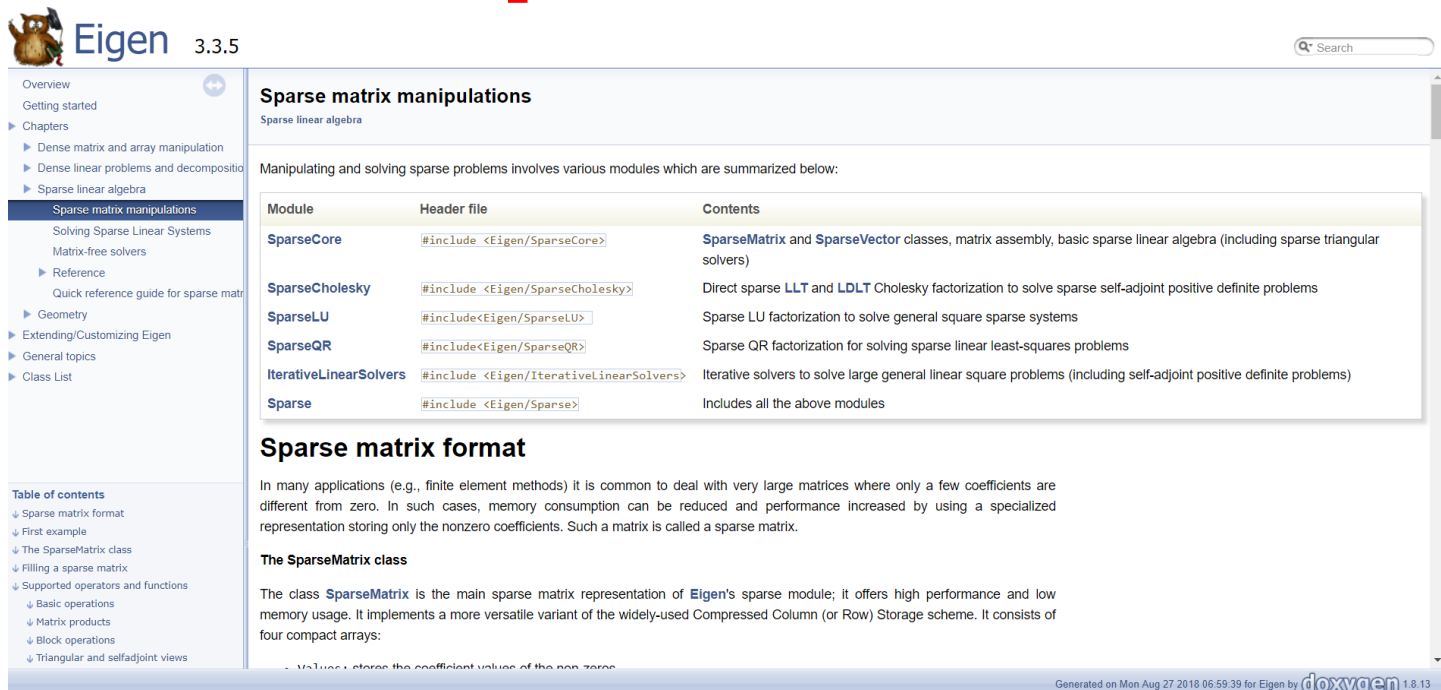


Induced by the **connectivity** of the triangle mesh.

Iteration of CG has local effect  
⇒ **Precondition!**

# For IFT 6112

- **No need to implement** a linear solver
- **If a matrix is sparse, your code should store it as a sparse matrix!**



The screenshot shows the Eigen 3.3.5 documentation page. The main heading is "Sparse matrix manipulations" under the sub-heading "Sparse linear algebra". A table lists various modules and their contents:

Module	Header file	Contents
<b>SparseCore</b>	<code>#include &lt;Eigen/SparseCore&gt;</code>	<b>SparseMatrix</b> and <b>SparseVector</b> classes, matrix assembly, basic sparse linear algebra (including sparse triangular solvers)
<b>SparseCholesky</b>	<code>#include &lt;Eigen/SparseCholesky&gt;</code>	Direct sparse <b>LLT</b> and <b>LDLT</b> Cholesky factorization to solve sparse self-adjoint positive definite problems
<b>SparseLU</b>	<code>#include &lt;Eigen/SparseLU&gt;</code>	Sparse LU factorization to solve general square sparse systems
<b>SparseQR</b>	<code>#include &lt;Eigen/SparseQR&gt;</code>	Sparse QR factorization for solving sparse linear least-squares problems
<b>IterativeLinearSolvers</b>	<code>#include &lt;Eigen/IterativeLinearSolvers&gt;</code>	Iterative solvers to solve large general linear square problems (including self-adjoint positive definite problems)
<b>Sparse</b>	<code>#include &lt;Eigen/Sparse&gt;</code>	Includes all the above modules

Below the table, there is a section titled "Sparse matrix format" which explains that in many applications (e.g., finite element methods) it is common to deal with very large matrices where only a few coefficients are different from zero. In such cases, memory consumption can be reduced and performance increased by using a specialized representation storing only the nonzero coefficients. Such a matrix is called a sparse matrix.

The section "The SparseMatrix class" states that the class **SparseMatrix** is the main sparse matrix representation of Eigen's sparse module; it offers high performance and low memory usage. It implements a more versatile variant of the widely-used Compressed Column (or Row) Storage scheme. It consists of four compact arrays:

- `m_values`: store the coefficient values of the non-zero

The footer of the page indicates it was generated on Mon Aug 27 2018 06:59:39 for Eigen by **doxygen** 1.8.13.

[https://eigen.tuxfamily.org/dox/group\\_\\_TutorialSparse.html](https://eigen.tuxfamily.org/dox/group__TutorialSparse.html)



# Optimization Terminology

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\text{s.t. } g(x) = 0$$

$$h(x) \geq 0$$

**Objective (“Energy Function”)**

# Optimization Terminology

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\text{s.t. } g(x) = 0$$

$$h(x) \geq 0$$

**Equality Constraints**

# Optimization Terminology

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\text{s.t. } g(x) = 0$$

$$h(x) \geq 0$$

**Inequality Constraints**

# Encapsulates Many Problems

$$\begin{aligned} \min_{x \in \mathbb{R}^n} f(x) \\ \text{s.t. } g(x) = 0 \\ h(x) \geq 0 \end{aligned}$$

$$Ax = b \leftrightarrow f(x) = \|Ax - b\|_2$$

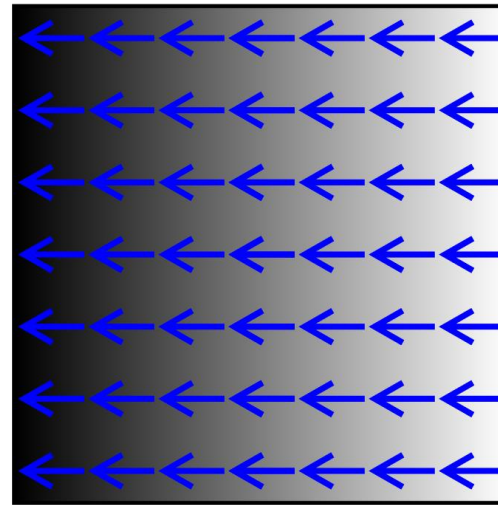
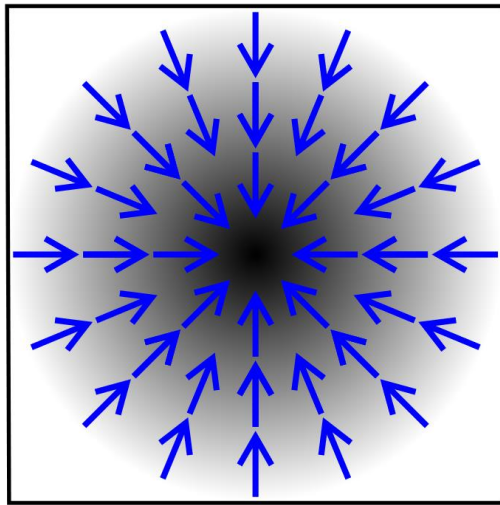
$$Ax = \lambda x \leftrightarrow f(x) = \|Ax\|_2, g(x) = \|x\|_2 - 1$$

$$\text{Roots of } g(x) \leftrightarrow f(x) = 0$$

# Notions from Calculus

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$\rightarrow \nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$$



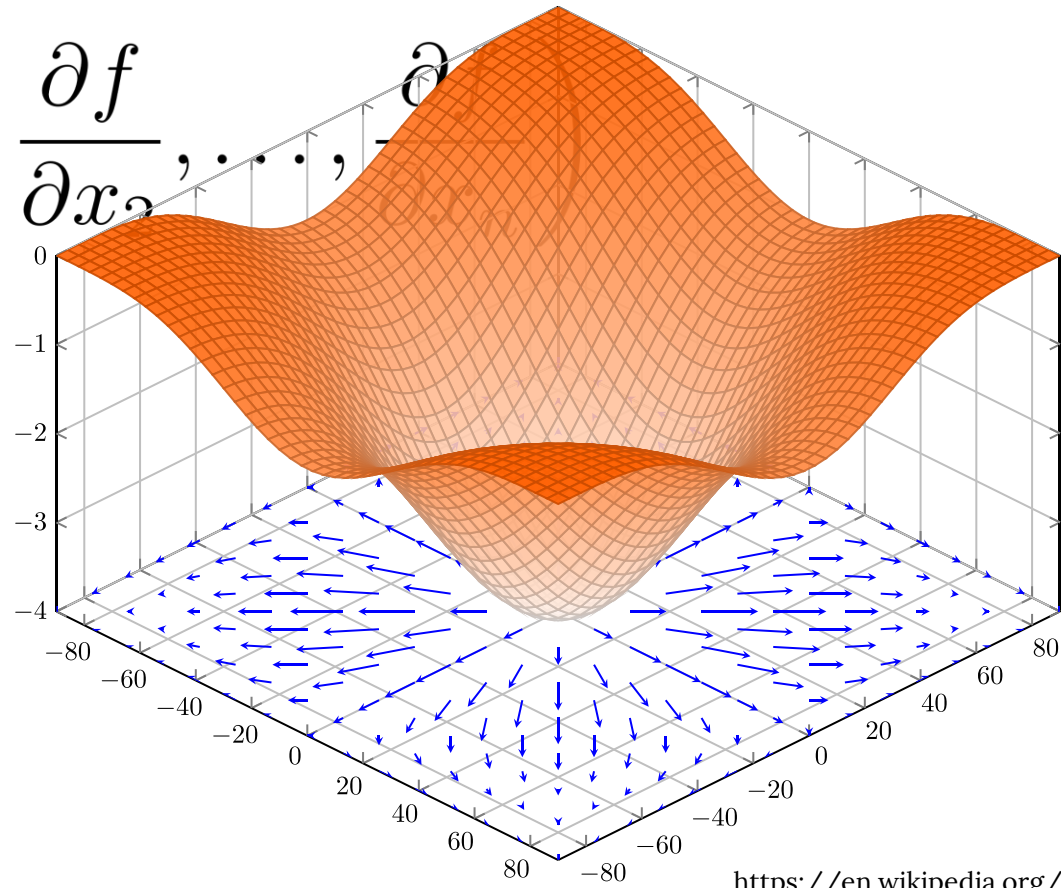
<https://en.wikipedia.org/?title=Gradient>

