



# Optimization for Deep Learning

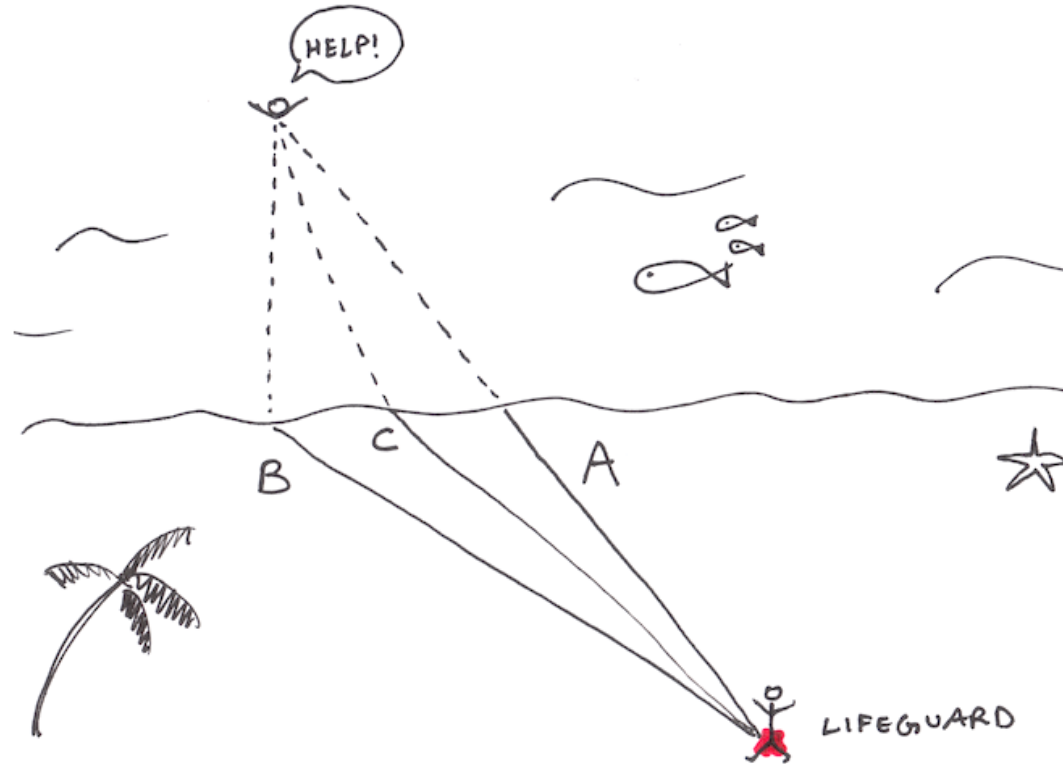
**Industrial AI Lab.**  
**Prof. Seungchul Lee**

# Optimization

- An important tool in
  - 1) Engineering problem solving and
  - 2) Decision science
- People optimize
- Nature optimizes

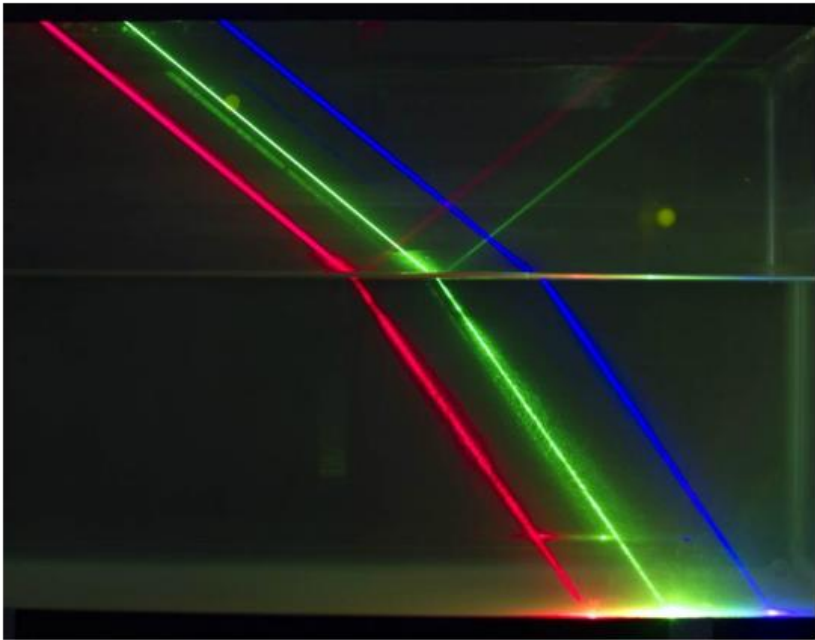
# Optimization

- People optimize



# Optimization

- Nature optimizes



# Optimization

- 3 key components
  - 1) Objective function
  - 2) Decision variable or unknown
  - 3) Constraints
- Procedures
  - 1) The process of identifying objective, variables, and constraints for a given problem (known as "modeling")
  - 2) Once the model has been formulated, optimization algorithm can be used to find its solutions

# Optimization: Mathematical Model

- In mathematical expression

$$\begin{array}{ll} \min_x & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \dots, m \end{array}$$

–  $x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$  is the decision variable

–  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is objective function

– Feasible region:  $C = \{x: g_i(x) \leq 0, \quad i = 1, \dots, m\}$

–  $x^* \in \mathbb{R}^n$  is an optimal solution if  $x^* \in C$  and  $f(x^*) \leq f(x), \forall x \in C$

# Optimization: Mathematical Model

- In mathematical expression

$$\begin{array}{ll} \min_x & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \dots, m \end{array}$$

- Remarks: equivalent

$$\begin{array}{lll} \min_x f(x) & \leftrightarrow & \max_x -f(x) \\ g_i(x) \leq 0 & \leftrightarrow & -g_i(x) \geq 0 \\ h(x) = 0 & \leftrightarrow & \begin{cases} h(x) \leq 0 \\ h(x) \geq 0 \end{cases} \text{ and} \end{array}$$

