CS-GY 6763: Lecture 9
Online and Stochastic Gradient Descent, Dimension Dependent Optimization

NYU Tandon School of Engineering, Prof. Christopher Musco

STANDARD OPTIMIZATION SETTING

Given a function f to minimize, assume we have:

- **Function oracle**: Evaluate $f(\mathbf{x})$ for any $\mathbf{x}$.
- **Gradient oracle**: Evaluate $\nabla f(\mathbf{x})$ for any $\mathbf{x}$.

Goal: Minimize the number of oracle calls to find $\tilde{\mathbf{x}}$ such that $f(\tilde{\mathbf{x}}) \leq \min_{\mathbf{x}} f(\mathbf{x}) + \epsilon$.

In machine learning applications, $f(\mathbf{x})$ is typically a loss function for a fixed training dataset.

ONLINE AND STOCHASTIC GRADIENT DESCENT

First part of class:

- Basics of an alternative setting: Online Learning + Optimization.
- Introduction to Regret Analysis.
- Application to analyzing Stochastic Gradient Descent.

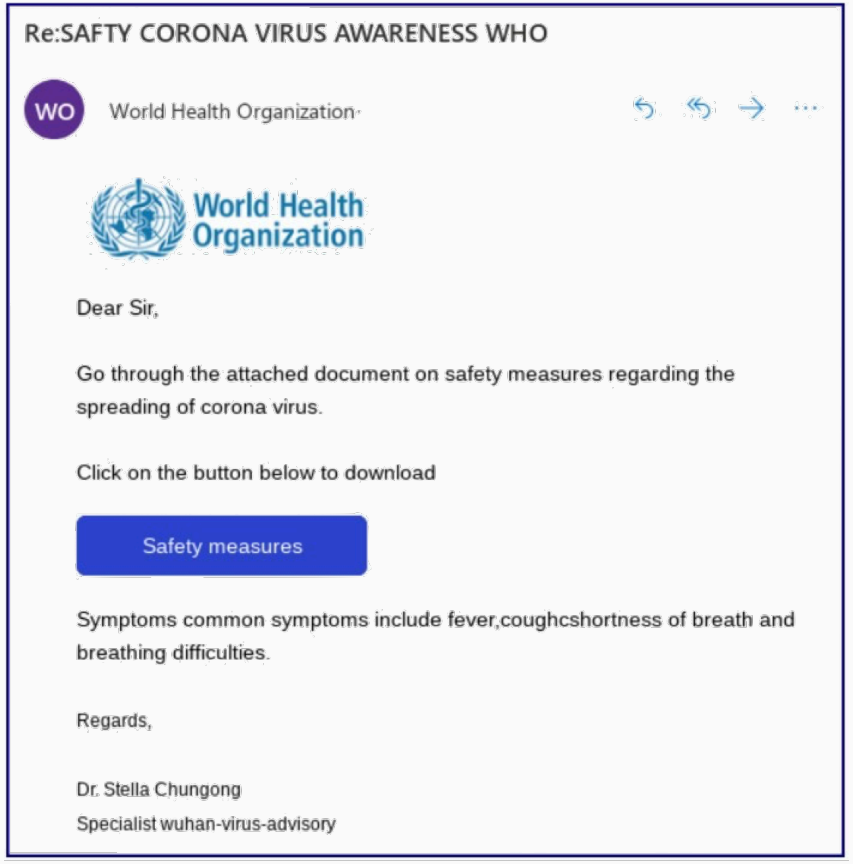
ONLINE LEARNING

Many machine learning problems are solved in an online setting with constantly changing data.

- Spam filters are incrementally updated and adapt as they see more examples of spam over time.
- Text recommendation engines (e.g. Github Copilot) need to be kept up-to-date as software libraries/APIs change.
- Content recommendation systems adapt to user behavior and clicks (which may not be a good thing...)

EXAMPLE

Machine learning based email spam filtering.



Re:SAFTY CORONA VIRUS AWARENESS WHO

WO World Health Organization

World Health Organization

Dear Sir,

Go through the attached document on safety measures regarding the spreading of corona virus.

Click on the button below to download

Safety measures

Symptoms common symptoms include fever,coughcshortness of breath and breathing difficulties.

Regards,

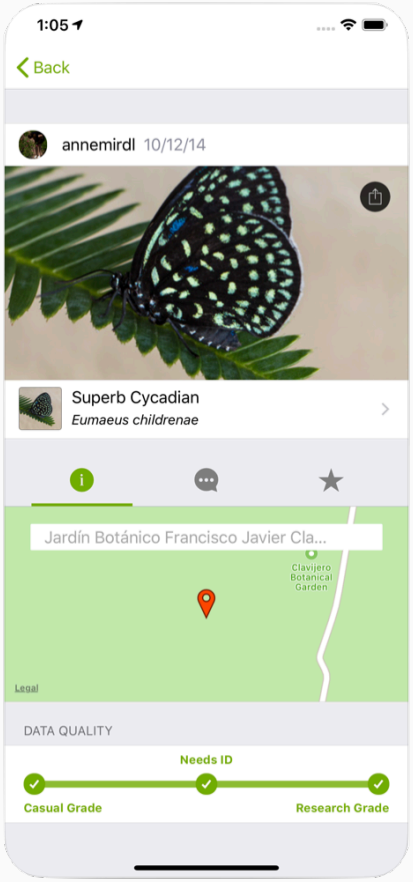
Dr. Stella Chungong
Specialist wuhan-virus-advisory

Markers for spam change overtime, so model might change.

EXAMPLE

Plant identification via iNaturalist app.

(California Academy of Science + National Geographic)



- When the app fails, image is classified via crowdsourcing (backed by huge network of amateurs and experts).
- Single model that is updated constantly, not retrained in batches.

ONLINE LEARNING FRAMEWORK

Choose some model $M_{\mathbf{x}}$ parameterized by parameters $\mathbf{x}$ and some loss function ℓ. At time steps $1, \ldots, T$, receive data vectors $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(T)}$.

- At each time step, we pick ("play") a parameter vector $\mathbf{x}^{(i)}$.
- Make prediction $\tilde{y}^{(i)} = M_{\mathbf{x}^{(i)}}(\mathbf{a}_i)$.
- Then told true value or label $y^{(i)}$. Possibly use this information to choose a new $\mathbf{x}^{(i+1)}$.
- Goal is to minimize cumulative loss:

$$L = \sum_{i=1}^{n} \ell(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)})$$

For example, for a regression problem we might use the ℓ_2 loss:

$$\ell(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)}) = \left| \langle \mathbf{x}^{(i)}, \mathbf{a}^{(i)} \rangle - y^{(i)} \right|^2.$$

For classification, we could use logistic/cross-entropy loss.

Abstraction as optimization problem: Instead of a single objective function f, we have a single (initially unknown) function $f_1, \ldots, f_T : \mathbb{R}^d \to \mathbb{R}$ for each time step.

- For time step $i \in 1, \ldots, T$, select vector $\mathbf{x}^{(i)}$.
- Observe f_i and pay cost $f_i(\mathbf{x}^{(i)})$
- Goal is to minimize $\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})$.

We make no assumptions that $f_1, \ldots, f_T$ are related to each other at all!

REGRET BOUND

In offline optimization, we wanted to find $\hat{\mathbf{x}}$ satisfying $f(\hat{\mathbf{x}}) \leq \min_{\mathbf{x}} f(\mathbf{x})$. Ask for a similar thing here.

Objective: Choose $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}$ so that:

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x}) \right] + \epsilon.$$

Here ϵ is called the **regret** of our solution sequence $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}$. Regret compares to the best fixed solution in hindsight.

We typically ϵ to be growing sublinearly in T.

Regret compares to the best fixed solution in hindsight.

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x}) \right] + \epsilon.$$

It's very possible that $\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) < \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x}) \right]$. Could we hope for something stronger?

Exercise: Argue that the following is impossible to achieve:

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\sum_{i=1}^{T} \min_{\mathbf{x}} f_i(\mathbf{x}) \right] + \epsilon.$$

HARD EXAMPLE FOR ONLINE OPTIMIZATION

Convex functions:

$$f_1(x) = |x - h_1|$$

$$\vdots$$

$$f_n(x) = |x - h_T|$$

where $h_1, \ldots, h_T$ are i.i.d. uniform $\{0, 1\}$.

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x}) \right] + \epsilon.$$

Beautiful balance:

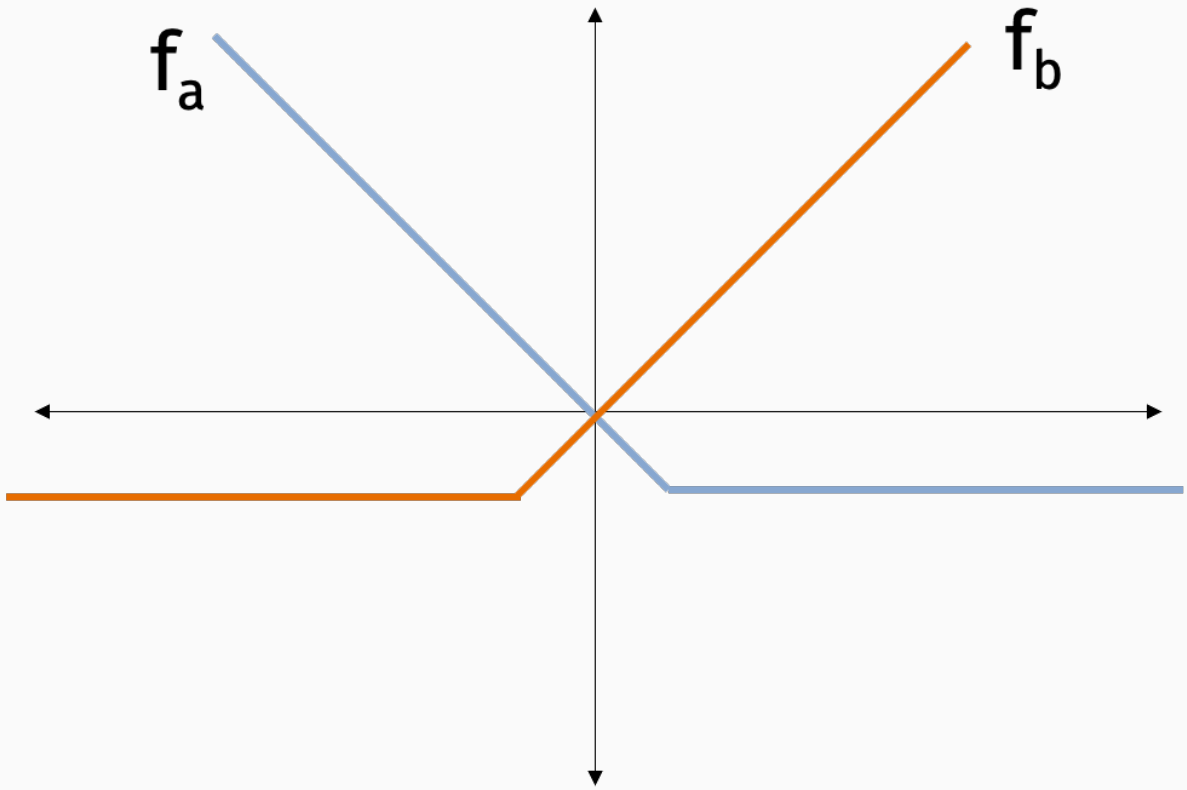
- Either $f_1, \ldots, f_T$ are similar or changing slowly, so we can learn predict f_i from earlier functions.
- Or $f_1, \ldots, f_T$ are very different, in which case $\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ is large, so regret bound is easy to achieve.
- Or we live somewhere in the middle.