# Gradient

# Notions from Calculus

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

$$\rightarrow \nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$$



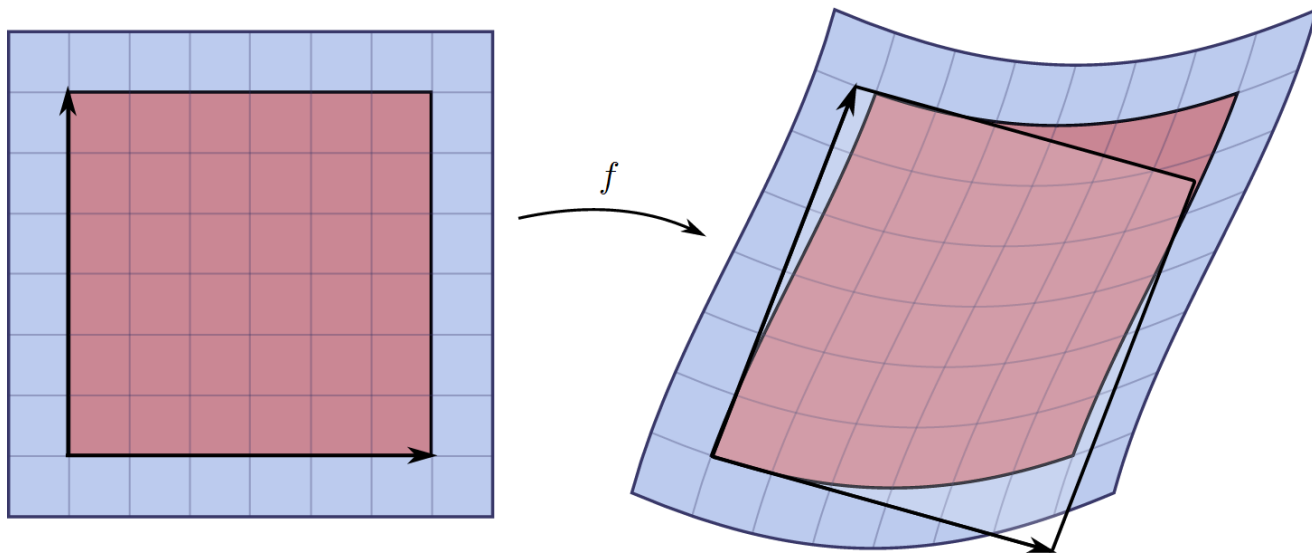
<https://en.wikipedia.org/?title=Gradient>

# Gradient

# Notions from Calculus

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\rightarrow (Df)_{ij} = \frac{\partial f_i}{\partial x_j}$$

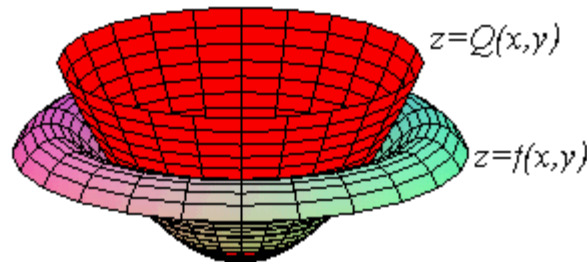


[https://en.wikipedia.org/wiki/Jacobian\\_matrix\\_and\\_determinant](https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant)

# Jacobian

# Notions from Calculus

$$f : \mathbb{R}^n \rightarrow \mathbb{R} \rightarrow H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$



$$f(x) \approx f(x_0) + \nabla f(x_0)^\top (x - x_0) + (x - x_0)^\top H f(x_0) (x - x_0)$$

<http://math.etsu.edu/multicalc/prealpha/Chap2/Chap2-5/10-3a-t3.gif>

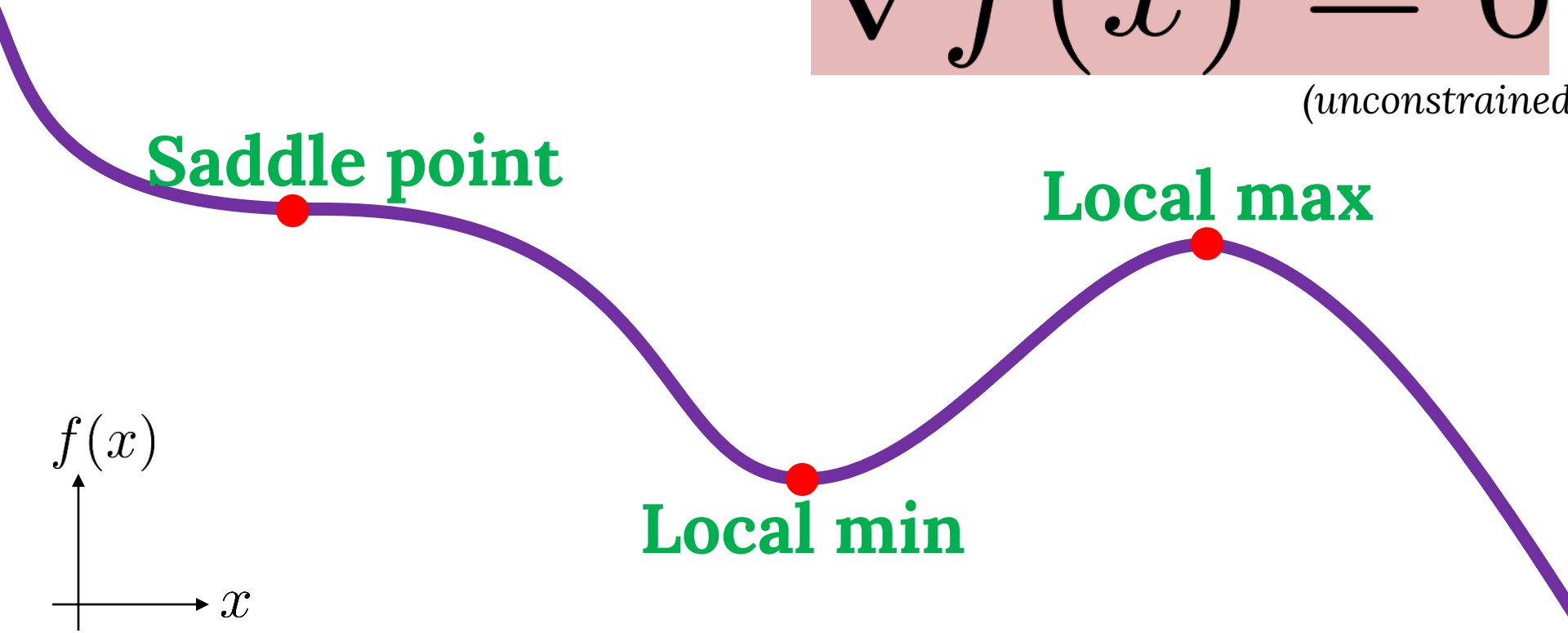
# Hessian



# Optimization to Root-Finding

$$\nabla f(x) = 0$$

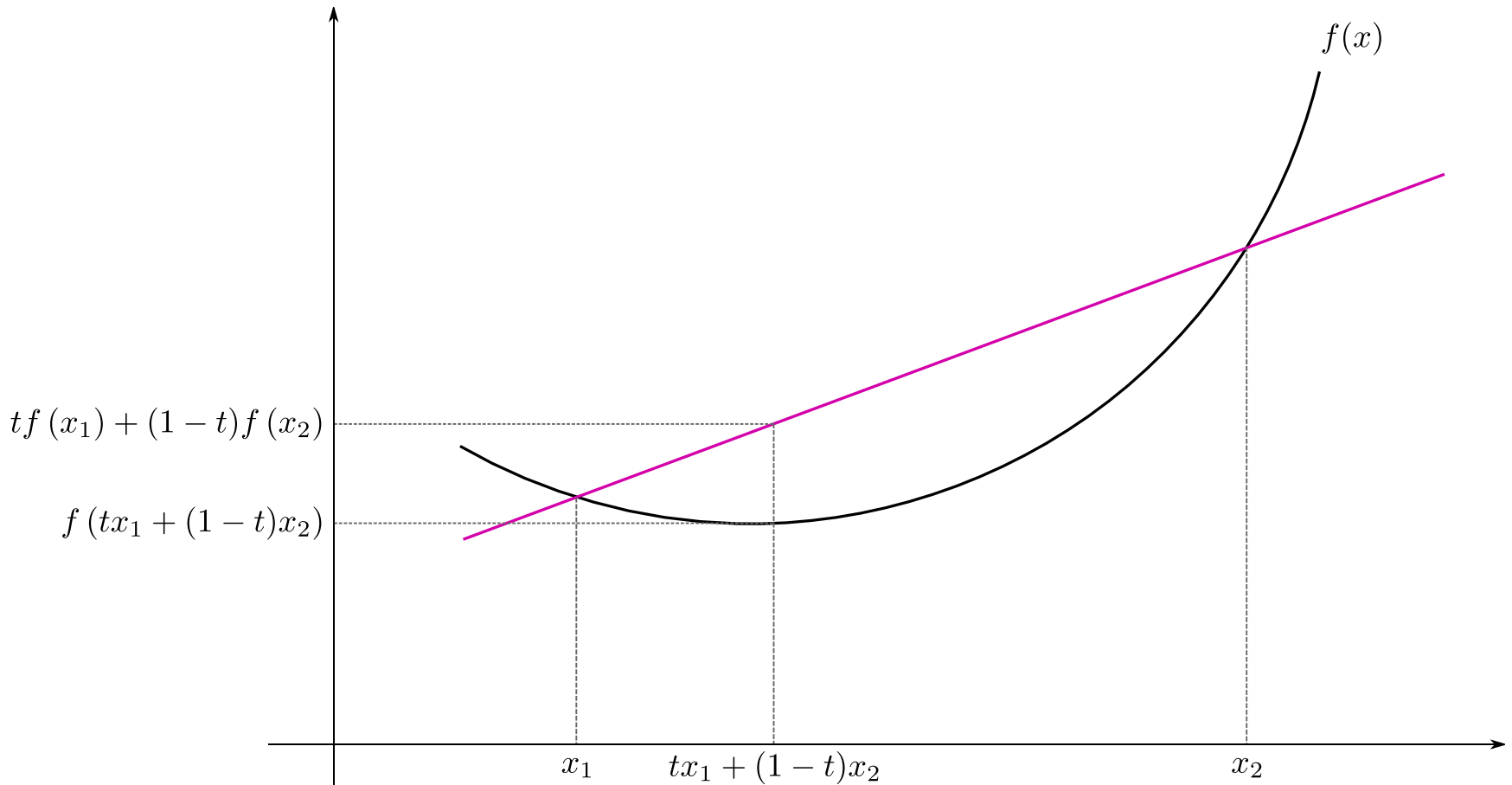
(unconstrained)