# Unconstrained vs. Constrained

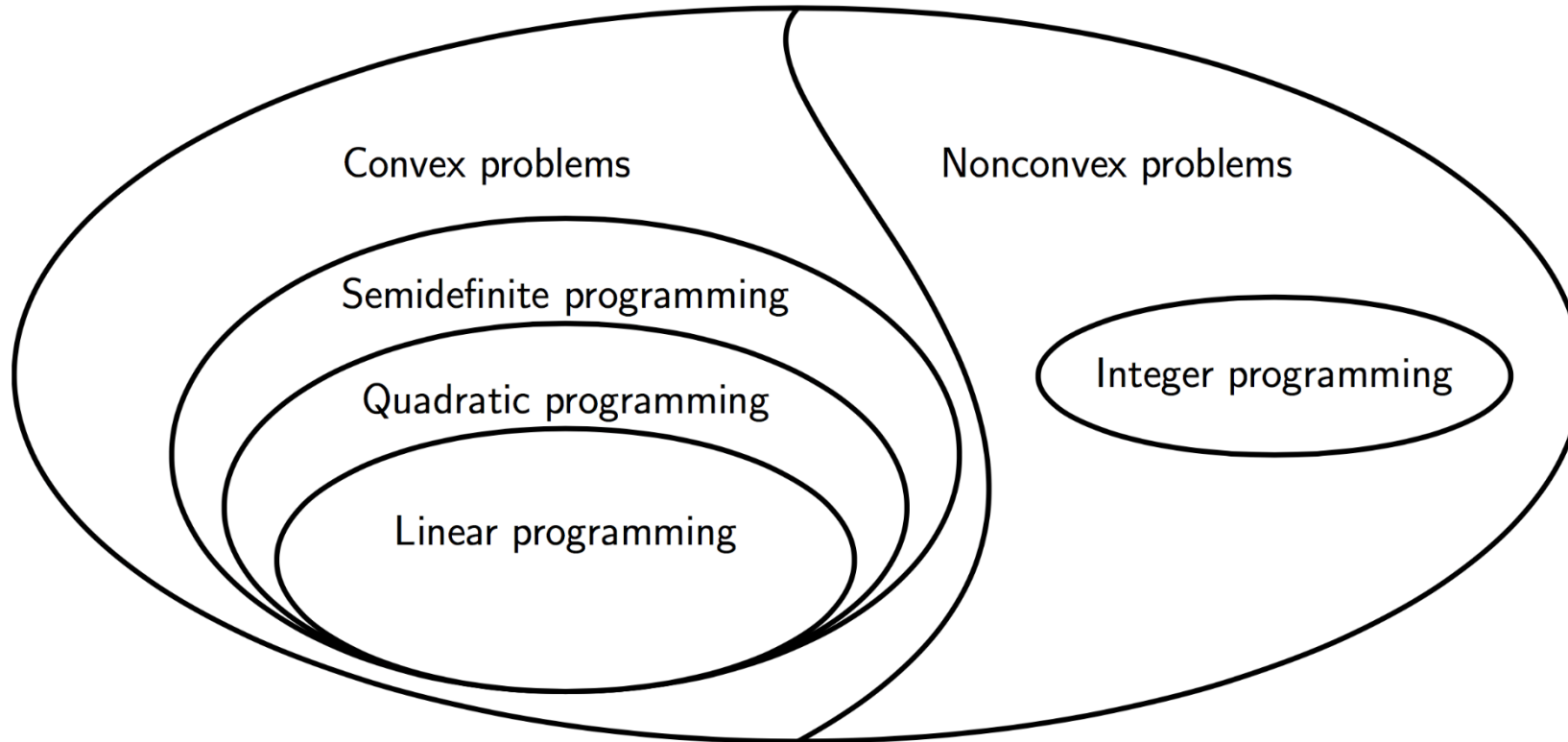
$$\underset{x}{\text{minimize}} \quad f(x)$$

vs.

$$\begin{aligned} &\underset{x}{\text{minimize}} \quad f(x) \\ &\text{subject to} \quad g_i(x) \leq 0, \quad i = 1, \dots, m \\ &\quad \quad \quad h_i(x) = 0, \quad i = 1, \dots, p \end{aligned}$$



# Convex vs. Nonconvex



# Convex Optimization

# Convex Optimization

- An extremely powerful subset of all optimization problems

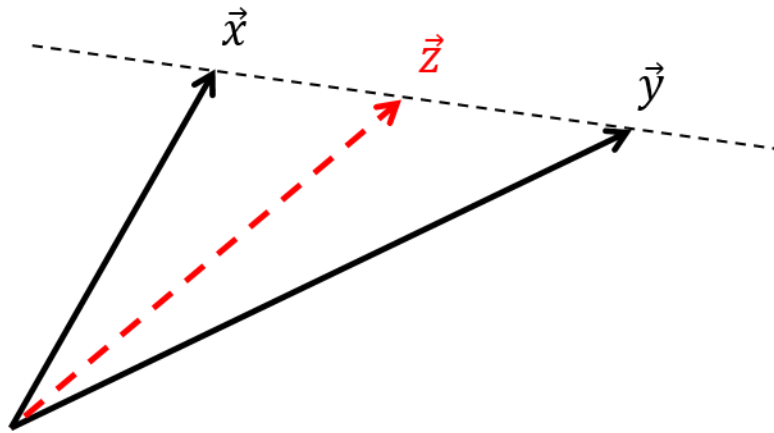
$$\begin{array}{ll}\underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & x \in \mathcal{C}\end{array}$$

- $f: \mathbb{R}^n \rightarrow \mathbb{R}$  is a convex function and
- Feasible region  $\mathcal{C}$  is a convex set

- Key property of convex optimization:
  - all local solutions are global solutions
- We will use CVX (or CVXPY) as a convex optimization solver
  - <https://www.cvxpy.org/>
  - Many examples later