Follow-the-leader algorithm:

- Choose $\mathbf{x}^{(0)}$.
- For $i = 1, \ldots, T$:
  - Let $\mathbf{x}^{(i)} = \arg\min_{\mathbf{x}} \sum_{j=1}^{i-1} f_j(\mathbf{x})$.
  - Play $\mathbf{x}^{(i)}$.
  - Observe f_i and incur cost $f_i(\mathbf{x}^{(i)})$.

Simple and intuitive, but there are two issues with this approach. One is computational, one is related to the accuracy.

Hard case:



Online Gradient descent:

- Choose $\mathbf{x}^{(1)}$ and $\eta = \frac{R}{G\sqrt{T}}$.
- For $i = 1, \ldots, T$:
  - Play $\mathbf{x}^{(i)}$.
  - Observe f_i and incur cost $f_i(\mathbf{x}^{(i)})$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_i(\mathbf{x}^{(i)})$

If $f_1, \ldots, f_T = f$ are all the same, this is the same as regular gradient descent. We update parameters using the gradient ∇f at each step.

Online gradient descent (OGD)

$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ (the offline optimum)

Assume:

- $f_1, \ldots, f_T$ are all convex.
- Each is G-Lipschitz: for all $\mathbf{x}$, i, $\|\nabla f_i(\mathbf{x})\|_2 \leq G$.
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

Online Gradient descent:

- Choose $\mathbf{x}^{(1)}$ and $\eta = \frac{R}{G\sqrt{T}}$.
- For $i = 1, \ldots, T$:
  - Play $\mathbf{x}^{(i)}$.
  - Observe f_i and incur cost $f_i(\mathbf{x}^{(i)})$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_i(\mathbf{x}^{(i)})$

Let $\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ (the offline optimum)

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}$.

Average regret overtime is bounded by $\frac{\epsilon}{T} \leq \frac{RG}{\sqrt{T}}$.

Goes $\to 0$ as $T \to \infty$.

All this with no assumptions on how $f_1, \ldots, f_T$ relate to each other! They could have even been chosen adversarially – e.g. with f_i depending on our choice of $\mathbf{x}_i$ and all previous choices.

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}$.

Claim 1: For all $i = 1, \ldots, T$,

$$f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{(i)} - \mathbf{x}^*\|_2^2 - \|\mathbf{x}^{(i+1)} - \mathbf{x}^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

(Same proof for standard GD. Only uses convexity of f_i.)

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}$.

Claim 1: For all $i = 1, \ldots, T$,

$$f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{(i)} - \mathbf{x}^*\|_2^2 - \|\mathbf{x}^{(i+1)} - \mathbf{x}^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

Telescoping Sum:

$$\begin{aligned}\sum_{i=1}^{T} \left[f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*)\right] &\leq \frac{\|\mathbf{x}^{(1)} - \mathbf{x}^*\|_2^2 - \|\mathbf{x}^{(T)} - \mathbf{x}^*\|_2^2}{2\eta} + \frac{T\eta G^2}{2} \\ &\leq \frac{R^2}{2\eta} + \frac{T\eta G^2}{2}\end{aligned}$$

Stochastic gradient descent (SGD)

Efficient offline optimization method for functions f with finite sum structure:

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x}).$$

Goal is to find $\hat{\mathbf{x}}$ such that $f(\hat{\mathbf{x}}) \leq f(\mathbf{x}^*) + \epsilon$.

- The most widely use optimization algorithm in modern machine learning.
- Easily analyzed as a special case of online gradient descent!

Recall the machine learning setup. In empirical risk minimization, we can typically write:

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$$

where f_i is the loss function for a particular data example $(\mathbf{a}^{(i)}, y^{(i)})$.

Example: least squares linear regression.

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\mathbf{x}^T \mathbf{a}^{(i)} - y^{(i)})^2$$

Note that by linearity, $\nabla f(\mathbf{x}) = \sum_{i=1}^{n} \nabla f_i(\mathbf{x})$.

Main idea: Use random approximate gradient in place of actual gradient.

Pick random $j \in 1, \ldots, n$ and update $\mathbf{x}$ using $\nabla f_j(\mathbf{x})$.

$$\mathbb{E}\left[\nabla f_j(\mathbf{x})\right] = \frac{1}{n}\nabla f(\mathbf{x}).$$

$n\nabla f_j(\mathbf{x})$ is an unbiased estimate for the true gradient $\nabla f(\mathbf{x})$, but can typically be computed in a $1/n$ fraction of the time!

Trade slower convergence for cheaper iterations.

Stochastic gradient descent

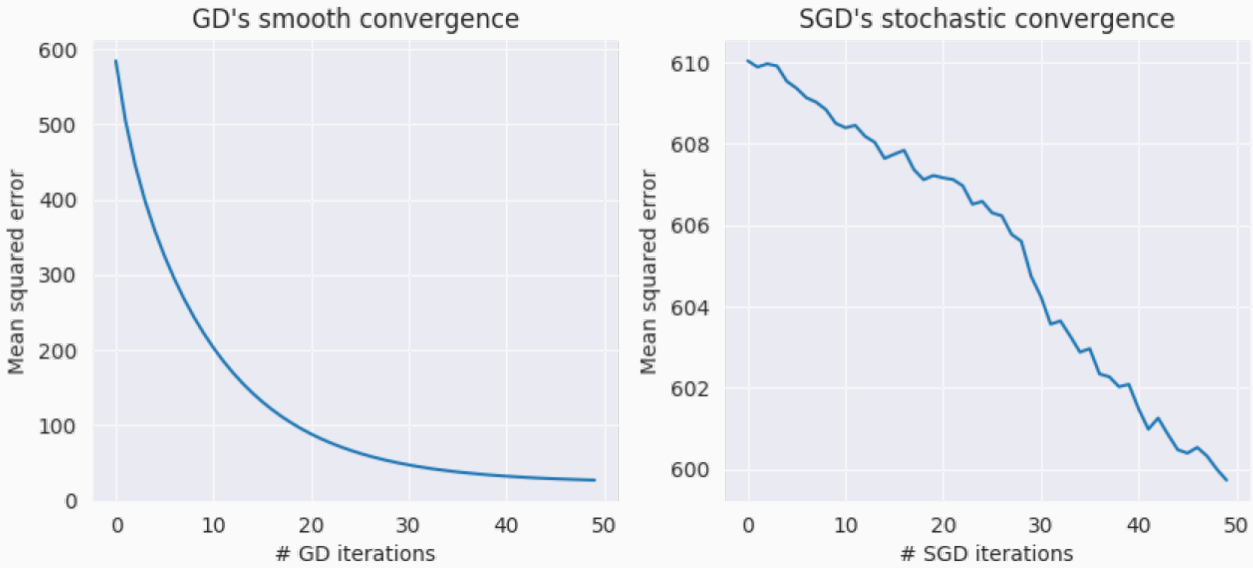
Stochastic first-order oracle for $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$.

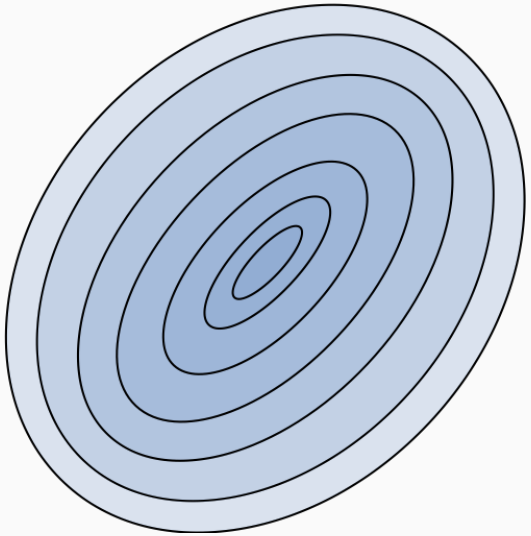
- **Function Query:** For any chosen $j, \mathbf{x}$, return $f_j(\mathbf{x})$
- **Gradient Query:** For any chosen $j, \mathbf{x}$, return $\nabla f_j(\mathbf{x})$

Stochastic Gradient descent:

- Choose starting vector $\mathbf{x}^{(1)}$, step size η
- For $i = 1, \ldots, T$:
  - Pick random j_i uniformly at random from $1, \ldots, n$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_{j_i}(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \frac{1}{T} \sum_{i=1}^{T} \mathbf{x}^{(i)}$

VISUALIZING SGD





STOCHASTIC GRADIENT DESCENT

Assume:

- Finite sum structure: $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$, with $f_1, \ldots, f_n$ all convex.
- Lipschitz functions: for all $\mathbf{x}$, j, $\|\nabla f_j(\mathbf{x})\|_2 \leq \frac{G'}{n}$.
  - What does this imply about Lipschitz constant of f?
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

Stochastic Gradient descent:

- Choose $\mathbf{x}^{(1)}$, steps T, step size $\eta = \frac{R}{G'\sqrt{T}}$.
- For $i = 1, \ldots, T$:
  - Pick random $j_i \in 1, \ldots, n$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_{j_i}(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \frac{1}{T} \sum_{i=1}^{T} \mathbf{x}^{(i)}$

Approach: View as online gradient descent run on function sequence $f_{j_1}, \ldots, f_{j_T}$.