**Critical point**

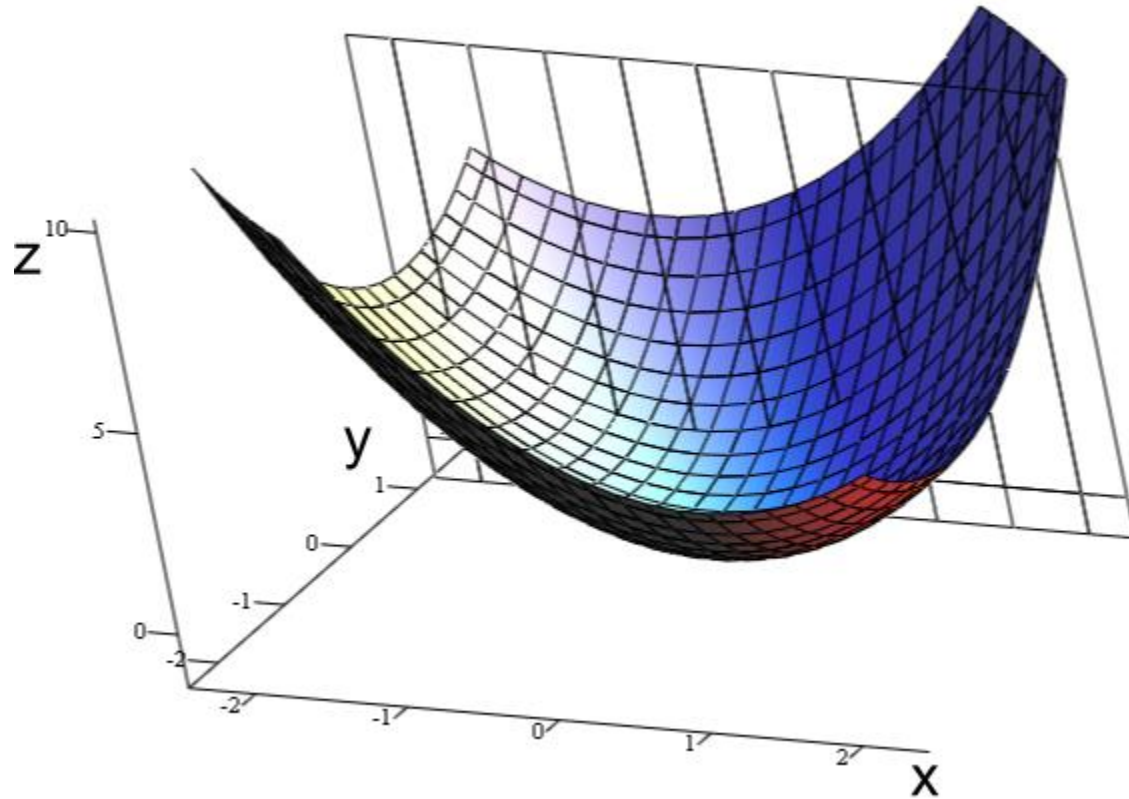
# Convex Functions

$$f''(x) > 0$$



# Convex Functions

$$H(x) \geq 0$$



[https://en.wikipedia.org/wiki/Convex\\_function](https://en.wikipedia.org/wiki/Convex_function)

Generic tools are often not too effective!

# Generic Advice

Try the  
**simplest solver first.**

# Rough Plan

- Linear problems
- **Unconstrained optimization**
- Equality-constrained optimization
- Variational problems

# Unconstrained optimization

$$\min_{x \in \mathbb{R}^n} f(x)$$

Trivial when  $f(x)$  is linear

Easy when  $f(x)$  is quadratic

Hard in case of generic non-linear.

# Special Case: Least-Squares

$$\min_x \frac{1}{2} \|Ax - b\|_2^2$$

$$\rightarrow \min_x \frac{1}{2} x^\top A^\top Ax - b^\top Ax + \|b\|_2^2$$

$$\implies A^\top Ax = A^\top b$$

**Normal equations**  
(better solvers for this case!)



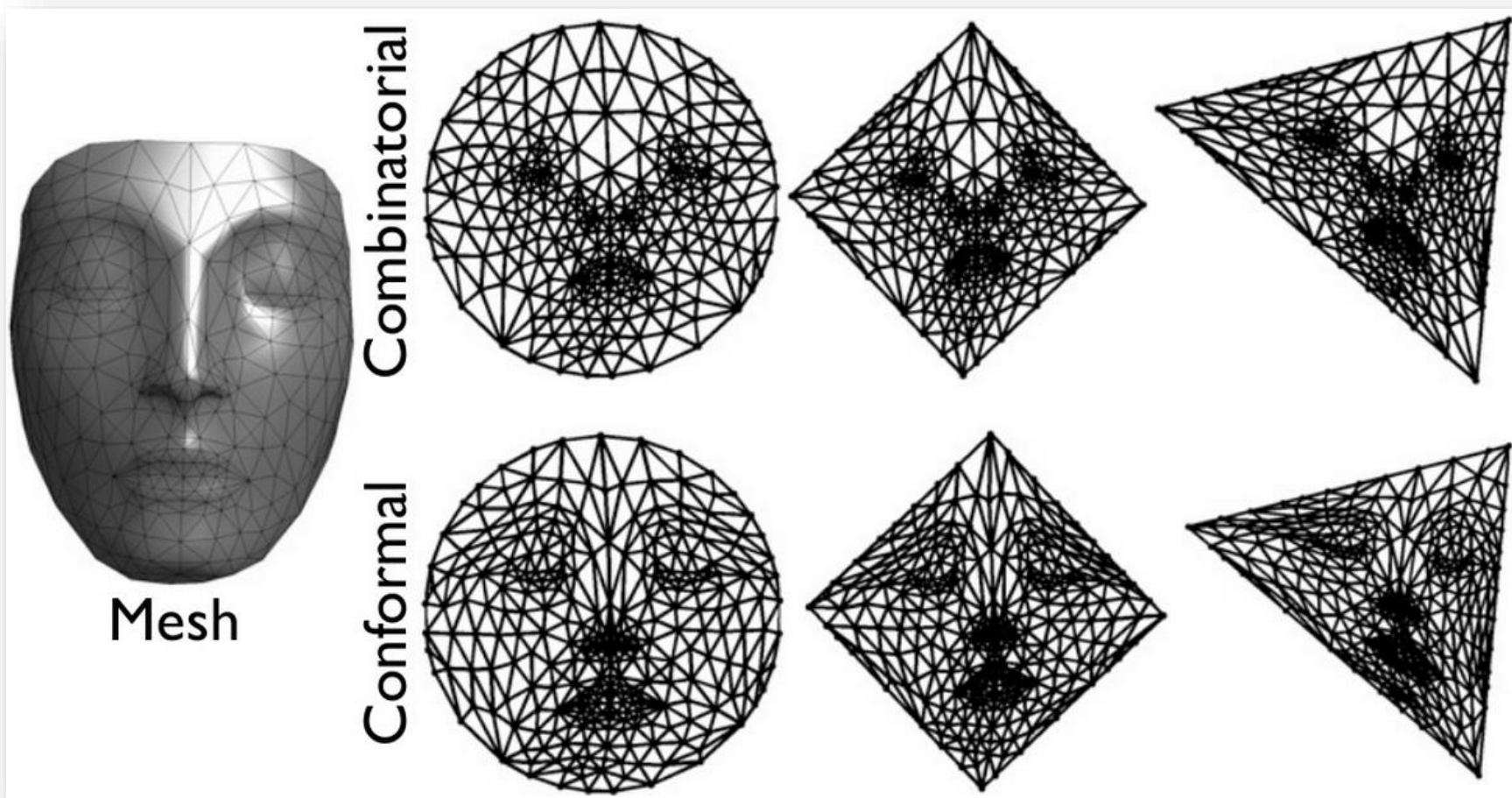
# Useful Document

## **The Matrix Cookbook**

**Petersen and Pedersen**

[http://www2.imm.dtu.dk/pubdb/views/edoc\\_download.php/3274/pdf/imm3274.pdf](http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf)

# Example: Mesh Embedding



# Linear Solve for Embedding

$$\begin{aligned} \min_{x_1, \dots, x_{|V|}} \quad & \sum_{(i,j) \in E} w_{ij} \|x_i - x_j\|_2^2 \\ \text{s.t.} \quad & x_v \text{ fixed } \forall v \in V_0 \end{aligned}$$