# Linear Interpolation between Two Points

- $\vec{z} = \theta \vec{x} + (1 - \theta) \vec{y}$  and  $\theta \in [0, 1]$

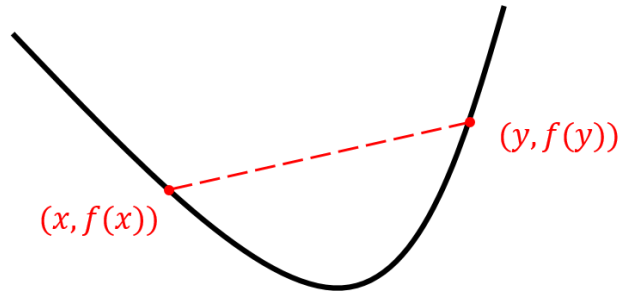


$$\begin{aligned}\vec{z} &= \vec{y} + \theta (\vec{x} - \vec{y}), & 0 \leq \theta \leq 1 \\ &= \theta \vec{x} + (1 - \theta) \vec{y}\end{aligned}$$

$$\vec{z} = \alpha \vec{x} + \beta \vec{y}, \quad \alpha + \beta = 1 \quad \text{and} \quad 0 \leq \alpha, \beta$$

# Convex Function and Convex Set

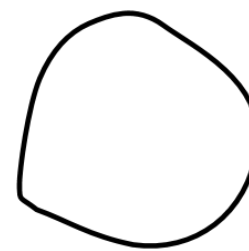
convex function



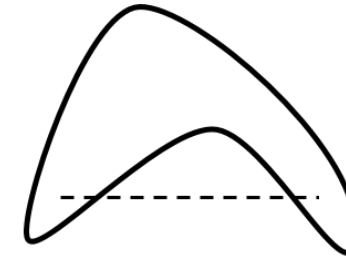
for any  $x, y \in \mathbb{R}^n$  and  $\theta \in [0, 1]$

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

convex set



Convex set



Non-convex set

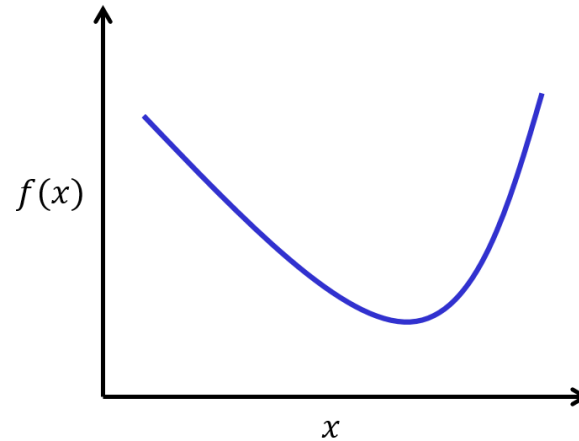
for a  $x, y \in \mathcal{C}$  and  $\theta \in [0, 1]$ ,

$$\theta x + (1 - \theta)y \in \mathcal{C}$$

# Solving Optimization Problems

# Solving Optimization Problems

- Starting with the unconstrained, one dimensional case



- To find minimum point  $x^*$ , we can look at the derivative of the function  $f'(x)$
  - Any location where  $f'(x) = 0$  will be a “flat” point in the function
- For convex problems, this is guaranteed to be a minimum

# Solving Optimization Problems

- Generalization for multivariate function  $f: \mathbb{R}^n \rightarrow \mathbb{R}$ 
  - the gradient of  $f$  must be zero

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

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- For defined as above, *gradient* is a  $n$ -dimensional vector containing partial derivatives with respect to each dimension

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

- For continuously differentiable  $f$  and unconstrained optimization, optimal point must have

$$\nabla_x f(x^*) = 0$$



# How do we Find $\nabla_x f(x) = 0$

- Direct solution
  - In some cases, it is possible to analytically compute  $x^*$  such that  $\nabla_x f(x^*) = 0$

$$f(x) = 2x_1^2 + x_2^2 + x_1 x_2 - 6x_1 - 5x_2$$

$$\Rightarrow \nabla_x f(x) = \begin{bmatrix} 4x_1 + x_2 - 6 \\ 2x_2 + x_1 - 5 \end{bmatrix}$$

$$\Rightarrow x^* = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

# Gradients

- Matrix derivatives

$y$	$\frac{\partial y}{\partial x}$
$Ax$	$A^T$
$x^T A$	$A$
$x^T x$	$2x$
$x^T Ax$	$Ax + A^T x$

# How to Find $\nabla_x f(x) = 0$

- Direct solution
  - In some cases, it is possible to analytically compute  $x^*$  such that  $\nabla_x f(x^*) = 0$

$y$	$\frac{\partial y}{\partial x}$
$Ax$	$A^T$
$x^T A$	$A$
$x^T x$	$2x$
$x^T Ax$	$Ax + A^T x$

$$f(x) = 2x_1^2 + x_2^2 + x_1 x_2 - 6x_1 - 5x_2$$

$$\Rightarrow \nabla_x f(x) = \begin{bmatrix} 4x_1 + x_2 - 6 \\ 2x_2 + x_1 - 5 \end{bmatrix}$$

$$\Rightarrow x^* = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

# Examples

- affine function  $g(x) = a^T x + b$

$$\nabla g(x) = a, \quad \nabla^2 g(x) = 0$$

$y$	$\frac{\partial y}{\partial x}$
$Ax$	$A^T$
$x^T A$	$A$
$x^T x$	$2x$
$x^T Ax$	$Ax + A^T x$

- quadratic function  $g(x) = x^T P x + q^T x + r$  ,  $P = P^T$

$$\nabla g(x) = 2Px + q, \quad \nabla^2 g(x) = 2P$$

- $g(x) = \|Ax - b\|^2 = x^T A^T A x - 2b^T A x + b^T b$

$$\nabla g(x) = 2A^T A x - 2A^T b, \quad \nabla^2 g(x) = 2A^T A$$

# Revisit: Least-Square Solution

- Scalar Objective:  $J = \|Ax - y\|^2$

$$\begin{aligned} J(x) &= (Ax - y)^T (Ax - y) \\ &= (x^T A^T - y^T) (Ax - y) \\ &= x^T A^T Ax - x^T A^T y - y^T Ax + y^T y \end{aligned}$$

$$\begin{aligned} \frac{\partial J}{\partial x} &= A^T Ax + (A^T A)^T x - A^T y - (y^T A)^T \\ &= 2A^T Ax - 2A^T y = 0 \end{aligned}$$