Only use the fact that step equals gradient in expectation.

Jensen's inequality

For a convex function f and points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}$

$$f\left(\frac{1}{t} \cdot \mathbf{x}^{(1)} + \ldots + \frac{1}{t} \cdot \mathbf{x}^{(t)}\right) \leq \frac{1}{t} \cdot f(\mathbf{x}^{(1)}) + \ldots + \frac{1}{t} \cdot f(\mathbf{x}^{(t)})$$

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iterations:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

Claim 1:

$$f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq \frac{1}{T}\sum_{i=1}^{T}\left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*)\right]$$

Prove using Jensen's Inequality:

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iterations:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

$$\begin{aligned}
\mathbb{E}[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)] &\leq \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}\left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*)\right] \\
&= \frac{1}{T}\sum_{i=1}^{T} n\mathbb{E}\left[f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^*)\right] \\
&= \frac{n}{T} \cdot \mathbb{E}\left[\sum_{i=1}^{T} f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^*)\right]
\end{aligned}$$

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iterations:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

$$\begin{aligned}
\mathbb{E}[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)] &\leq \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}\left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*)\right] \\
&= \frac{1}{T}\sum_{i=1}^{T} n\mathbb{E}\left[f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^*)\right] \\
&\leq \frac{n}{T} \cdot \mathbb{E}\left[\sum_{i=1}^{T} f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^{offline})\right],
\end{aligned}$$

where $\mathbf{x}^{offline} = \arg\min_{\mathbf{x}} \sum_{i=1}^{T} f_{j_i}(\mathbf{x})$.

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iterations:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

$$\begin{aligned}
\mathbb{E}[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)] &\leq \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}\left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*)\right] \\
&= \frac{1}{T}\sum_{i=1}^{T} n\mathbb{E}\left[f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^*)\right] \\
&\leq \frac{n}{T}\mathbb{E}\left[\sum_{i=1}^{T} f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^{offline})\right] \\
&\leq \frac{n}{T} \cdot \left(R \cdot \frac{G'}{n} \cdot \sqrt{T}\right) \qquad \text{(by OGD guarantee.)}
\end{aligned}$$

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

Number of iterations for error ϵ:

- **Gradient Descent**: $T = \frac{R^2 G^2}{\epsilon^2}$.
- **Stochastic Gradient Descent**: $T = \frac{R^2 G'^2}{\epsilon^2}$.

Always have $G \leq G'$:

$$\begin{aligned}\max_{\mathbf{x}} \|\nabla f(\mathbf{x})\|_2 &\leq \max_{\mathbf{x}} \left(\|\nabla f_1(\mathbf{x})\|_2 + \ldots + \|\nabla f_n(\mathbf{x})\|_2\right) \\ &\leq \max_{\mathbf{x}} \left(\|\nabla f_1(\mathbf{x})\|_2\right) + \ldots + \max_{\mathbf{x}} \left(\|\nabla f_n(\mathbf{x})\|_2\right) \\ &\leq n \cdot \frac{G'}{n} = G'.\end{aligned}$$

So GD converges strictly faster than *SGD*.

But for a fair comparison:

- SGD cost = (# of iterations) $\cdot\, O(1)$
- GD cost = (# of iterations) $\cdot\, O(n)$

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

We always have $G \leq G'$. When it is much smaller then GD will perform better. When it is closer to this upper bound, SGD will perform better.

What is an extreme case where $G = G'$?

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

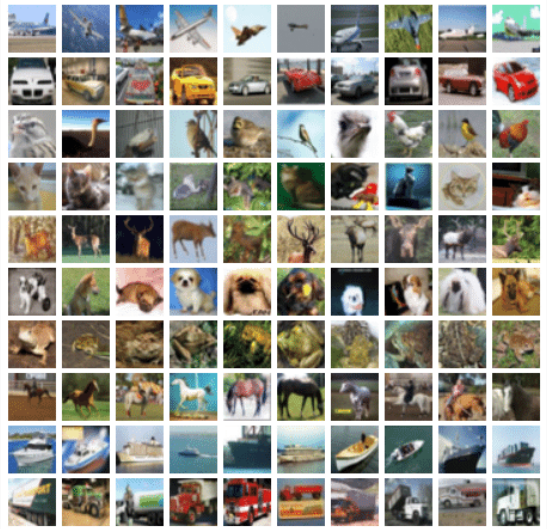
What if each gradient $\nabla f_i(\mathbf{x})$ looks like random vectors in $\mathbb{R}^d$? E.g. with $\mathcal{N}(0,1)$ entries?

$$\mathbb{E}\left[\|\nabla f_i(\mathbf{x})\|_2^2\right] =$$

$$\mathbb{E}\left[\|\nabla f(\mathbf{x})\|_2^2\right] = \mathbb{E}\left[\|\sum_{i=1}^{n} \nabla f_i(\mathbf{x})\|_2^2\right] =$$

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

Takeaway: SGD performs better when there is more structure or repetition in the data set.



PRECONDITIONING

Main idea: Instead of minimizing $f(\mathbf{x})$, find another function $g(\mathbf{x})$ with the same minimum but which is better suited for first order optimization (e.g., has a smaller conditioner number).

Claim: Let $h(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$ be an invertible function. Let $g(\mathbf{x}) = f(h(\mathbf{x}))$. Then

$$\min_{\mathbf{x}} f(\mathbf{x}) = \min_{\mathbf{x}} g(\mathbf{x}) \quad \text{and} \quad \arg\min_{\mathbf{x}} f(\mathbf{x}) = h\left(\arg\min_{\mathbf{x}} g(\mathbf{x})\right).$$

First Requirement: We need $g(\mathbf{x})$ to still be convex.

Claim: Let $\mathbf{P}$ be an invertible $d \times d$ matrix and let $g(\mathbf{x}) = f(\mathbf{P}\mathbf{x})$.

$g(\mathbf{x})$ is always convex.

For this choice of preconditioner, if $\tilde{\mathbf{x}} \approx \arg\min g(\mathbf{x})$, we would want to return $\mathbf{P}\tilde{\mathbf{x}}$ as a near minimizer of f.

Additional Goals:

- $g(\mathbf{x})$ should be better conditioned (more smooth, more strongly convex, etc.) than f.
- $\mathbf{P}$ needs to be easy to compute and we need to be able to apply $\mathbf{Px}$ efficiently.

Common choice: Diagonal preconditioner.

- Choose $\mathbf{P}$ to be a diagonal matrix $\mathbf{D}$.
- For $f(\mathbf{x}) = \|\mathbf{Ax} - \mathbf{b}\|_2^2$, common choice is $\mathbf{D} = \mathbf{diag}(\mathbf{A}^T\mathbf{A})^{-1}$, which is known as the **Jacobi preconditioner**.

Often works very well in practice!

DIAGONAL PRECONDITIONER

```
A =

    -734     1    33    9111     0
     -31    -2   108    5946   -19
     232    -1   101    3502    10
     426     0   -65   12503     9
    -373     0    26    9298     0
    -236    -2   -94    2398    -1
    2024     0  -132   -6904   -25
   -2258    -1    92   -6516     6
    2229     0     0   11921   -22
     338     1    -5  -16118   -23
```

```
>> cond(A'*A)

ans =

   8.4145e+07
```

```
>> P = sqrt(inv(diag(diag(A'*A))));
>> cond(P*A'*A*P)

ans =

   10.3878
```

Another view: If $g(\mathbf{x}) = f(\mathbf{P}\mathbf{x})$ then $\nabla g(\mathbf{x}) = \mathbf{P}^T \nabla f(\mathbf{P}\mathbf{x})$.

$\nabla g(\mathbf{x}) = \mathbf{P} \nabla f(\mathbf{P}\mathbf{x})$ when $\mathbf{P}$ is symmetric.

Gradient descent on g:

- For $t = 1, \ldots, T$,
  - $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \mathbf{P} \left[\nabla f(\mathbf{P}\mathbf{x}^{(t)}) \right]$

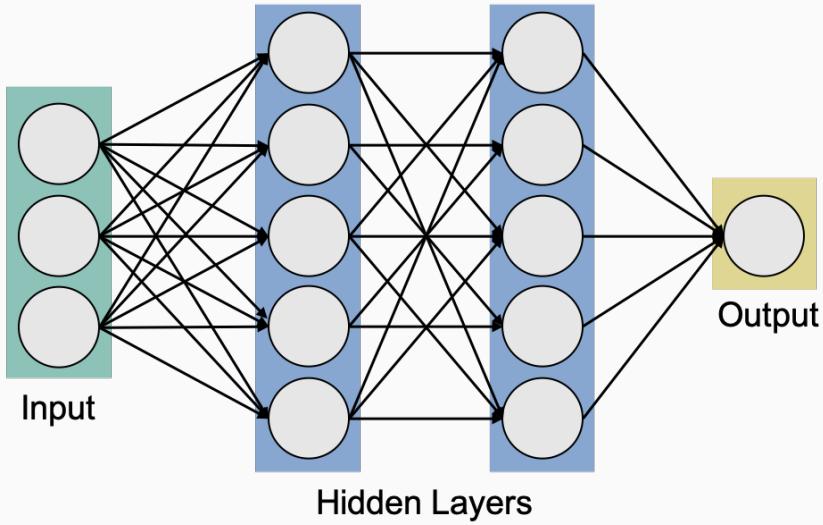
Gradient descent on g:

- For $t = 1, \ldots, T$,
  - $\mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} - \eta \mathbf{P}^2 \left[\nabla f(\mathbf{y}^{(t)}) \right]$

When $\mathbf{P}$ is diagonal, this is just gradient descent with a different step size for each parameter!

Algorithms based on this idea:

- AdaGrad
- RMSprop
- Adam optimizer



(Pretty much all of the most widely used optimization methods for training neural networks.)

BREAK

First Order Optimization: Given a convex function f and a convex set $\mathcal{S}$,

$$\textbf{Goal: Find } \hat{\mathbf{x}} \in \mathcal{S} \text{ such that } f(\hat{\mathbf{x}}) \leq \min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x}) + \epsilon.$$

Assume we have:

- **Function oracle**: Evaluate $f(\mathbf{x})$ for any $\mathbf{x}$.
- **Gradient oracle**: Evaluate $\nabla f(\mathbf{x})$ for any $\mathbf{x}$.
- **Projection oracle**: Evaluate $P_{\mathcal{S}}(\mathbf{x})$ for any $\mathbf{x}$.