- $w_{ij} \equiv 1$ : Tutte embedding
- $w_{ij}$  from mesh: Harmonic embedding


**Assumption:  $w$  symmetric.**

# Rough Plan

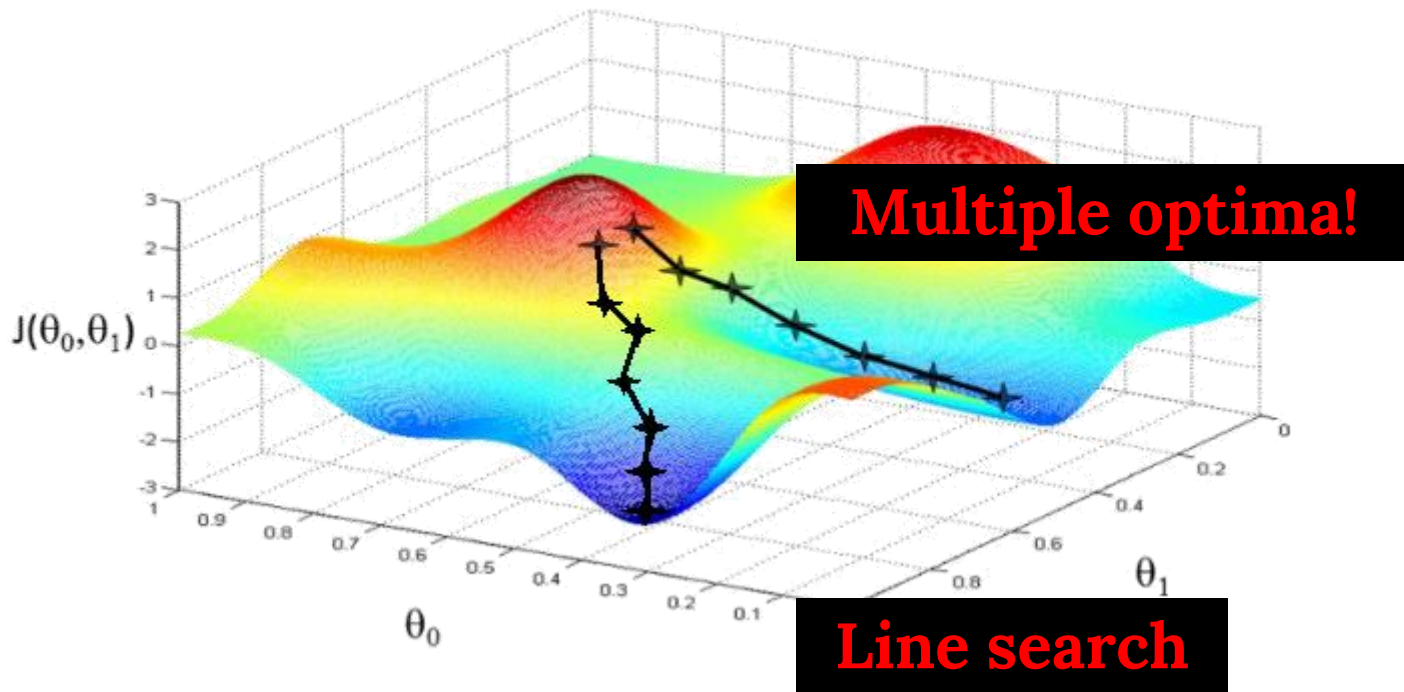
- Linear problems
- **Unconstrained optimization**
- Equality-constrained optimization
- Variational problems

# Unconstrained Optimization

$$\min_x f(x)$$

**Unstructured.**

# Basic Algorithms



$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

**Gradient descent**

# Basic Algorithms

$$\lambda_0 = 0, \lambda_s = \frac{1}{2}(1 + \sqrt{1 + 4\lambda_{s-1}^2}), \gamma_s = \frac{1 - \lambda_2}{\lambda_{s+1}}$$

$$y_{s+1} = x_s - \frac{1}{\beta} \nabla f(x_s)$$

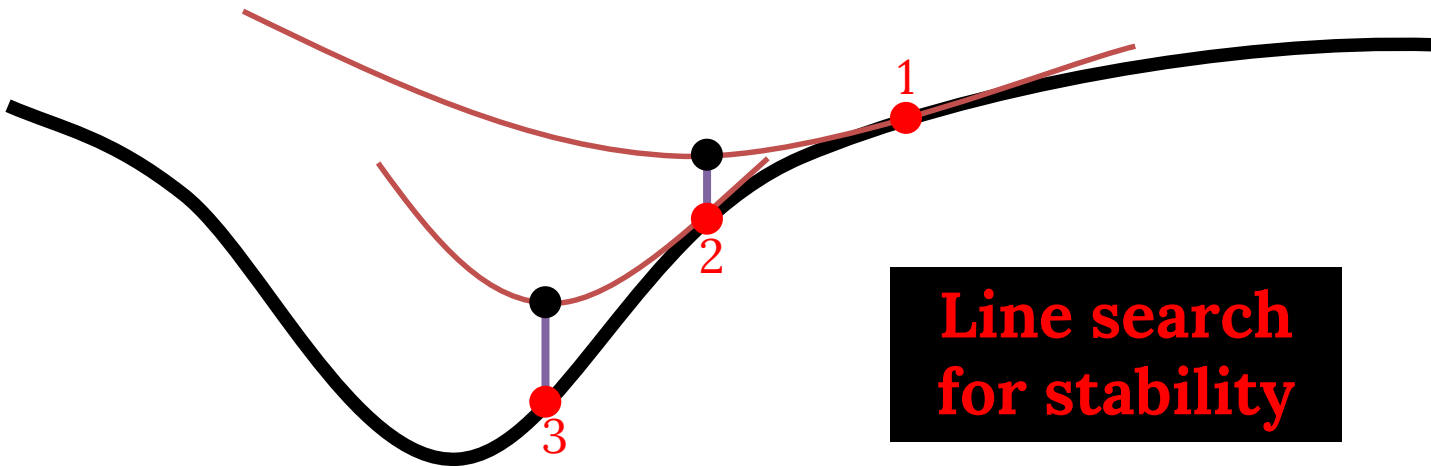
$$x_{s+1} = (1 - \gamma_s)y_{s+1} + \gamma_s y_s$$

**Quadratic convergence on convex problems!**  
(Nesterov 1983)

**Accelerated gradient descent**

# Basic Algorithms

$$x_{k+1} = x_k - [H f(x_k)]^{-1} \nabla f(x_k)$$



**Newton's Method**



# Basic Algorithms

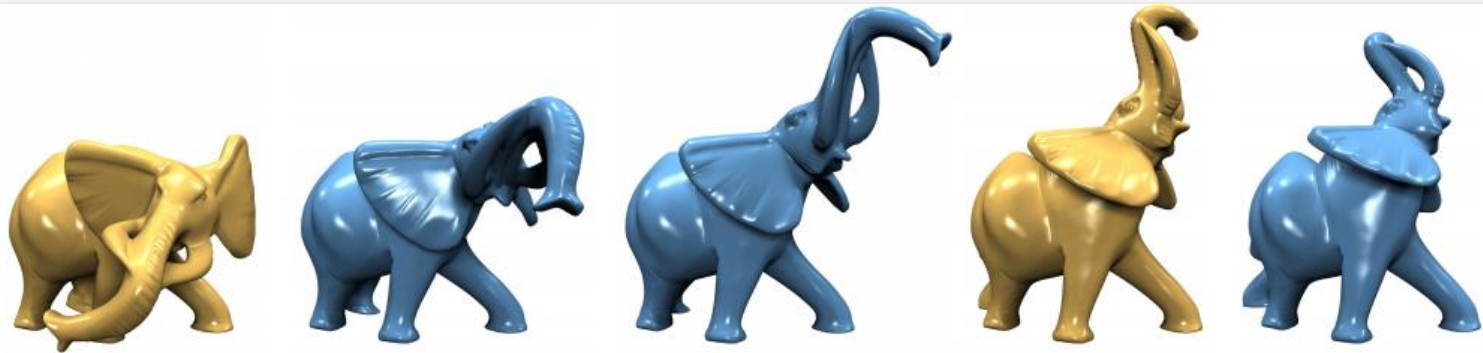
$$x_{k+1} = x_k - M_k^{-1} \nabla f(x_k)$$

**Hessian  
approximation**

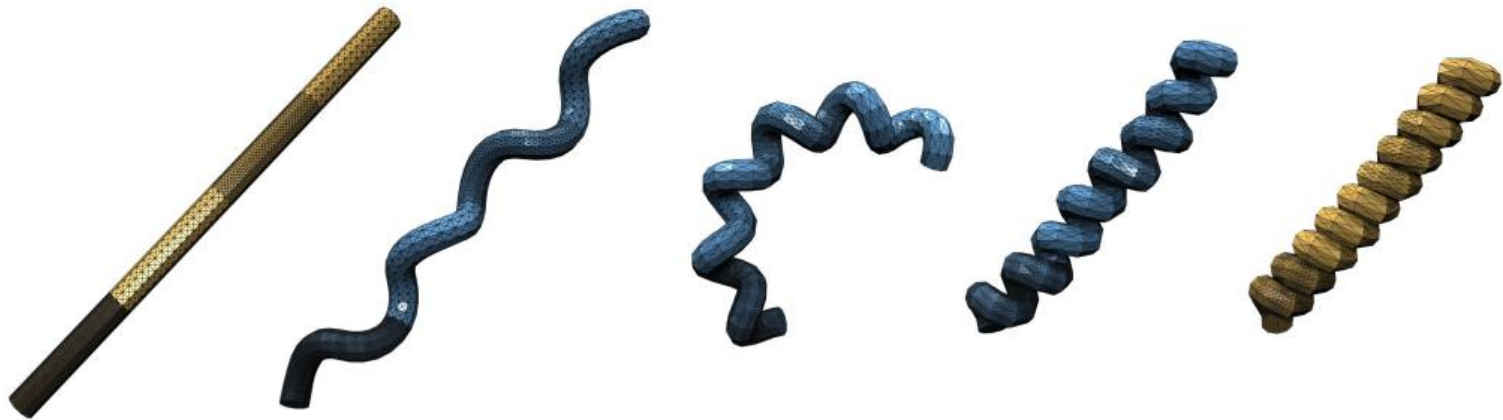
- (Often **sparse**) approximation from previous samples and gradients
- Inverse in **closed form!**

**Quasi-Newton: BFGS and friends**

# Example: Shape Interpolation



**Figure 5:** *Interpolation and extrapolation of the yellow example poses. The blending weights are 0, 0.35, 0.65, 1.0, and 1.25.*



**Figure 6:** *Interpolation of an adaptively meshed and strongly twisted helix with blending weights 0, 0.25, 0.5, 0.75, 1.0.*

# Interpolation Pipeline

Roughly:

1. **Linearly interpolate** edge lengths and dihedral angles.

$$\ell_e^* = (1 - t)\ell_e^0 + t\ell_e^1$$

$$\theta_e^* = (1 - t)\theta_e^0 + t\theta_e^1$$

2. **Nonlinear** optimization for vertex positions.

$$\min_{x_1, \dots, x_m} \lambda \sum_e w_e (\ell_e(x) - \ell_e^*)^2$$

**Sum of squares:  
Gauss-Newton**

$$+ \mu \sum_e w_b (\theta_e(x) - \theta_e^*)^2$$

# Software

- **Matlab**: `fminunc` or `minfunc`
- **C++**: `libLBFGS`, `dlib`, others

Typically provide functions for **function** and **gradient** (and optionally, **Hessian**).