$$\implies (A^T A) x = A^T y$$

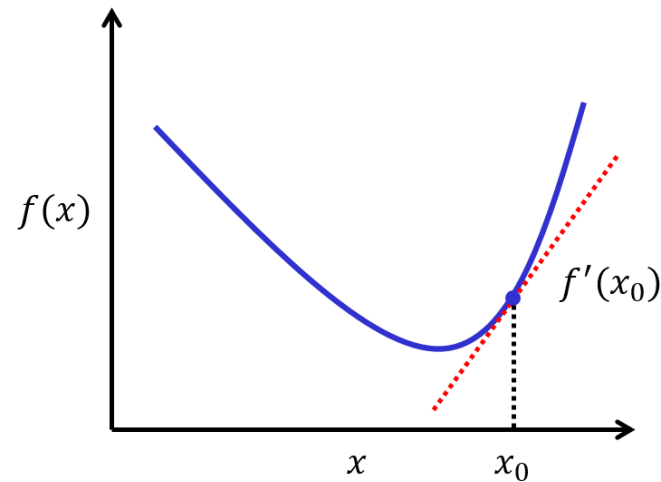
$$\therefore x^* = (A^T A)^{-1} A^T y$$

$y$	$\frac{\partial y}{\partial x}$
$Ax$	$A^T$
$x^T A$	$A$
$x^T x$	$2x$
$x^T Ax$	$Ax + A^T x$

# How do we Find $\nabla_x f(x) = 0$

- Iterative methods

- More commonly the condition that the gradient equal zero will not have an analytical solution, require iterative methods

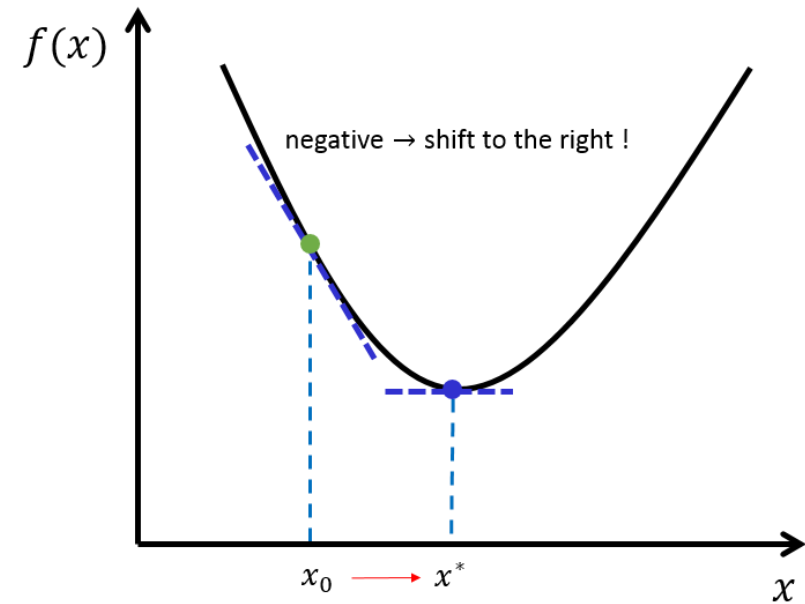
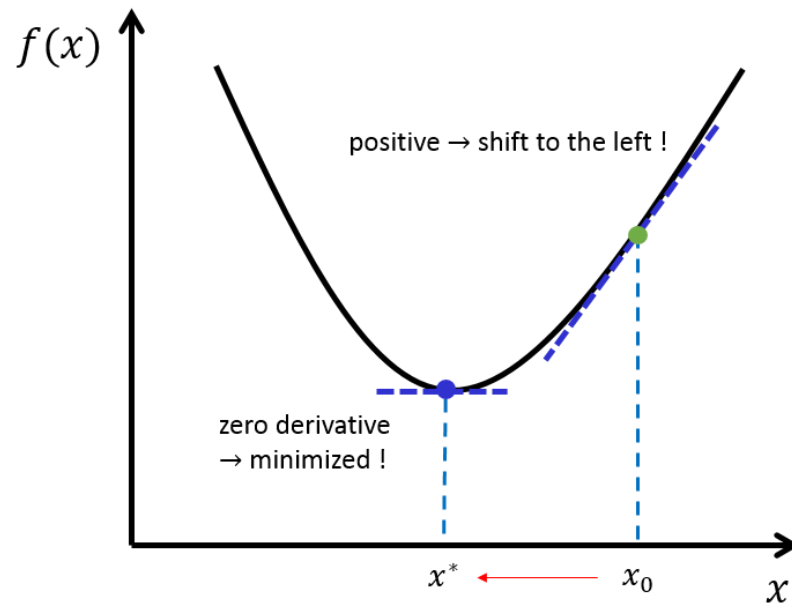


- The gradient points in the direction of “steepest ascent” for function  $f$

# Descent Direction (1D)

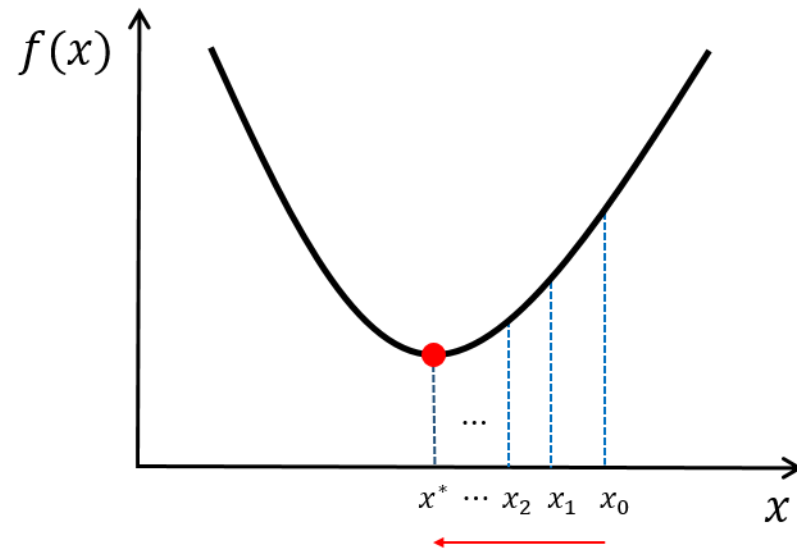
- It motivates the *gradient descent* algorithm, which repeatedly takes steps in the direction of the negative gradient

$$x \leftarrow x - \alpha \nabla_x f(x) \quad \text{for some step size } \alpha > 0$$