Gradient descent requires $O\left(\frac{R^2 G^2}{\epsilon^2}\right)$ calls to each oracle to solve the problem.

We were only able to improve the ϵ dependence by making stronger assumptions on f (strong convexity, smoothness).

DIMENSION DEPENDENT BOUND

Alternatively, we can get much better bounds if we are willing to depend on the problem dimension. I.e. on d if $f(\mathbf{x})$ is a function mapping d-dimensional vectors to scalars.

We already know how to do this for a few special functions:

$$f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \qquad \text{where} \qquad \mathbf{A} \in \mathbb{R}^{n \times d}.$$

DIMENSION DEPENDENT BOUND

Let $f(\mathbf{x})$ be bounded between $[-B, B]$ on $\mathcal{S}$.

Theorem (Dimension Dependent Convex Optimization)

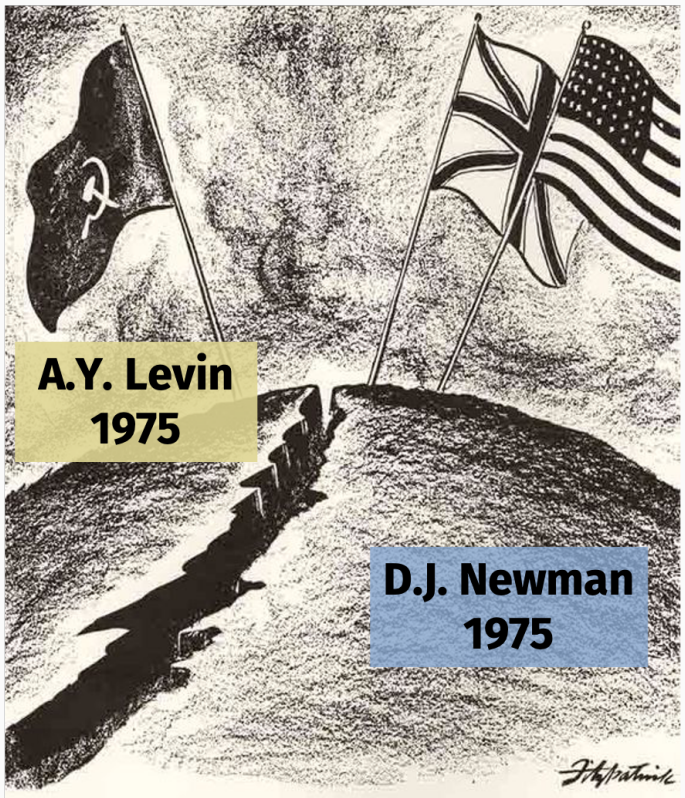
There is an algorithm (the Center-of-Gravity Method) which finds $\hat{\mathbf{x}}$ satisfying $f(\hat{\mathbf{x}}) \leq \min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x}) + \epsilon$ using $O(d \log(B/\epsilon))$ calls to a function and gradient oracle for convex f.

Caveat: Assumes we have some representation of $\mathcal{S}$, not just a projection oracle. We will discuss this more later.

Note: For an unconstrained problem with known starting radius R, can take $\mathcal{S}$ to be the ball of radius R around $\mathbf{x}^{(1)}$. If $\|\nabla f(\mathbf{x})\|_2 \leq G$, we always have $B = O(RG)$.

CENTER OF GRAVITY METHOD

Natural “cutting plane” method. Developed simultaneous on opposite sides of iron curtain.



Not used in practice (we will discuss why) but the basic idea underlies many popular algorithms.

A few basic ingredients:

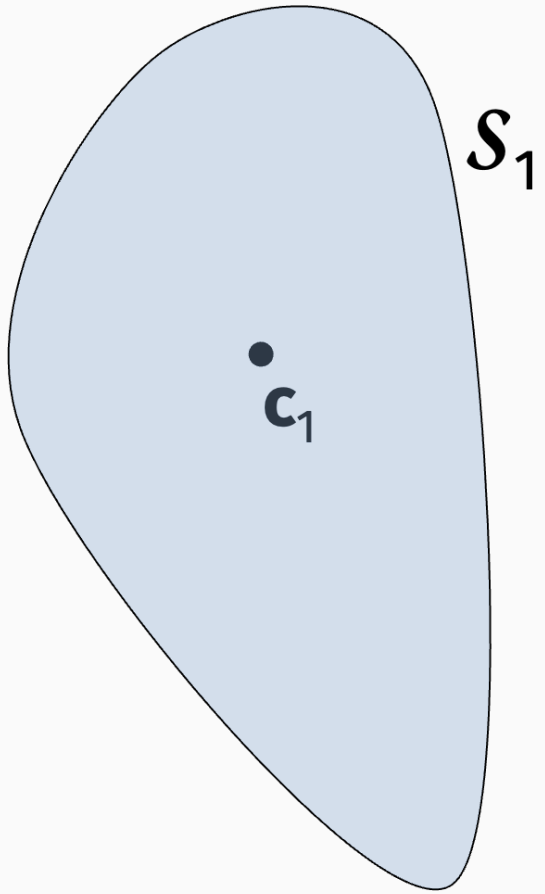
1. The center-of-gravity of a convex set $\mathcal{S}$ is defined as:

$$c = \frac{\int_{x \in \mathcal{S}} x \, dx}{\text{vol}(\mathcal{S})} = \frac{\int_{x \in \mathcal{S}} x \, dx}{\int_{x \in \mathcal{S}} 1 \, dx}$$

2. For two convex sets $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A} \cap \mathcal{B}$ is convex. Proof by picture:

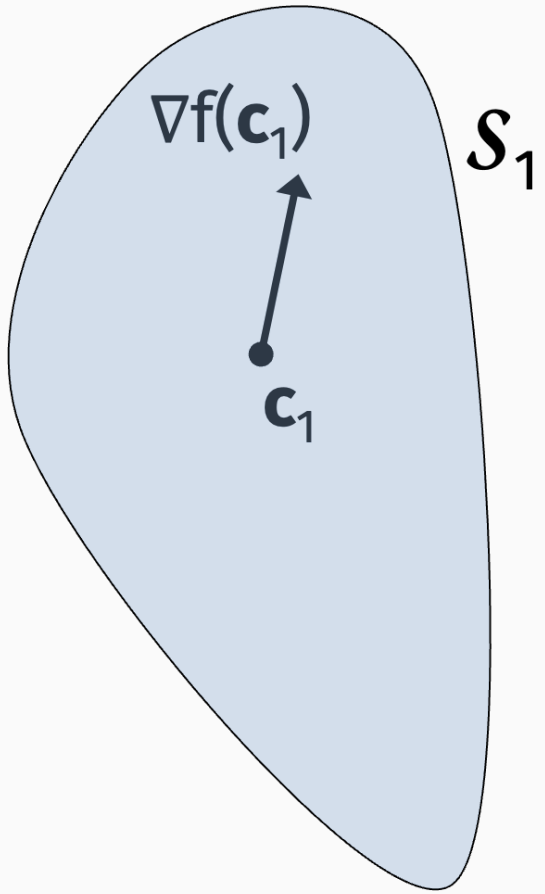
Natural "cutting plane" method.

- $\mathcal{S}_1 = \mathcal{S}$
- For $t = 1, \ldots, T$:
  - $\mathbf{c}_t$ = center of gravity of $\mathcal{S}_t$.
  - Compute $\nabla f(\mathbf{c}_t)$.
  - $\mathcal{H} = \{\mathbf{x} \big| \langle \nabla f(\mathbf{c}_t), \mathbf{x} - \mathbf{c}_t \rangle \leq 0\}$.
  - $\mathcal{S}_{t+1} = \mathcal{S}_t \cap H$
- Return $\hat{\mathbf{x}} = \arg\min_t f(\mathbf{c}_t)$



Natural “cutting plane” method.

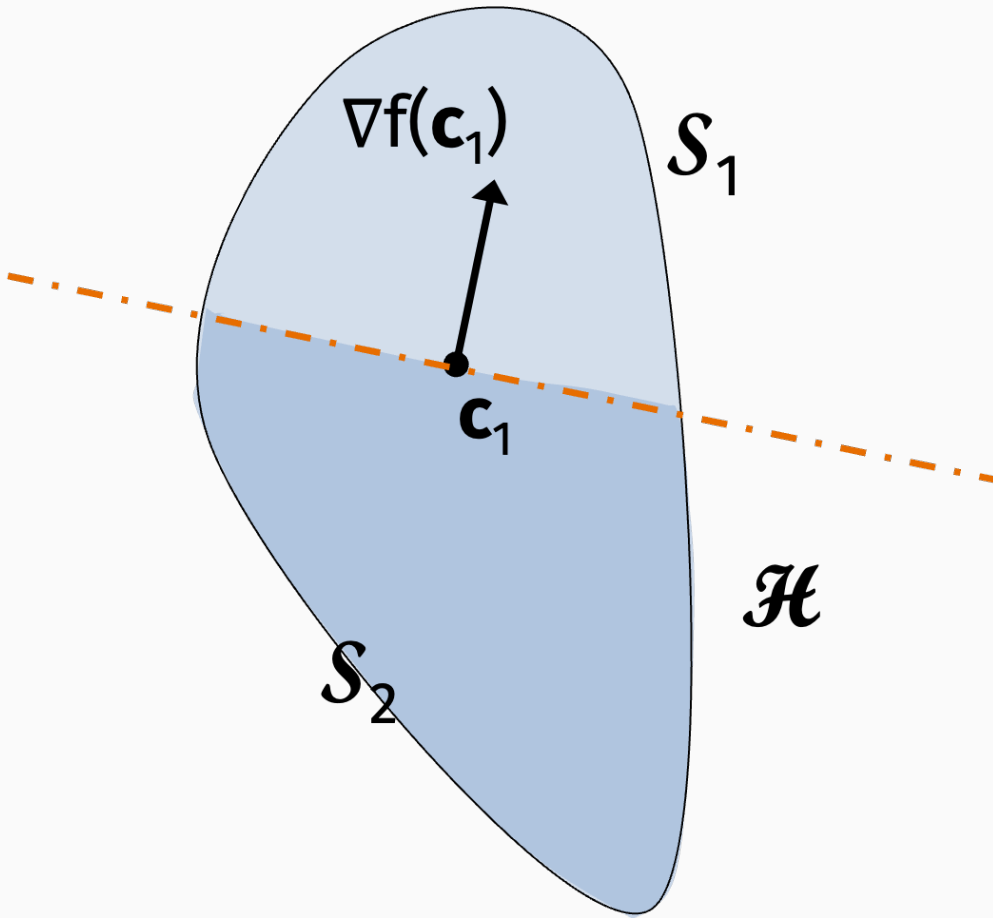
- $\mathcal{S}_1 = \mathcal{S}$
- For $t = 1, \ldots, T$:
  - $\mathbf{c}_t =$ center of gravity of $\mathcal{S}_t$.
  - Compute $\nabla f(\mathbf{c}_t)$.
  - $\mathcal{H} = \{\mathbf{x} \mid \langle \nabla f(\mathbf{c}_t), \mathbf{x} - \mathbf{c}_t \rangle \leq 0\}$.
  - $\mathcal{S}_{t+1} = \mathcal{S}_t \cap H$
- Return $\hat{\mathbf{x}} = \arg\min_t f(\mathbf{c}_t)$