**Try several!**

# Some Tricks

Lots of small elements:  $\|x\|_2^2 = \sum_i x_i^2$

Lots of zeros:  $\|x\|_1 = \sum_i |x_i|$

Uniform norm:  $\|x\|_\infty = \max_i |x_i|$

Low rank:  $\|X\|_* = \sum_i \sigma_i$

Mostly zero columns:  $\|X\|_{2,1} = \sum_j \sqrt{\sum_i x_{ij}^2}$

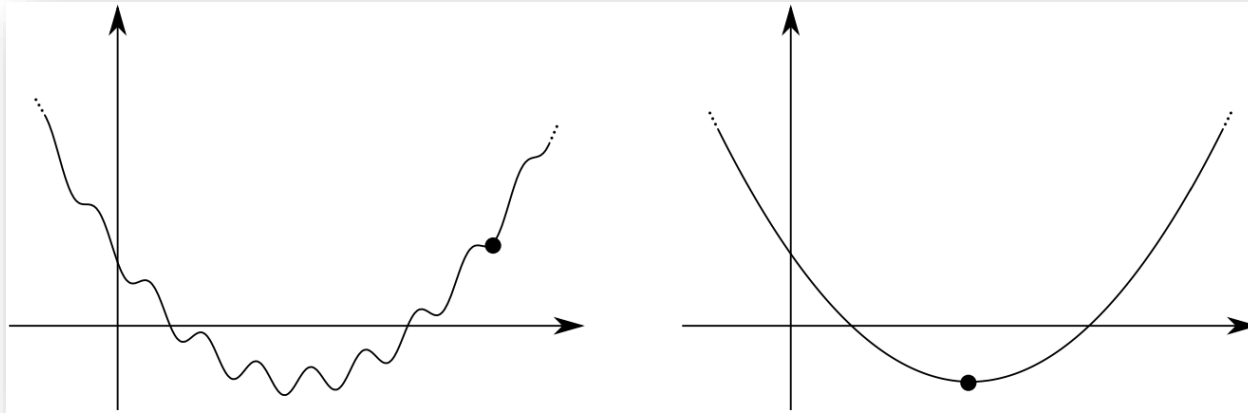
Smooth:  $\int \|\nabla f\|_2^2$

Piecewise constant:  $\int \|\nabla f\|_2$

???: Early stopping

## Regularization

# Some Tricks



Original



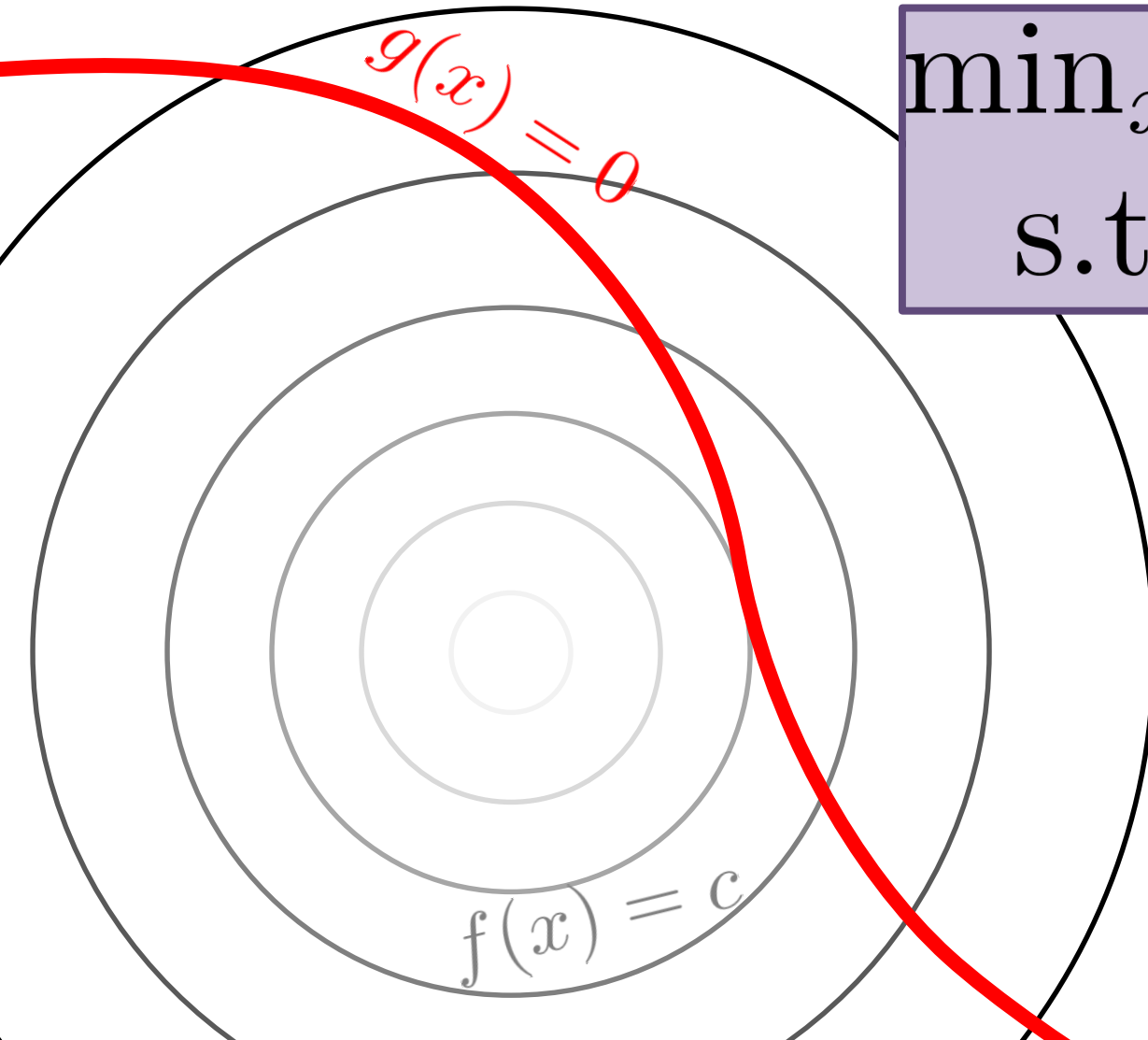
Blurred

**Multiscale/graduated optimization**

# Rough Plan

- Linear problems
- Unconstrained optimization
- **Equality-constrained optimization**
- Variational problems