# Gradient Descent

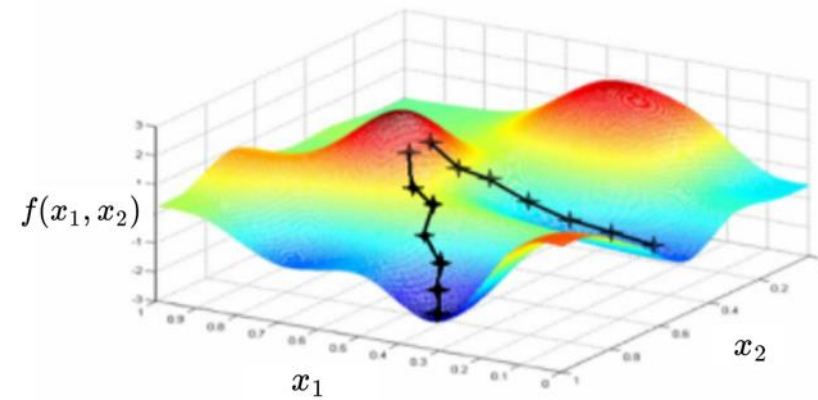
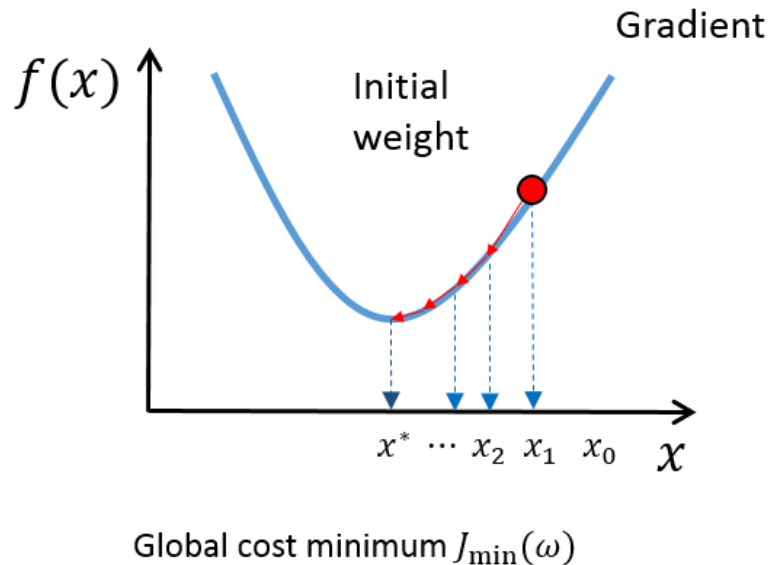
Repeat:  $x \leftarrow x - \alpha \nabla_x f(x)$  for some *step size*  $\alpha > 0$





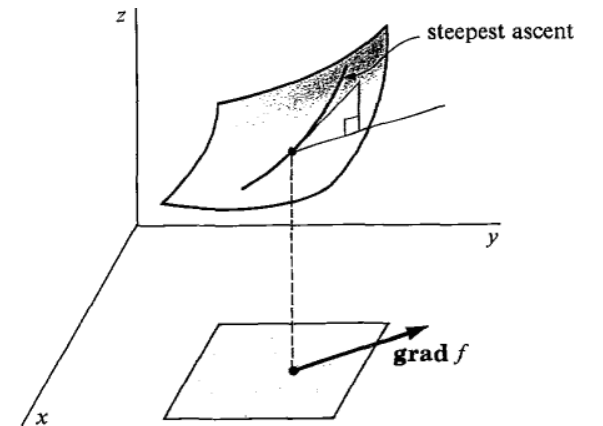
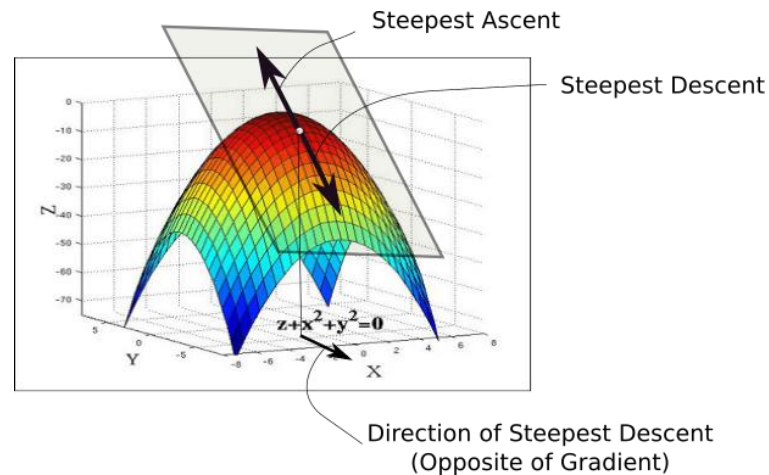
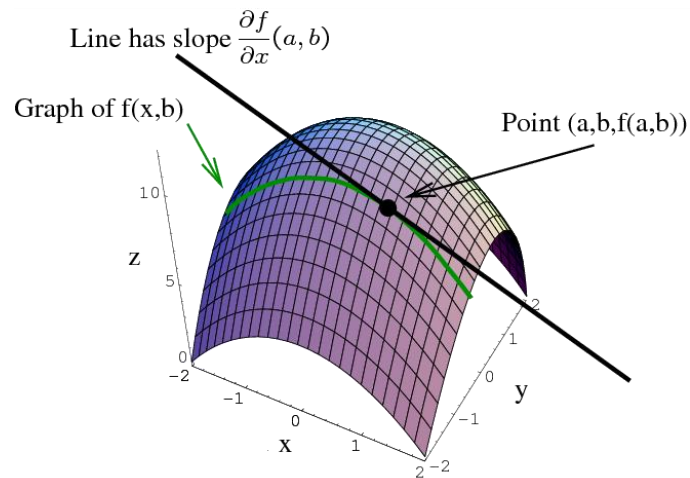
# Gradient Descent in High Dimension