CENTER OF GRAVITY METHOD

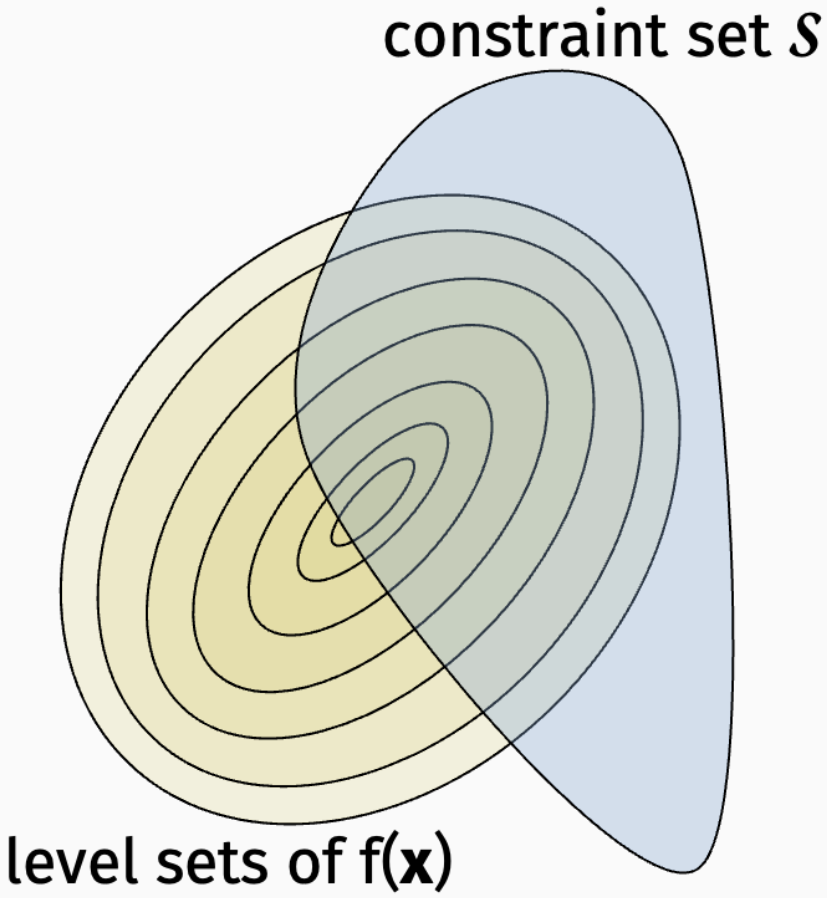
Natural “cutting plane” method.

- $\mathcal{S}_1 = \mathcal{S}$
- For $t = 1, \ldots, T$:
  - $\mathbf{c}_t$ = center of gravity of $\mathcal{S}_t$.
  - Compute $\nabla f(\mathbf{c}_t)$.
  - $\mathcal{H} = \{\mathbf{x} \mid \langle \nabla f(\mathbf{c}_t), \mathbf{x} - \mathbf{c}_t \rangle \leq 0\}$.
  - $\mathcal{S}_{t+1} = \mathcal{S}_t \cap H$
- Return $\hat{\mathbf{x}} = \arg\min_t f(\mathbf{c}_t)$



Intuitively, why does it make sense to search in $\mathcal{S}_t \cap \mathcal{H}$ where:

$$\mathcal{H} = \{\mathbf{x} \big| \langle \nabla f(\mathbf{c}_t), \mathbf{x} - \mathbf{c}_t \rangle \leq 0\}?$$



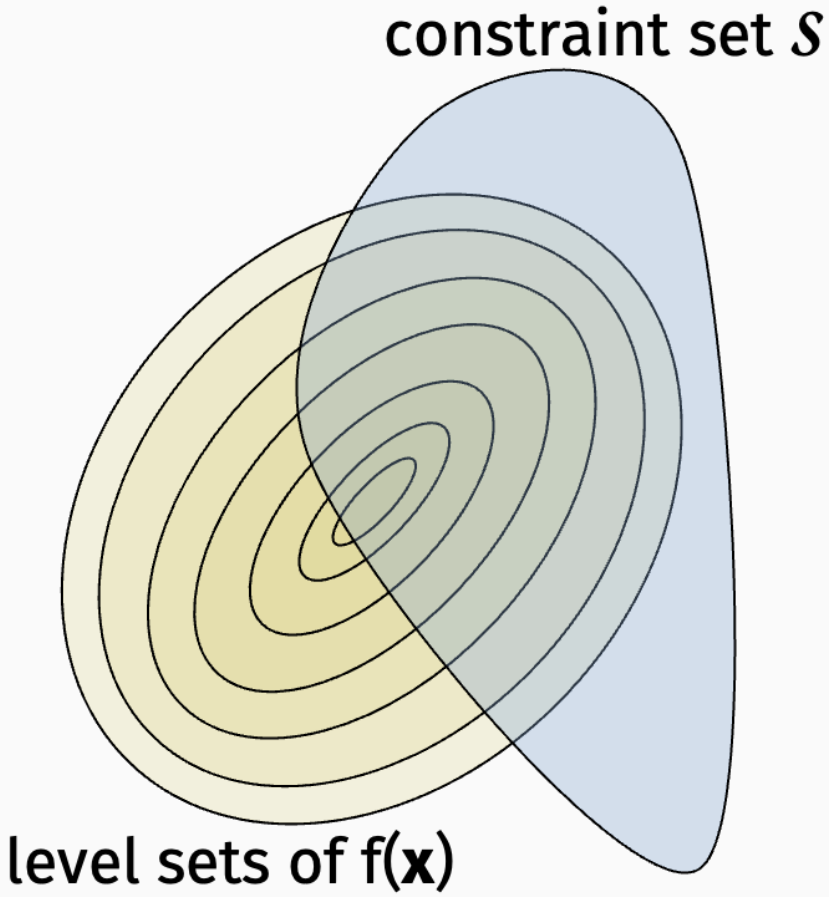
Intuitively, why does it make sense to search in $\mathcal{S}_t \cap \mathcal{H}$ where:

$$\mathcal{H} = \{\mathbf{x} \big| \langle \nabla f(\mathbf{c}_t), \mathbf{x} - \mathbf{c}_t \rangle \leq 0\}?$$

By convexity,

$f(\mathbf{y}) \geq f(\mathbf{c}_t) + \langle \nabla f(\mathbf{c}_t), \mathbf{y} - \mathbf{c}_t \rangle$.

If $\mathbf{y} \notin \{\mathcal{S}_t \cap \mathcal{H}\}$ then $\langle \nabla f(\mathbf{c}_t), \mathbf{y} - \mathbf{c}_t \rangle$ is negative, so $f(\mathbf{y}) > f(\mathbf{c}_t)$.



Theorem (Center-of-Gravity Convergence)

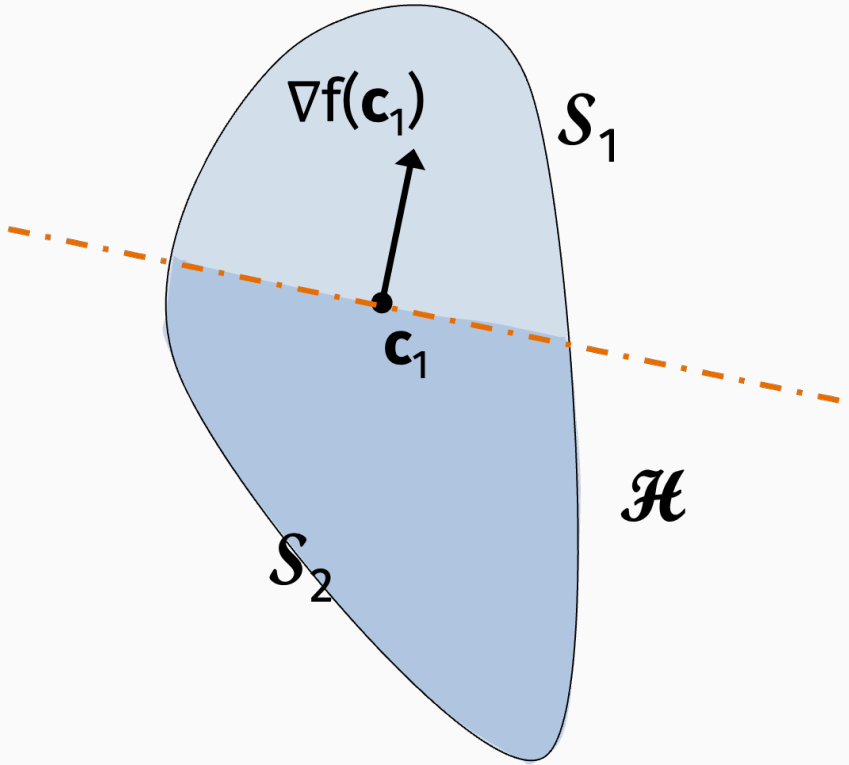
Let f be a convex function with values in $[-B, B]$. Let $\hat{\mathbf{x}}$ be the output of the center-of-gravity method run for T iterations. Then:

$$f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq 2B \left(1 - \frac{1}{e}\right)^{T/d} \leq 2Be^{-T/3d}.$$

If we set $T = 3d \log(2B/\epsilon)$, then $f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq \epsilon$.

KEY GEOMETRIC TOOL

Want to argue that, at every step of the algorithm, we "cut off" a large portion of the convex set we are searching over:



Theorem (Grünbaum's Theorem)

For any convex set $\mathcal{S}$ with center-of-gravity c*, and any halfspace* $\mathcal{Z} = \{\mathsf{x} \big| \langle \mathsf{a}, \mathsf{x} - \mathsf{c} \rangle \leq 0\}$ *then:*

$$\frac{\mathsf{vol}(\mathcal{S} \cap \mathcal{Z})}{\mathsf{vol}(\mathcal{S})} \geq \frac{1}{e} \approx .368$$

Want to argue that, at every step of the algorithm, we "cut off" a large portion of the convex set we are searching over.