# Lagrange Multipliers: Idea

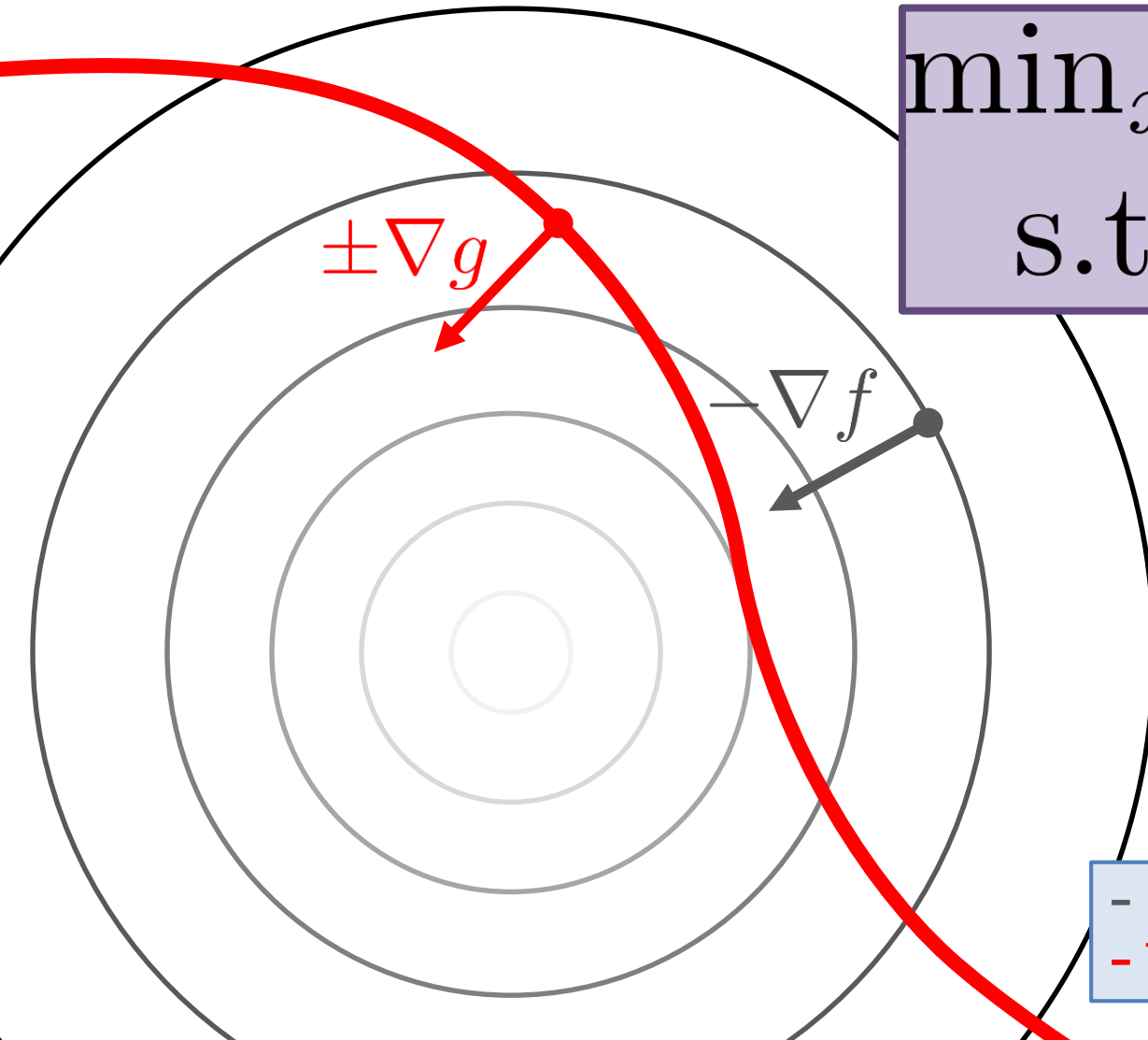


$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & g(x) = 0 \end{array}$$



# Lagrange Multipliers: Idea

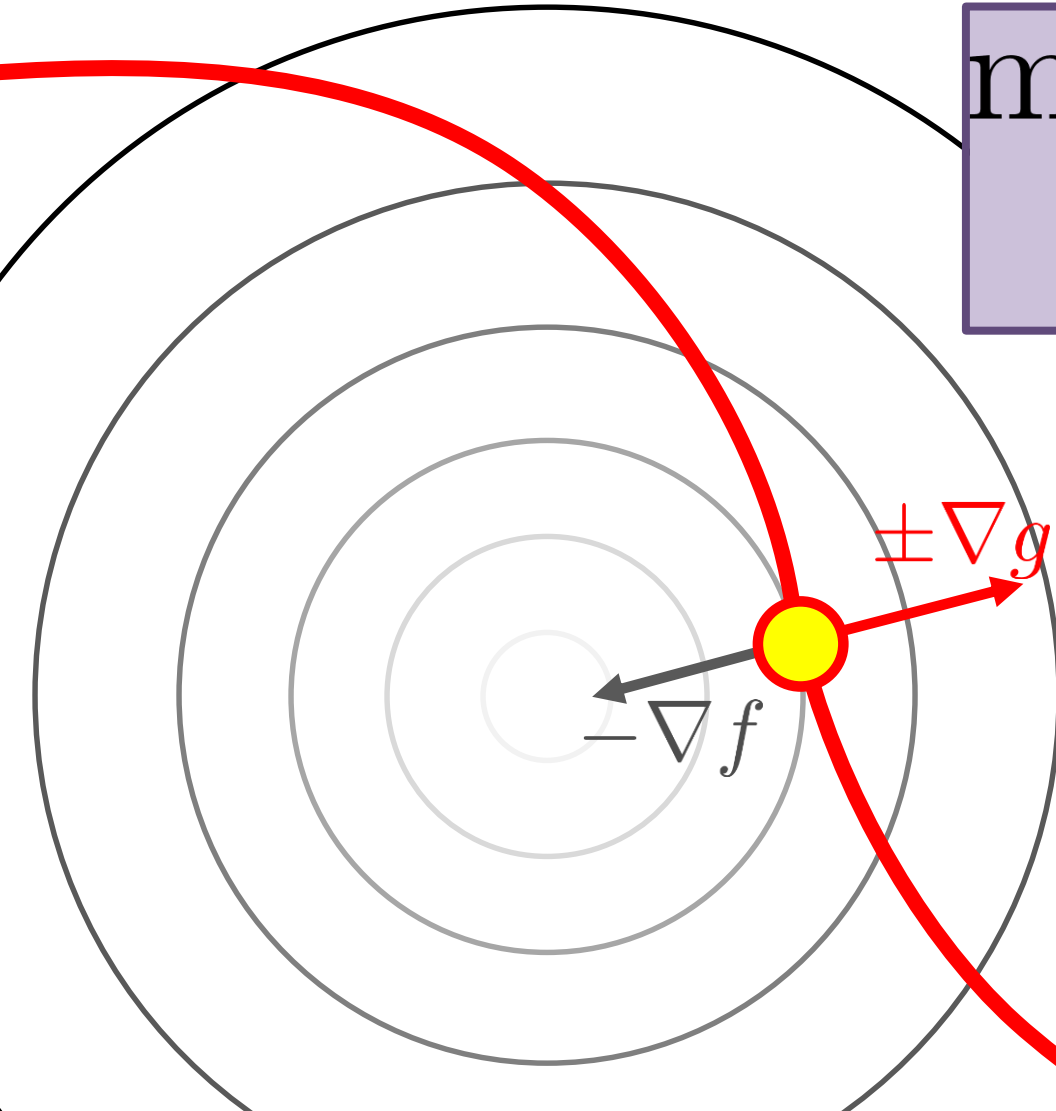
$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & g(x) = 0 \end{array}$$



- Decrease  $f$ :  $-\nabla f$
- Violate constraint:  $\pm \nabla g$

# Lagrange Multipliers: Idea

$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & g(x) = 0 \end{array}$$



**Want:**

$$\nabla f \parallel \nabla g$$
$$\implies \nabla f = \lambda \nabla g$$

# Use of Lagrange Multipliers

Turns constrained optimization into  
**unconstrained root-finding.**

$$\nabla f(x) = \lambda \nabla g(x)$$

$$g(x) = 0$$

# Quadratic with Linear Equality

$$\begin{aligned} \min_x \quad & \frac{1}{2} x^\top A x - b^\top x + c \\ \text{s.t.} \quad & M x = v \end{aligned}$$

(assume A is symmetric and positive definite)



$$\begin{pmatrix} A & M^\top \\ M & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} b \\ v \end{pmatrix}$$

# Many Options

- **Reparameterization**

Eliminate constraints to reduce to unconstrained case

- **Newton's method**

Approximation: quadratic function with linear constraint

- **Penalty method**

Augment objective with barrier term, e.g.  $f(x) + \rho|g(x)|$

# Example: Symmetric Eigenvectors

$$f(x) = x^\top Ax \implies \nabla f(x) = 2Ax$$

$$g(x) = \|x\|_2^2 \implies \nabla g(x) = 2x$$

$$\implies Ax = \lambda x$$

# Returning to Parameterization

$$\begin{aligned} \min_{x_1, \dots, x_{|V|}} & \sum_{(i,j) \in E} w_{ij} \|x_i - x_j\|_2^2 \\ \text{s.t.} & x_v \text{ fixed } \forall v \in V_0 \end{aligned}$$

**What if  
 $V_0 = \{\}$ ?**

# Nontriviality Constraint

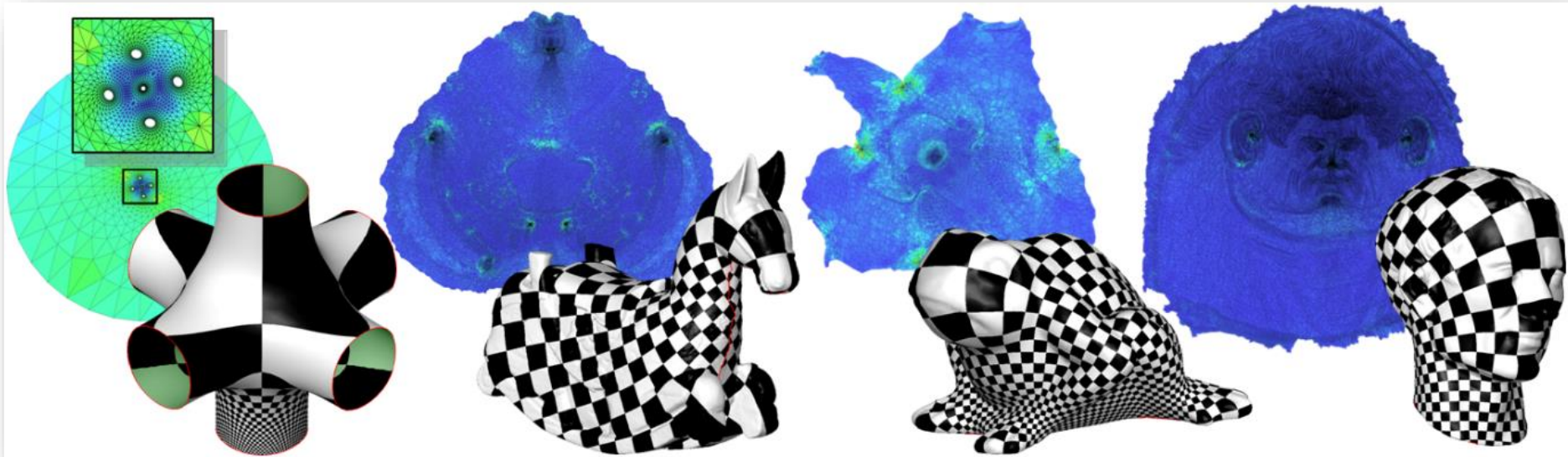
$$\left\{ \begin{array}{l} \min_x \quad \|Ax\|_2 \\ \text{s.t.} \quad \|x\|_2 = 1 \end{array} \right\} \mapsto A^\top Ax = \lambda x$$

**Prevents** trivial solution  $x \equiv 0$ .

Extract the **smallest eigenvalue**.