Repeat:  $x \leftarrow x - \alpha \nabla_x f(x)$  for some *step size*  $\alpha > 0$



# Gradient Descent in High Dimension

Repeat:  $x \leftarrow x - \alpha \nabla_x f(x)$  for some *step size*  $\alpha > 0$



# Gradient Descent

$$\begin{aligned} \min \quad & (x_1 - 3)^2 + (x_2 - 3)^2 \\ = \min \quad & \frac{1}{2} \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 6 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 18 \end{aligned}$$

$$\begin{aligned} f &= \frac{1}{2} X^T H X + g^T X \\ \nabla f &= H X + g \end{aligned}$$

- Update rule:  $X_{i+1} = X_i - \alpha_i \nabla f(X_i)$

```
H = np.matrix([[2, 0],[0, 2]])
g = -np.matrix([[6],[6]])

x = np.zeros((2,1))
alpha = 0.2

for i in range(25):
    df = H*x + g
    x = x - alpha*df

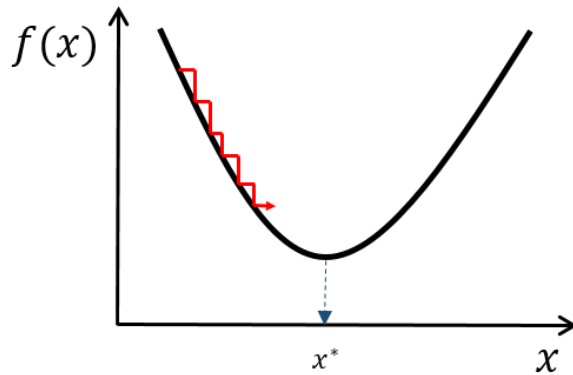
print(x)
```

```
[[ 2.99999147]
 [ 2.99999147]]
```

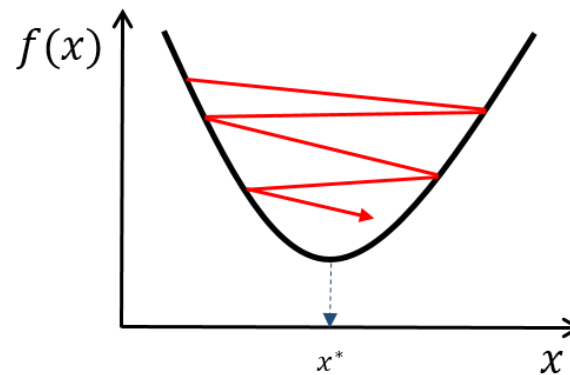
$y$	$\frac{\partial y}{\partial x}$
$Ax$	$A^T$
$x^T A$	$A$
$x^T x$	$2x$
$x^T A x$	$Ax + A^T x$