Theorem (Grünbaum's Theorem)

For any convex set $\mathcal{S}$ with center-of-gravity $\mathbf{c}$*, and any halfspace* $\mathcal{Z} = \{\mathbf{x} \big| \langle \mathbf{a}, \mathbf{x} - \mathbf{c} \rangle \leq 0\}$ *then:*

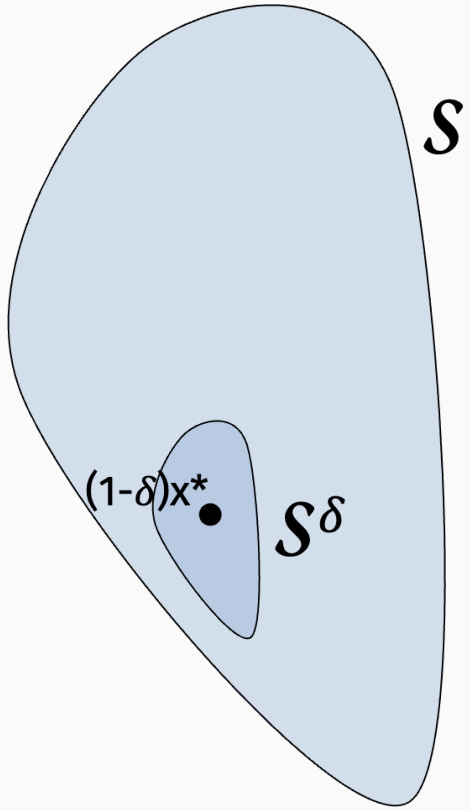
$$\frac{\text{vol}(\mathcal{S} \cap \mathcal{Z})}{\text{vol}(\mathcal{S})} \geq \frac{1}{e} \approx .368$$

Let $\mathcal{Z}$ be the compliment of $\mathcal{H}$ from the algorithm. Then we cut off at least a $1/e$ fraction of the convex body on every iteration.

Corollary: After t steps, $\text{vol}(\mathcal{S}_t) \leq \left(1 - \frac{1}{e}\right)^t \text{vol}(\mathcal{S})$.

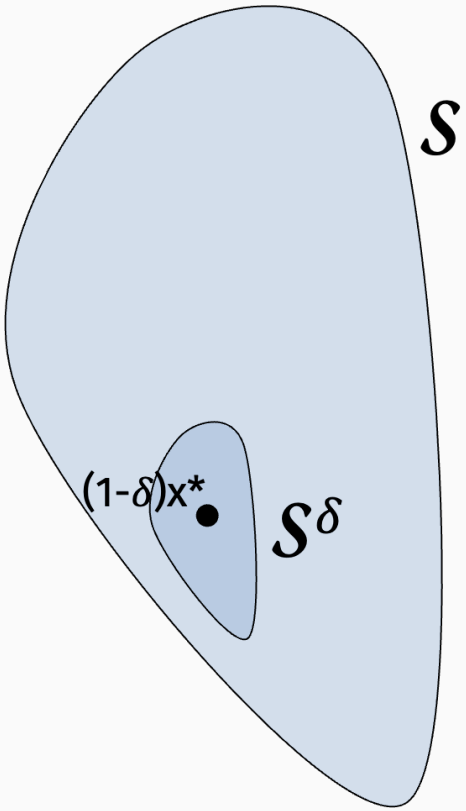
Let δ be a small parameter to be chosen later.

Let $\mathcal{S}^\delta = \{(1-\delta)\mathbf{x}^* + \delta\mathbf{x} \,\big|\, \text{for } \mathbf{x} \in \mathcal{S}\}$.



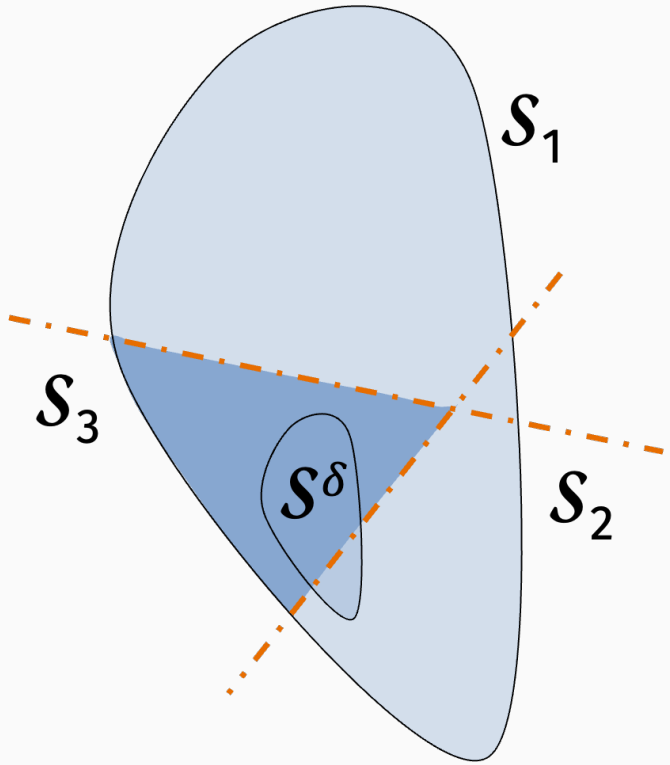
Claim: Every point $\mathbf{y}$ in $\mathcal{S}^\delta$ has good function value.

CONVERGENCE PROOF



For any $\mathbf{y} \in \mathcal{S}^\delta$:

$$\begin{aligned} f(\mathbf{y}) &= f((1-\delta)\mathbf{x}^* + \delta\mathbf{x}) \\ &\leq (1-\delta)f(\mathbf{x}^*) + \delta f(\mathbf{x}) \\ &\leq f(\mathbf{x}^*) - \delta f(\mathbf{x}^*) + \delta f(\mathbf{x}) \\ &\leq f(\mathbf{x}^*) + 2B\delta. \end{aligned}$$

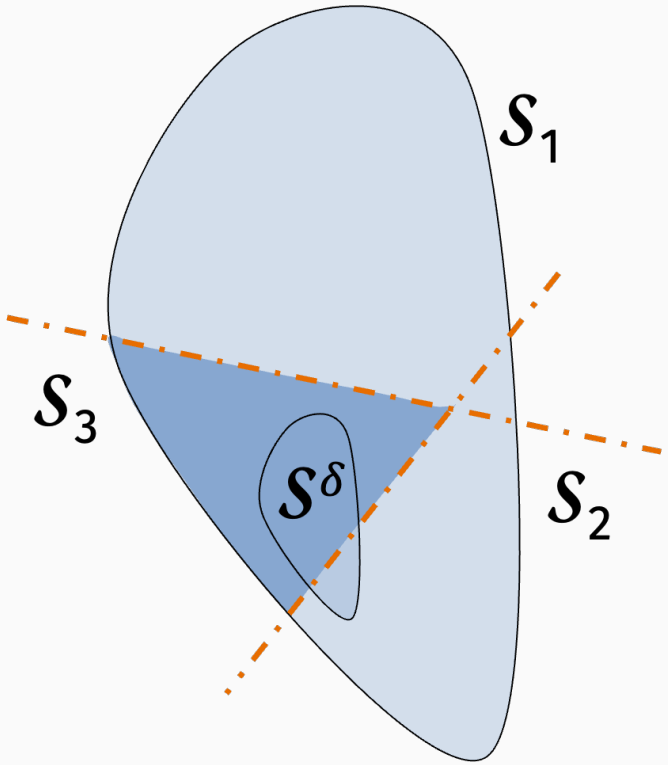


We also have: $\mathsf{vol}(\mathcal{S}^\delta) = \delta^d \, \mathsf{vol}(\mathcal{S})$.

Set $\delta = \left(1 - \frac{1}{e}\right)^{T/d}$. After T steps, $\mathsf{vol}(\mathcal{S}_t) \leq \left(1 - \frac{1}{e}\right)^T = \mathsf{vol}(\mathcal{S}^\delta)$.

Either S_t exactly equals S^δ, in which case our next centroid gives error $\leq 2B\delta$.

Or we must have "chopped off" at least one point $\mathbf{y}$ in $\mathcal{S}^\delta$ by the time we reach step T.



Claim: If we "chopped off" at least one point $\mathbf{y}$ in $\mathcal{S}^\delta$ by the time we reach step T then for some centroid $\mathbf{c}_1, \ldots, \mathbf{c}_t$, $f(\mathbf{c}_t) < 2B\delta$.

Proof:

$$\begin{aligned} 2B\delta \geq f(\mathbf{y}) &\geq f(\mathbf{c}_t) + \langle \nabla f(\mathbf{c}_t), \mathbf{y} - \mathbf{c}_t \rangle \\ &> f(\mathbf{c}_t). \end{aligned}$$

Algorithm returns $\arg\min_{\mathbf{c}_i} f(\mathbf{c}_i)$.

Theorem (Center-of-Gravity Convergence)

Let f be a convex function with values in $[-B, B]$. Let $\hat{\mathbf{x}}$ be the output of the center-of-gravity method run for T iterations. Then:

$$f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq 2B \left(1 - \frac{1}{e}\right)^{T/d} \leq 2Be^{-T/3d}.$$

If we set $T = O\left(d \log(B/\epsilon)\right)$, then $f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq \epsilon$.

In terms of gradient-oracle complexity, this is essentially optimal. So why isn't the algorithm used?

In general computing the centroid is hard. #P-hard even when when $\mathcal{S}$ is an intersection of half-spaces (a polytope).

Even if the problem isn't hard for your starting convex body $\mathcal{S}$, it likely will become hard for $\mathcal{S} \cap \mathcal{H}_1 \cap \mathcal{H}_2 \cap \mathcal{H}_3 \ldots$.

So while the oracle complexity of dimension-dependent optimization was settled in the 70s, a number of basic questions in terms of computational complexity.

We will discuss how to obtain a computationally efficient version of the center-of-gravity method called the **ellipsoid method.** This method is most famous for giving the first polynomial time algorithm for linear programming.