# Back to Parameterization



Mullen et al. "Spectral Conformal Parameterization." SGP 2008.

$$\min_u u^\top L_C u \iff L_C u = \lambda B u$$

$u^\top B e = 0$  ← Easy fix

$u^\top B u = 1$

# Basic Idea of Eigenalgorithms

$$A\vec{v} = c_1 A\vec{x}_1 + \cdots + c_n A\vec{x}_n$$

$$= c_1 \lambda_1 \vec{x}_1 + \cdots + c_n \lambda_n \vec{x}_n \text{ since } A\vec{x}_i = \lambda_i \vec{x}_i$$

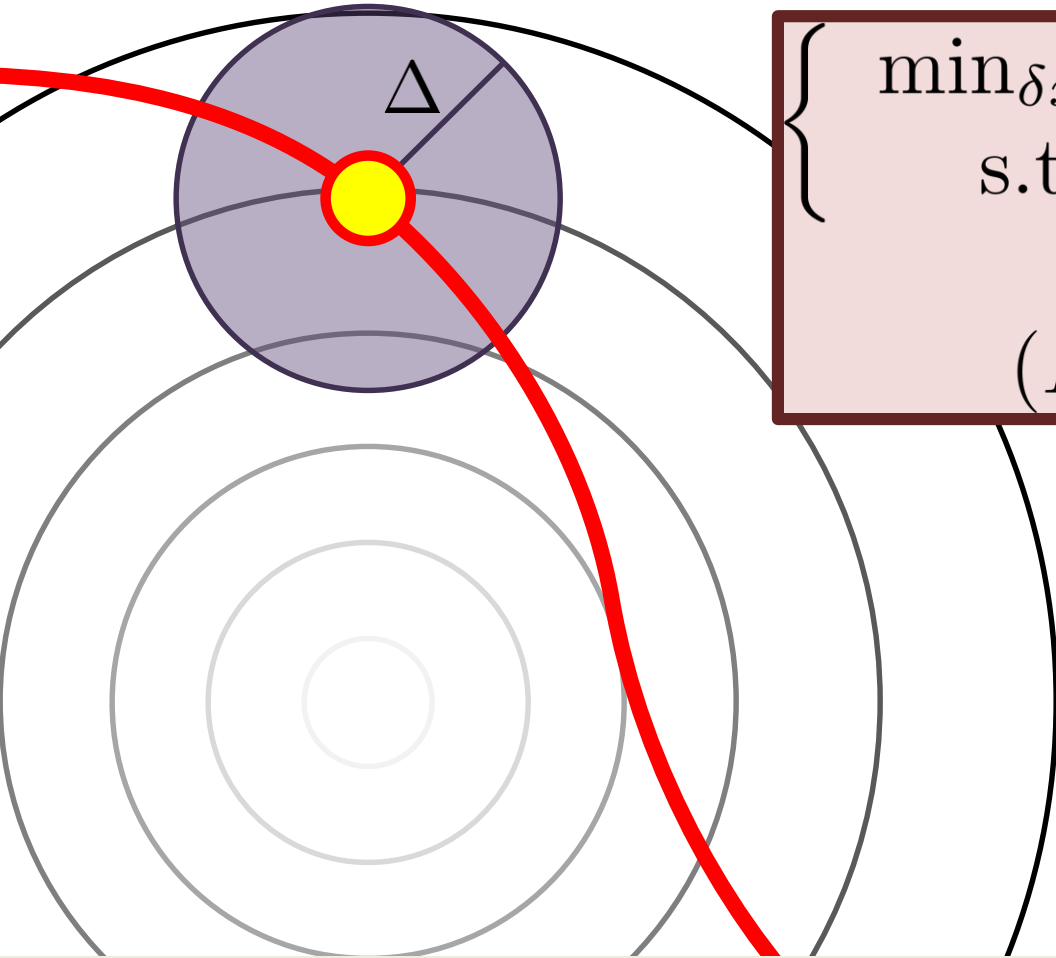
$$= \lambda_1 \left( c_1 \vec{x}_1 + \frac{\lambda_2}{\lambda_1} c_2 \vec{x}_2 + \cdots + \frac{\lambda_n}{\lambda_1} c_n \vec{x}_n \right)$$

$$A^2 \vec{v} = \lambda_1^2 \left( c_1 \vec{x}_1 + \left( \frac{\lambda_2}{\lambda_1} \right)^2 c_2 \vec{x}_2 + \cdots + \left( \frac{\lambda_n}{\lambda_1} \right)^2 c_n \vec{x}_n \right)$$

⋮

$$A^k \vec{v} = \lambda_1^k \left( c_1 \vec{x}_1 + \left( \frac{\lambda_2}{\lambda_1} \right)^k c_2 \vec{x}_2 + \cdots + \left( \frac{\lambda_n}{\lambda_1} \right)^k c_n \vec{x}_n \right).$$

# Trust Region Methods



$$\left\{ \begin{array}{l} \min_{\delta x} \quad \frac{1}{2} \delta x^\top H \delta x + w^\top x \\ \text{s.t.} \quad \|\delta x\|_2^2 \leq \Delta \end{array} \right\}$$

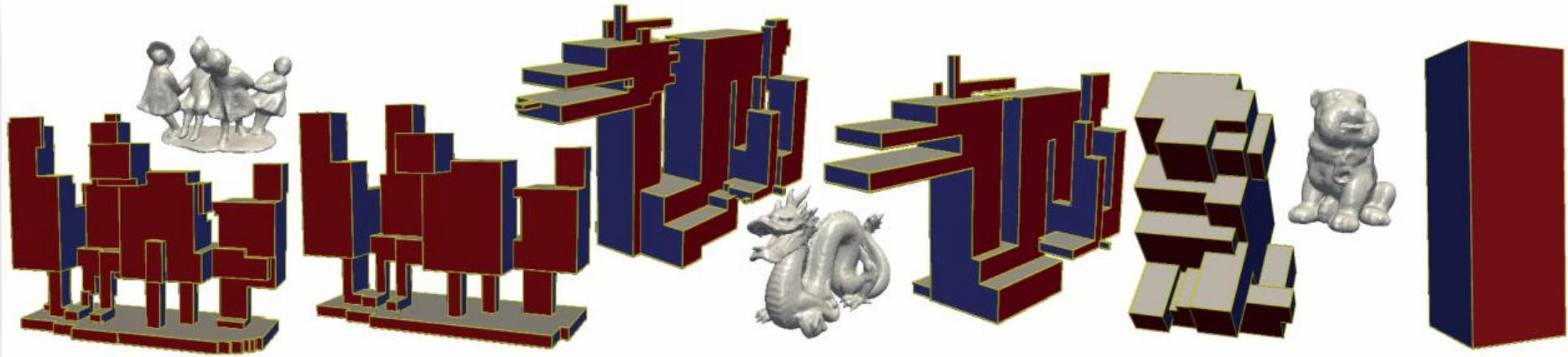
↓

$$(H + \lambda I) \delta x = -w$$

**Fix (or adjust)  
damping  
parameter  $\lambda > 0$ .**

**Example: Levenberg-Marquardt**

# Example: Polycube Maps



Huang et al. "L1-Based Construction of Polycube Maps from Complex Shapes." TOG 2014.

**Align with coordinate axes**

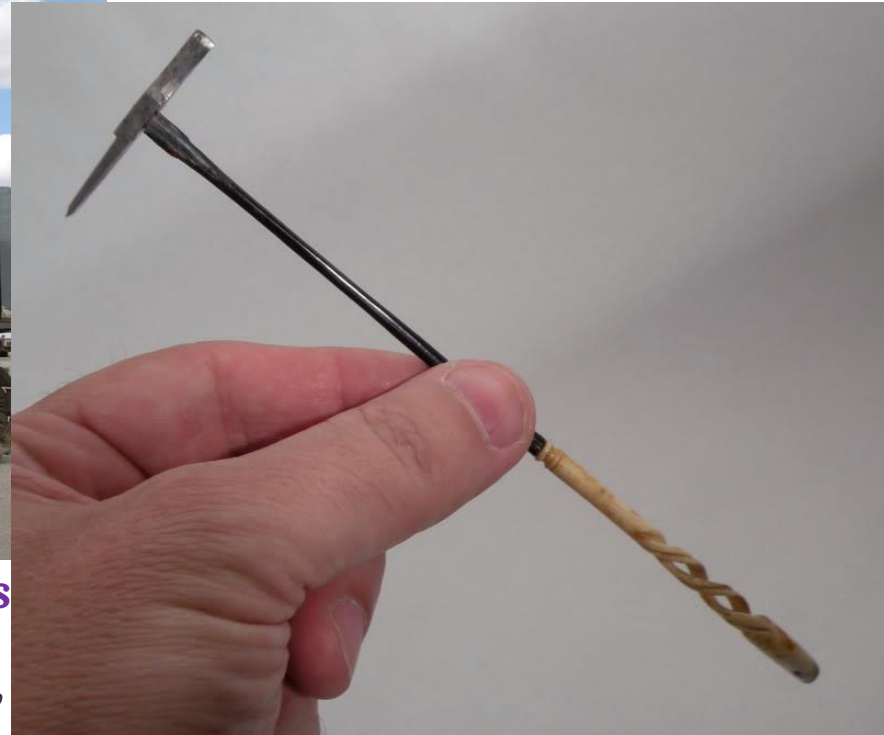
$$\begin{aligned} \min_X \quad & \sum_{b_i} \mathcal{A}(b_i; X) \|n(b_i; X)\|_1 \\ \text{s.t.} \quad & \sum_{b_i} \mathcal{A}(b_i; X) = \sum_{b_i} \mathcal{A}(b_i; X_0) \end{aligned}$$

**Preserve area**

*Note:* Final method includes more terms!

*Aside:*

# Convex Optimization Tools



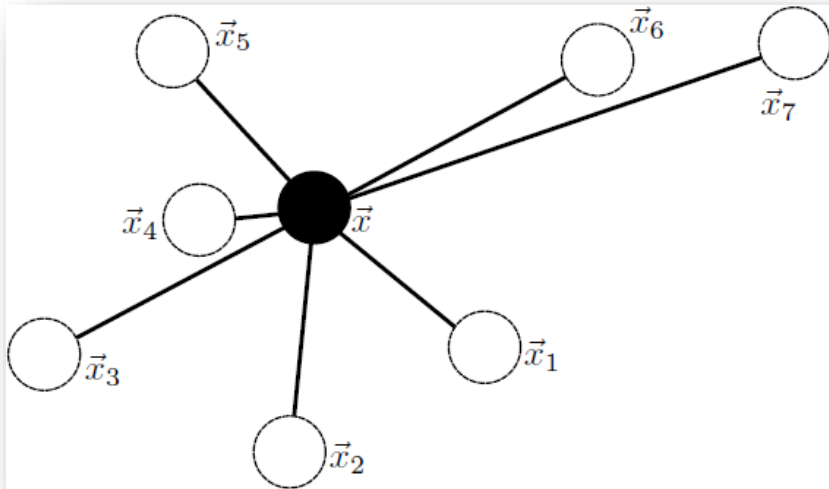
versus

*Sometimes work for non-convex problems...*

**Try lightweight options**

# Iteratively Reweighted Least Squares

$$\min_x \sum_i \phi(x^\top a_i + b_i) \leftrightarrow \left\{ \begin{array}{l} \min_{x, y_i} \sum_i y_i (x^\top a_i + b_i)^2 \\ \text{s.t. } y_i = \phi(x^\top a_i + b_i) (x^\top a_i + b_i)^{-2} \end{array} \right\}$$