# Choosing Step Size $\alpha$

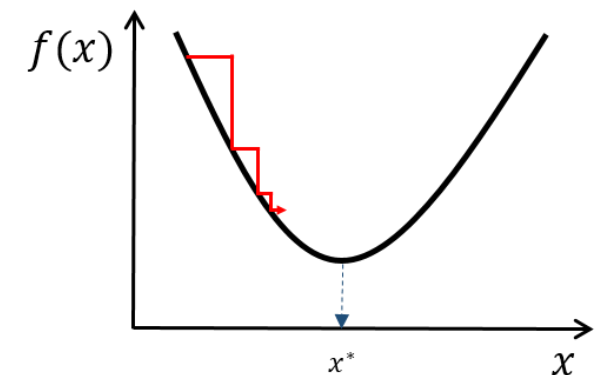
- Learning rate



Too small: converge  
very slowly

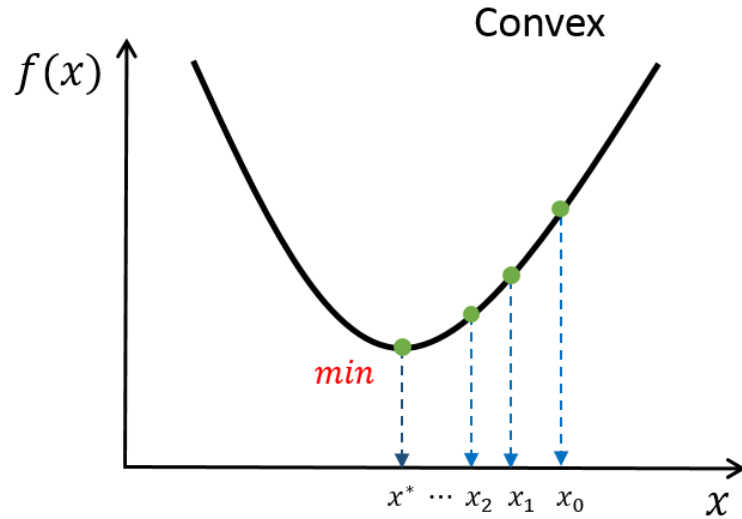


Too big: overshoot and  
even diverge

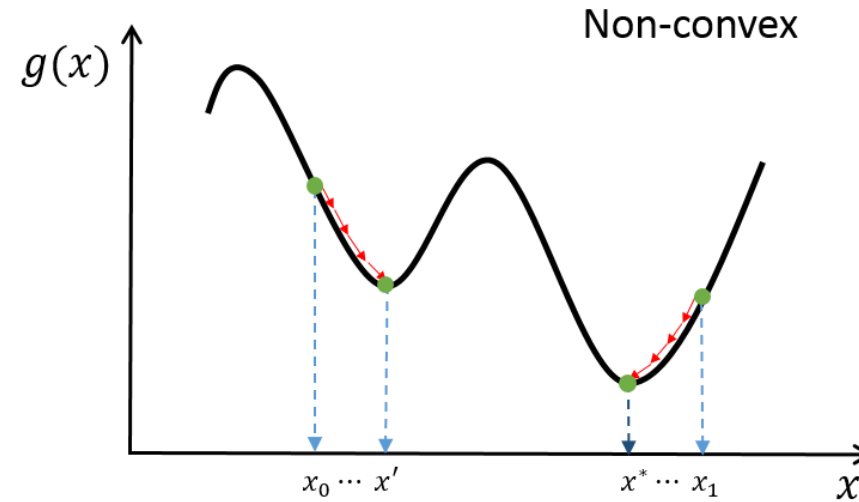


Reduce size over time

# Where will We Converge?



Any local minimum is a global minimum



Multiple local minima may exist

- Random initialization
- Multiple trials

# Gradient Descent vs. Analytical Solution

- Analytical solution for MSE
- Gradient descent
  - Easy to implement
  - Very general, can be applied to any differentiable loss functions
  - Requires less memory and computations (for stochastic methods)
- Gradient descent provides a general learning framework
- Can be used both for classification and regression
- Training Neural Networks: Gradient Descent

# Practically Solving Optimization Problems

- The good news: for many classes of optimization problems, people have already done all the “hard work” of developing numerical algorithms
  - A wide range of tools that can take optimization problems in “natural” forms and compute a solution
- We will use CVX (or CVXPY) as an optimization solver
  - Only for convex problems
  - Download: <https://www.cvxpy.org/>
- Gradient descent
  - Neural networks/deep learning
  - TensorFlow

# Summary: Training Neural Networks

- Optimization procedure

Start at a random point

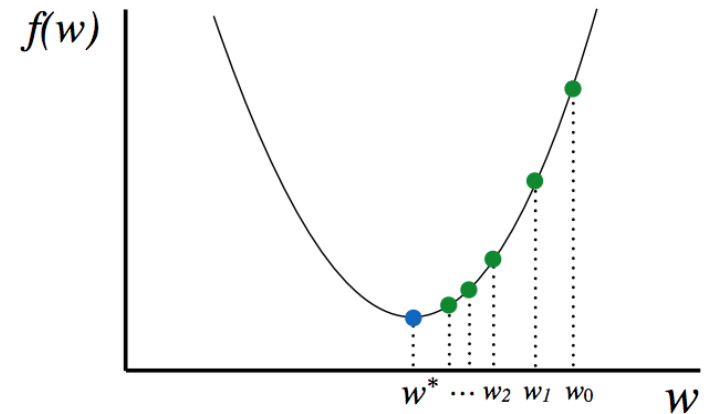
**Repeat**

Determine a descent direction

Choose a step size

Update

**Until** stopping criterion is satisfied



- It is not easy to numerically compute gradients in network in general.
  - The good news: people have already done all the "hard work" of developing numerical solvers (or libraries)
  - There are a wide range of tools
    - We will use **TensorFlow**



# Examples

# Linear Programming

- Objective function and constraints are both linear
- Convex

$$\max \quad 3x_1 + \frac{3}{2}x_2 \quad \leftarrow \text{objective function}$$

$$\begin{aligned} \text{subject to} \quad & -1 \leq x_1 \leq 2 \\ & 0 \leq x_2 \leq 3 \end{aligned} \quad \leftarrow \text{constraints}$$

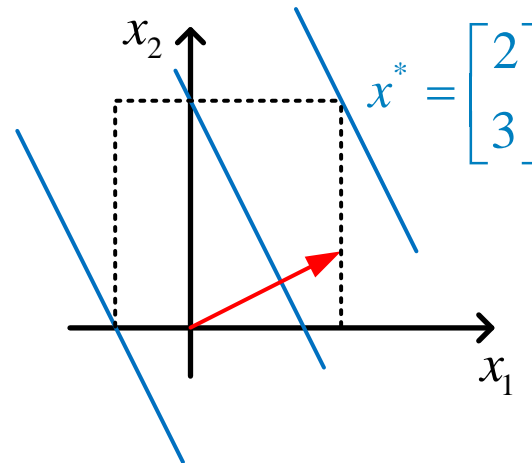
# Method 1: Geometric Approach

$$\max \quad 3x_1 + \frac{3}{2}x_2 \quad \leftarrow \text{objective function}$$

$$3x_1 + 1.5x_2 = C \quad \Rightarrow$$

$$\begin{aligned} \text{subject to} \quad & -1 \leq x_1 \leq 2 \quad \leftarrow \text{constraints} \\ & 0 \leq x_2 \leq 3 \end{aligned}$$

$$x_2 = -2x_1 + \frac{2}{3}C$$



## Method 2: CVXPY

- Many examples will be provided throughout the lecture

$$\begin{array}{ll} \max_x & 3x_1 + \frac{3}{2}x_2 \\ \text{subject to} & -1 \leq x_1 \leq 2 \\ & 0 \leq x_2 \leq 3 \end{array} \quad \Longrightarrow \quad \begin{array}{ll} \min_x & -\begin{bmatrix} 3 \\ 3/2 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ \text{subject to} & \begin{bmatrix} -1 \\ 0 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 2 \\ 3 \end{bmatrix} \end{array}$$

```
import numpy as np
import cvxpy as cvx

f = np.array([[3], [3/2]])
lb = np.array([[ -1], [0]])
ub = np.array([[2], [3]])

x = cvx.Variable(2,1)

obj = cvx.Minimize(-f.T*x)
constraints = [lb <= x, x <= ub]

prob = cvx.Problem(obj, constraints)
result = prob.solve()

print(x.value)
print(result)
```

```
[[ 1.99999999]
 [ 2.99999999]]
-10.499999966365493
```

$$\begin{array}{ll} \min_x & - \begin{bmatrix} 3 \\ 3/2 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\ \text{subject to} & \begin{bmatrix} -1 \\ 0 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} 2 \\ 3 \end{bmatrix} \end{array}$$

# Quadratic Form

$$q(x_1, \dots, x_n) = \sum_{i=1}^n \sum_{j=1}^n h_{ij} x_i x_j = x^T H x$$

# Quadratic Programming

$$\begin{array}{ll} \min & \frac{1}{2}x^2 + 3x + 4y \\ \text{subject to} & x + 3y \geq 15 \\ & 2x + 5y \leq 100 \\ & 3x + 4y \leq 80 \\ & x, y \geq 0 \end{array} \quad \Rightarrow \quad \begin{array}{ll} \min_X & X^T H X + f^T X \\ \text{subject to} & A X \leq b \\ & A_{eq} X = b_{eq} \\ & lb \leq X \leq ub \end{array}$$