BREAK

Linear programs (LPs) are one of the most basic convex constrained, convex optimization problems:

Let $\mathbf{c} \in \mathbb{R}^d, \mathbf{b} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{n \times d}$ be fixed vectors that define the problem, and let $\mathbf{x}$ be our variable parameter.

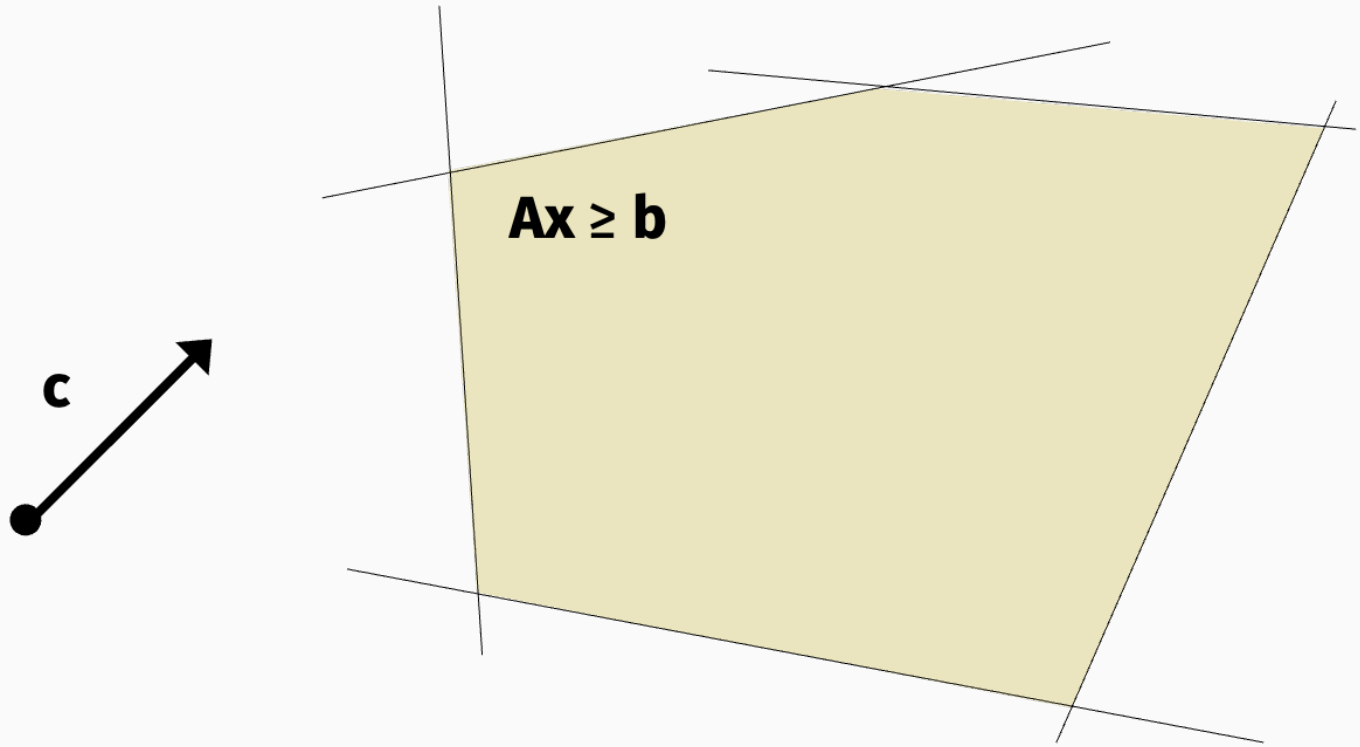
$$\min f(\mathbf{x}) = \mathbf{c}^T \mathbf{x}$$
$$\text{subject to } \mathbf{A}\mathbf{x} \geq \mathbf{b}.$$

Think about $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ as a union of half-space constraints:

$$\{\mathbf{x} : \mathbf{a}_1^T \mathbf{x} \geq b_1\}$$
$$\{\mathbf{x} : \mathbf{a}_2^T \mathbf{x} \geq b_2\}$$
$$\vdots$$
$$\{\mathbf{x} : \mathbf{a}_n^T \mathbf{x} \geq b_n\}$$

$$\min f(\mathbf{x}) = \mathbf{c}^T\mathbf{x}$$

subject to $\mathbf{Ax} \geq \mathbf{b}$.



LINEAR PROGRAMMING APPLICATIONS

- Classic optimization applications: industrial resource optimization problems were killer app in the 70s.
- Robust regression: $\min_x \|Ax - b\|_1$.
- $L1$ constrained regression: $\min_x \|x\|_1$ subject to $Ax = b$. Lots of applications in sparse recovery/compressed sensing.
- Solve $\min_x \|Ax - b\|_\infty$.
- Polynomial time algorithms for Markov Decision Processes.
- Many combinatorial optimization problems can be solved via LP relaxations.

Theorem (Khachiyan, 1979)

Assume $n = d$. The ellipsoid method solves any linear program with L-bit integer valued constraints in $O(n^4 L)$ time.

Ellipsoid is a relatively simple center-of-gravity like method!

A Soviet Discovery Rocks World of Mathematics

By MALCOLM W. BROWNE

A surprise discovery by an obscure Soviet mathematician has rocked the world of mathematics and computer analysis, and experts have begun exploring its practical applications.

Mathematicians describe the discovery by L.G. Khachian as a method by which computers can find guaranteed solutions to a class of very difficult problems that have hitherto been tackled on a kind of hit-or-miss basis.

Apart from its profound theoretical interest, the discovery may be applicable in weather prediction, complicated industrial processes, petroleum refining, the scheduling of workers at large factories, secret codes and many other things.

"I have been deluged with calls from virtually every department of government for an interpretation of the significance of this," a leading expert on computer methods, Dr. George B. Dantzig of Stanford University, said in an interview.

The solution of mathematical problems by computer must be broken down into a series of steps. One class of problem sometimes involves so many steps that it could take billions of years to compute.

The Russian discovery offers a way by which the number of steps in a solution can be dramatically reduced. It also offers the mathematician a way of learning quickly whether a problem has a solution or not, without having to complete the entire immense computation that may be required.

According to the American journal Sci-

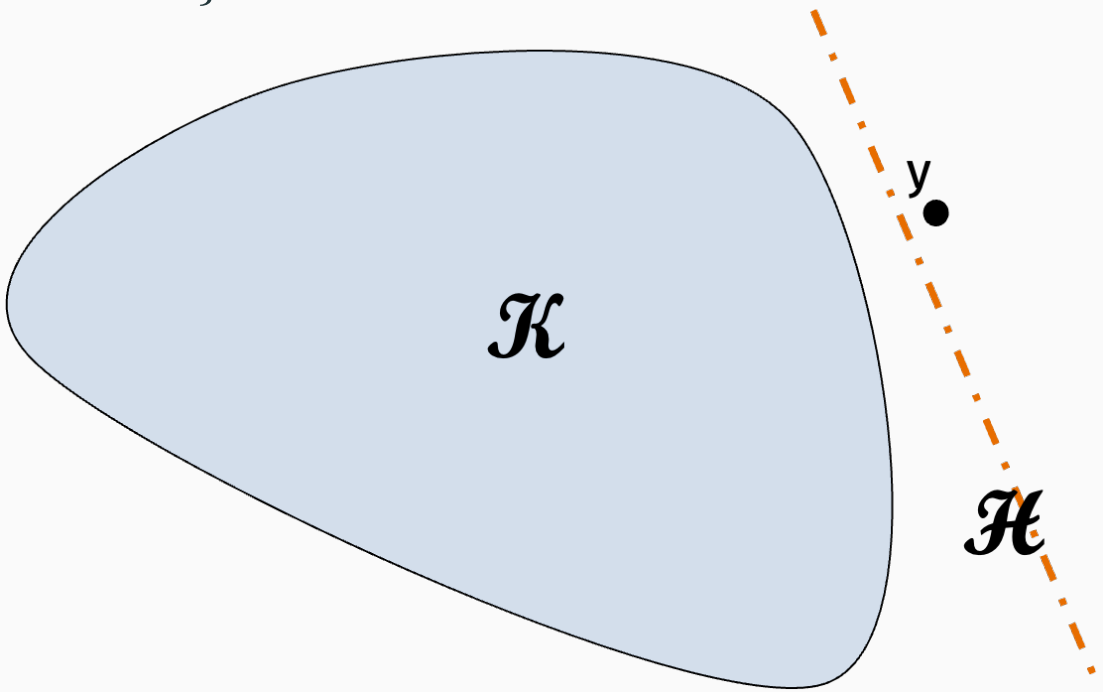
Continued on Page A20, Column 3

Front page of New York Times, November 9, 1979.

Simplifying the problem: Given a convex set $\mathcal{K}$ via access to separation oracle $S_{\mathcal{K}}$ for the set, determine if $\mathcal{K}$ is empty, or otherwise return any point $\mathbf{x} \in \mathcal{K}$.

$$S_k(\mathbf{y}) = \begin{cases} \emptyset & \text{if } \mathbf{y} \in \mathcal{K}. \\ \text{separating hyperplane } (\mathbf{a}, c) & \text{if } \mathbf{y} \notin \mathcal{K}. \end{cases}$$

Let $\mathcal{H} = \{\mathbf{x} : \mathbf{a}^T\mathbf{x} = c\}$.

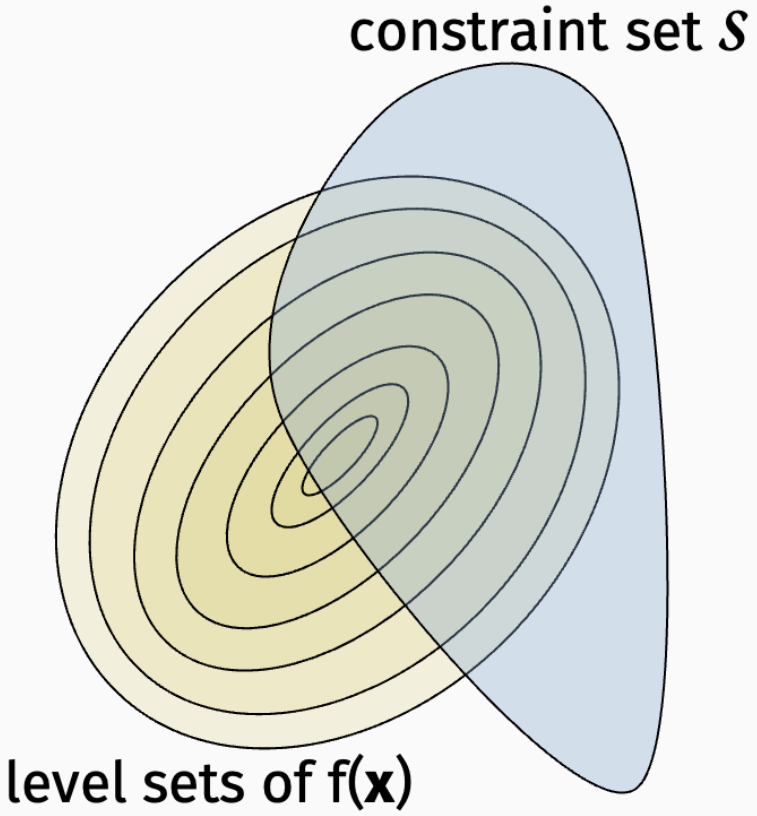


SEPARATION ORACLE

Example: How would you implement a separation oracle for a polytope $\{\mathbf{x} : \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$.