**“Geometric median”**

$$\min_x \sum_i \|x - p_i\|_2 \implies \begin{cases} x \leftarrow \min_x \sum_i y_i \|x - p_i\|_2^2 \\ y_i \leftarrow \|x - p_i\|_2^{-1} \end{cases}$$

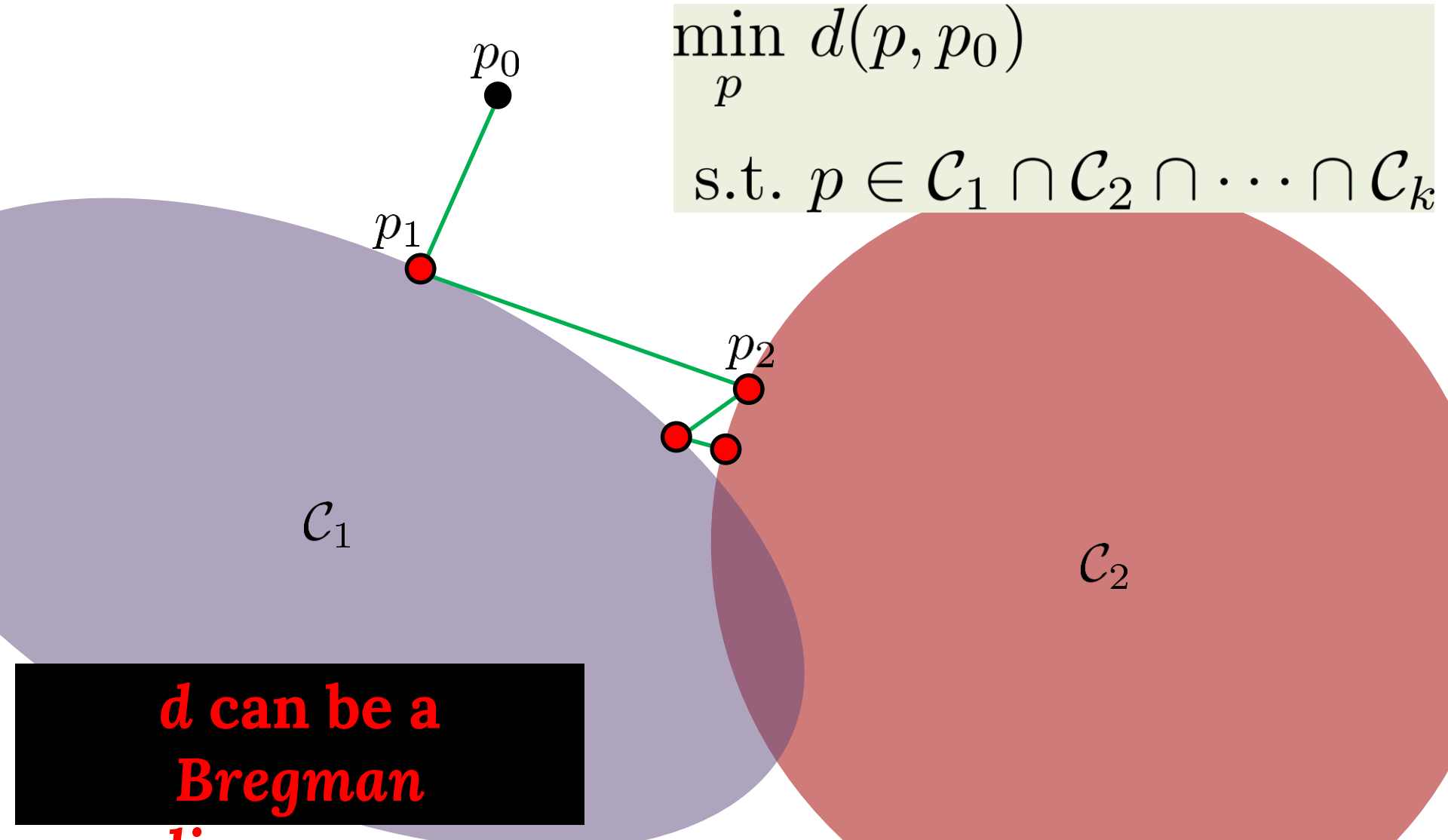
**Repeatedly solve linear systems**



# Alternating Projection

$$\min_p d(p, p_0)$$

$$\text{s.t. } p \in \mathcal{C}_1 \cap \mathcal{C}_2 \cap \cdots \cap \mathcal{C}_k$$



***d* can be a  
Bregman**

# Augmented Lagrangians

$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & g(x) = 0 \end{array}$$

↓

$$\begin{array}{ll} \min_x & f(x) + \frac{\rho}{2} \|g(x)\|_2^2 \\ \text{s.t.} & g(x) = 0 \end{array}$$

**Does nothing when  
constraint is satisfied**

**Add constraint to objective**



# Alternating Direction Method of Multipliers (ADMM)

$$\begin{aligned} \min_{x,z} \quad & f(x) + g(z) \\ \text{s.t.} \quad & Ax + Bz = c \end{aligned}$$

$$\Lambda_\rho(x, z; \lambda) = f(x) + g(z) + \lambda^\top (Ax + Bz - c) + \frac{\rho}{2} \|Ax + Bz - c\|_2^2$$

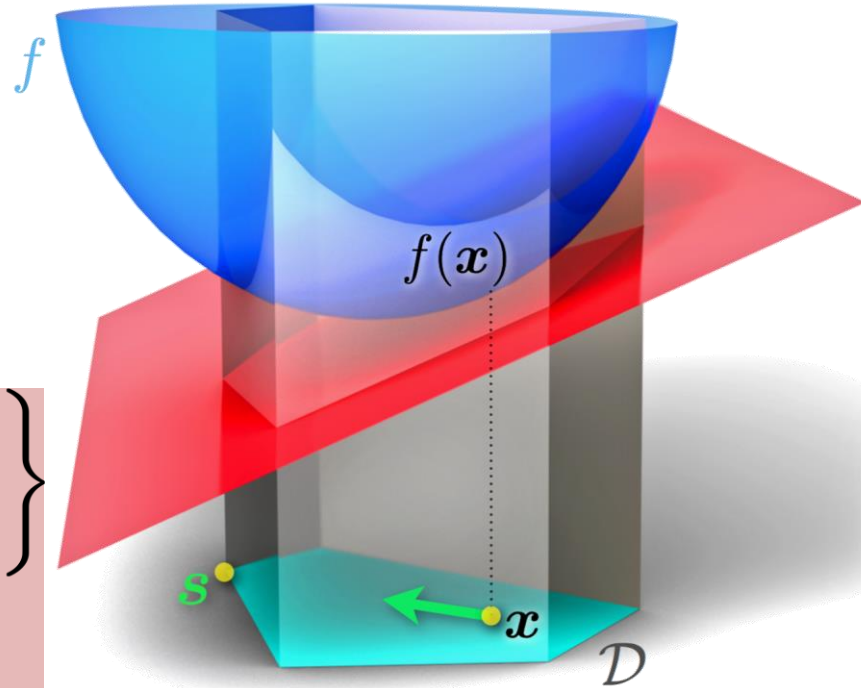
$$x \leftarrow \arg \min_x \Lambda_\rho(x, z, \lambda)$$

$$z \leftarrow \arg \min_z \Lambda_\rho(x, z, \lambda)$$

$$\lambda \leftarrow \lambda + \rho(Ax + Bz - c)$$

# Frank-Wolfe

*</aside>*



To minimize  $f(x)$  s.t.  $x \in \mathcal{D}$ :

$$s_k \leftarrow \left\{ \begin{array}{l} \arg \min_s s^\top \nabla f(x_k) \\ \text{s.t. } s \in \mathcal{D} \end{array} \right\}$$

$$\gamma \leftarrow \frac{2}{k+2}$$

$$x_{k+1} \leftarrow x_k + \gamma(s_k - x_k)$$

[https://en.wikipedia.org/wiki/Frank%E2%80%93Wolfe\\_algorithm](https://en.wikipedia.org/wiki/Frank%E2%80%93Wolfe_algorithm)

**Linearize objective, preserve constraints**

# Rough Plan

- Linear problems
- Unconstrained optimization
- Equality-constrained optimization
- **Variational problems**

# Variational Calculus: Big Idea

**Sometimes your unknowns  
are not numbers!**

Can we use calculus to optimize anyway?

# On the Board

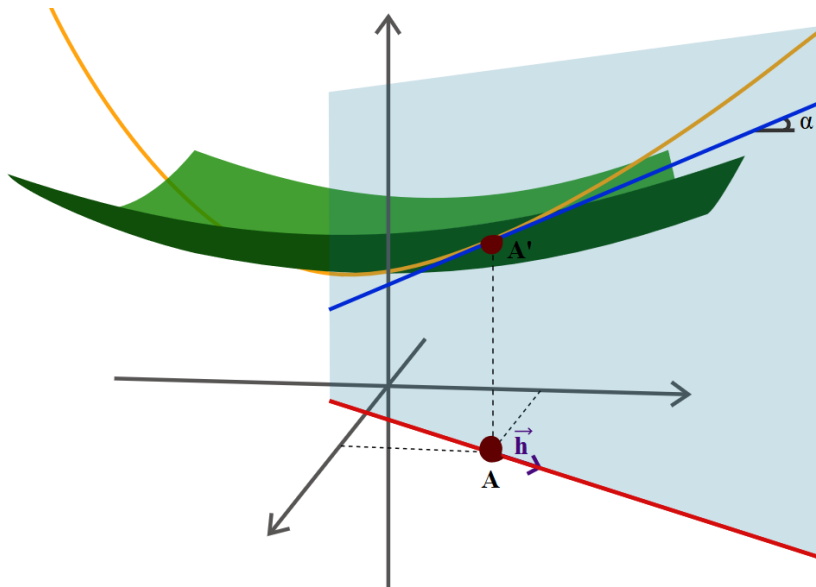
$$\min_f \int_{\Omega} \|\vec{v}(x) - \nabla f(x)\|_2^2 d\vec{x}$$

$$\min_{\int_{\Omega} f(x)^2 d\vec{x} = 1} \int_{\Omega} \|\nabla f(x)\|_2^2 d\vec{x}$$

# Gâteaux Derivative

$$d\mathcal{F}[u; \psi] := \frac{d}{dh} \mathcal{F}[u + h\psi] \Big|_{h=0}$$

Vanishes for all  $\psi$  at a critical point!



**Analog of derivative at  $u$  in  $\psi$  direction**