# Quadratic Programming

```
f = np.array([[3], [4]])
H = np.array([[1/2, 0], [0, 0]])

A = np.array([[-1, -3], [2, 5], [3, 4]])
b = np.array([[-15], [100], [80]])
lb = np.array([[0], [0]])

x = cvx.Variable(2,1)

obj = cvx.Minimize(cvx.quad_form(x, H) + f.T*x)
constraints = [A*x <= b, lb <= x]

prob = cvx.Problem(obj, constraints)
result = prob.solve()

print(x.value)
print(result)
```

$\Rightarrow$

$$\begin{array}{ll} \min_X & X^T H X + f^T X \\ \text{subject to} & A X \leq b \\ & A_{eq} X = b_{eq} \\ & lb \leq X \leq ub \end{array}$$

```
[[ 6.90879937e-10]
 [ 5.00000000e+00]]
20.000000000914817
```



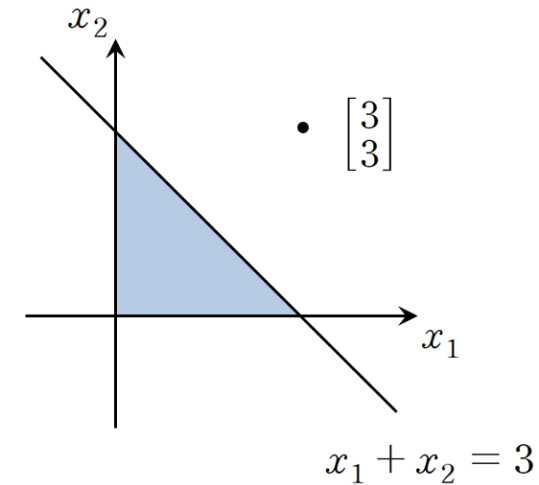
# Example: Shortest Distance

- Find the best location to listen to singer's voice

$$\begin{aligned} \min \quad & \sqrt{(x_1 - 3)^2 + (x_2 - 3)^2} \quad \Rightarrow \quad \min \quad (x_1 - 3)^2 + (x_2 - 3)^2 \\ \text{subject to} \quad & x_1 + x_2 \leq 3 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad & x_1^2 - 6x_1 + 9 + x_2^2 - 6x_2 + 9 \\ & = x_1^2 + x_2^2 - 6x_1 - 6x_2 + 18 \\ & = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 6 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 18 \\ & \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq 3 \\ & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \leq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} \phantom{0} \\ \phantom{0} \end{bmatrix} \end{aligned}$$

$$\begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix}$$



```
f = np.array([[ -6], [ -6]])
H = np.array([[ 1, 0], [ 0, 1]])

A = np.array([ 1, 1])
b = 3
lb = np.array([[ 0], [ 0]])

x = cvx.Variable(2, 1)

obj = cvx.Minimize(cvx.quad_form(x, H) + f.T*x)
constraints = [A*x <= b, lb <= x]

prob = cvx.Problem(obj, constraints)
result = prob.solve()

print(x.value)
```

# Example: Shortest Distance

$$\min d_1 + d_2 = \min \left\| \vec{a} - \begin{bmatrix} x \\ 0 \end{bmatrix} \right\|_2 + \left\| \vec{b} - \begin{bmatrix} x \\ 0 \end{bmatrix} \right\|_2$$

```
a = np.array([[0], [1]])
b = np.array([[4], [2]])

Aeq = np.array([0, 1])
beq = 0

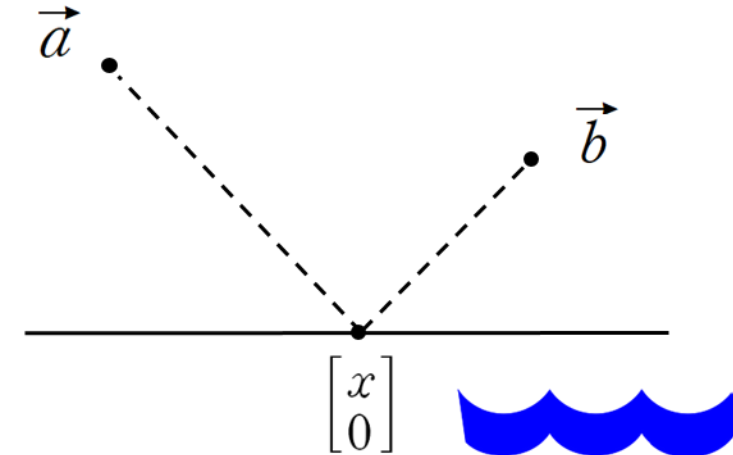
x = cvx.Variable(2, 1)

mu = 1
obj = cvx.Minimize(cvx.norm(a-x, 2) + mu*cvx.norm(b-x, 2))
constraints = [Aeq*x == beq]

prob = cvx.Problem(obj, constraints)
result = prob.solve()

print(x.value)
print(result)
```

```
[[ 1.33325114e+00]
 [ 5.33304239e-12]]
4.9999999941398166
```



# Example: Supply Chain Management

- Find a point that minimizes the sum of the transportation costs (or distance) from this point to 3 destination points

```
a = np.array([[np.sqrt(3)], [0]])
b = np.array([[-np.sqrt(3)], [0]])
c = np.array([[0], [3]])

x = cvx.Variable(2,1)

obj = cvx.Minimize(cvx.norm(a-x, 2) + cvx.norm(b-x, 2) + cvx.norm(c-x, 2))
#obj = cvx.Minimize(cvx.norm(a-x, 1) + cvx.norm(b-x, 1) + cvx.norm(c-x, 1))

prob = cvx.Problem(obj)
result = prob.solve()

print(x.value)
```

```
[[ -1.58674964e-16]
 [ 1.00000001e+00]]
```