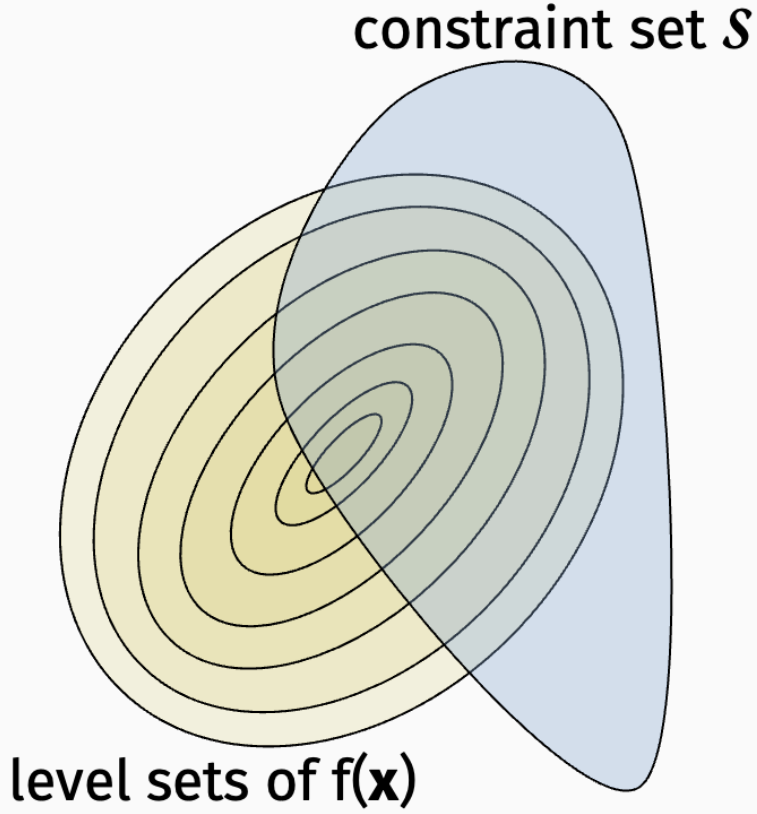
Original problem: $\min_{x \in \mathcal{S}} f(x)$.

How to reduce to determining if a convex set $\mathcal{K}$ is empty or not?



Original problem: $\min_{x \in \mathcal{S}} f(x)$.

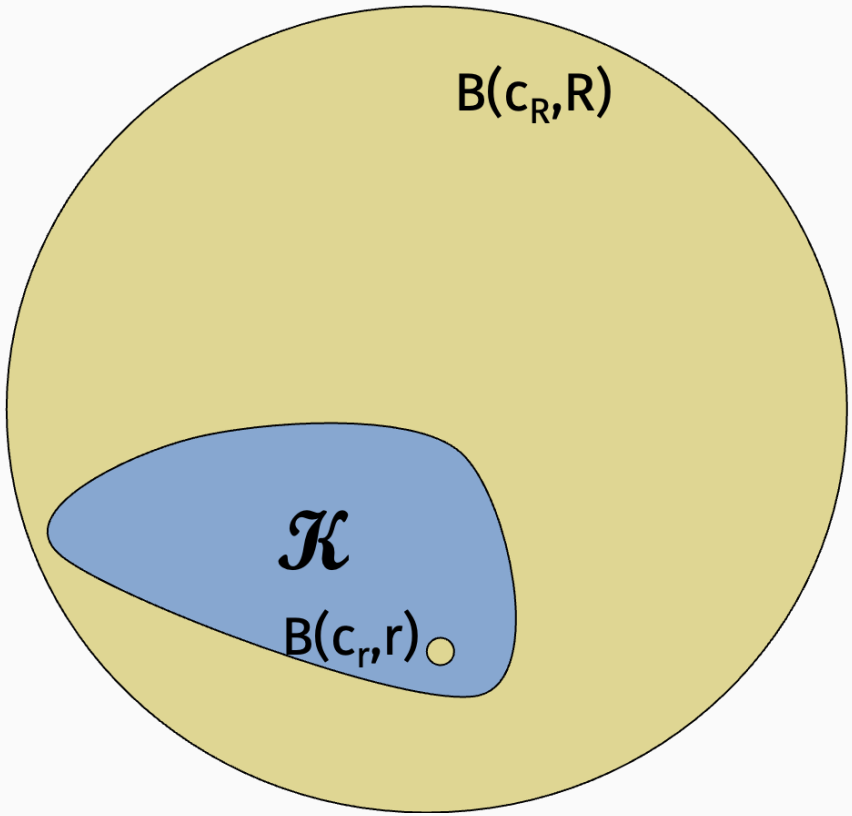
How to reduce to determining if a convex set $\mathcal{K}$ is empty or not?



Binary search! For a convex function $f(x)$, $\{x : f(x) \leq c\}$ is convex, and you can get a separation oracle via the gradient.

Goal of ellipsoid algorithm: Solve "Is $\mathcal{K}$ empty or not?" given a separation oracle for $\mathcal{K}$ under the assumptions that:

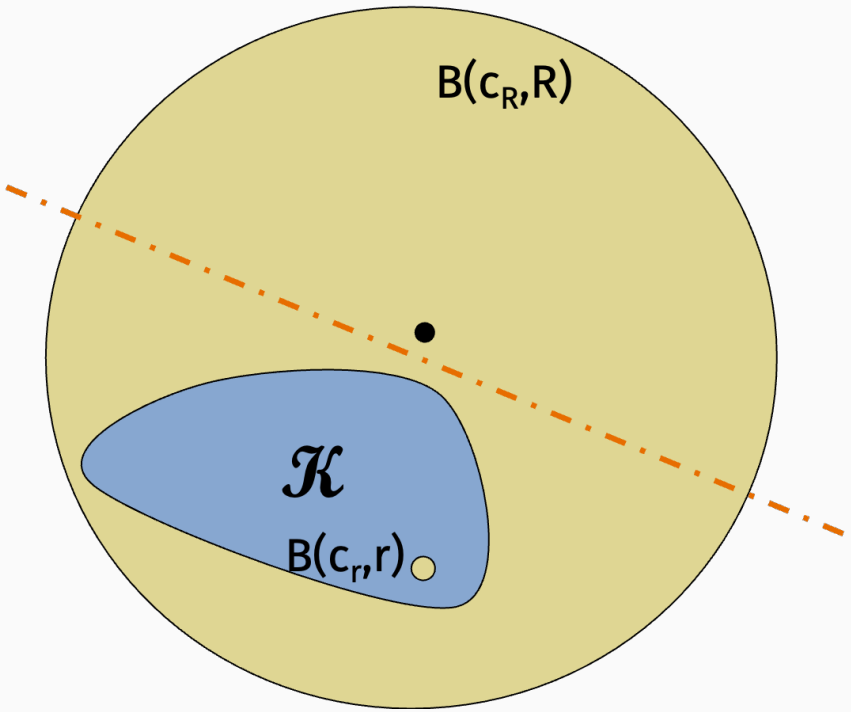
1. $\mathcal{K} \subset B(\mathbf{c}_R, R)$.
2. If non-empty, $\mathcal{K}$ contains $B(\mathbf{c}_r, r)$ for some $r < R$.



ELLIPSOID METHOD SKETCH

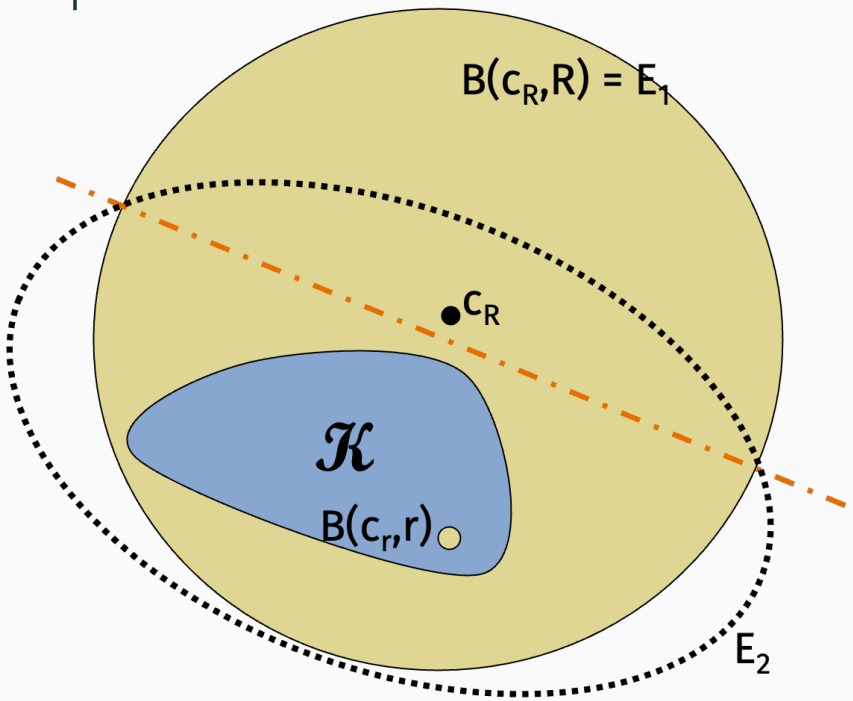
Iterative method similar to center-of-gravity:

1. Check if center $\mathbf{c}_R$ of $B(\mathbf{c}_R, R)$ is in $\mathcal{K}$.
2. If it is, we are done.
3. If not, cut search space in half, using separating hyperplane.



ELLIPSOID METHOD SKETCH

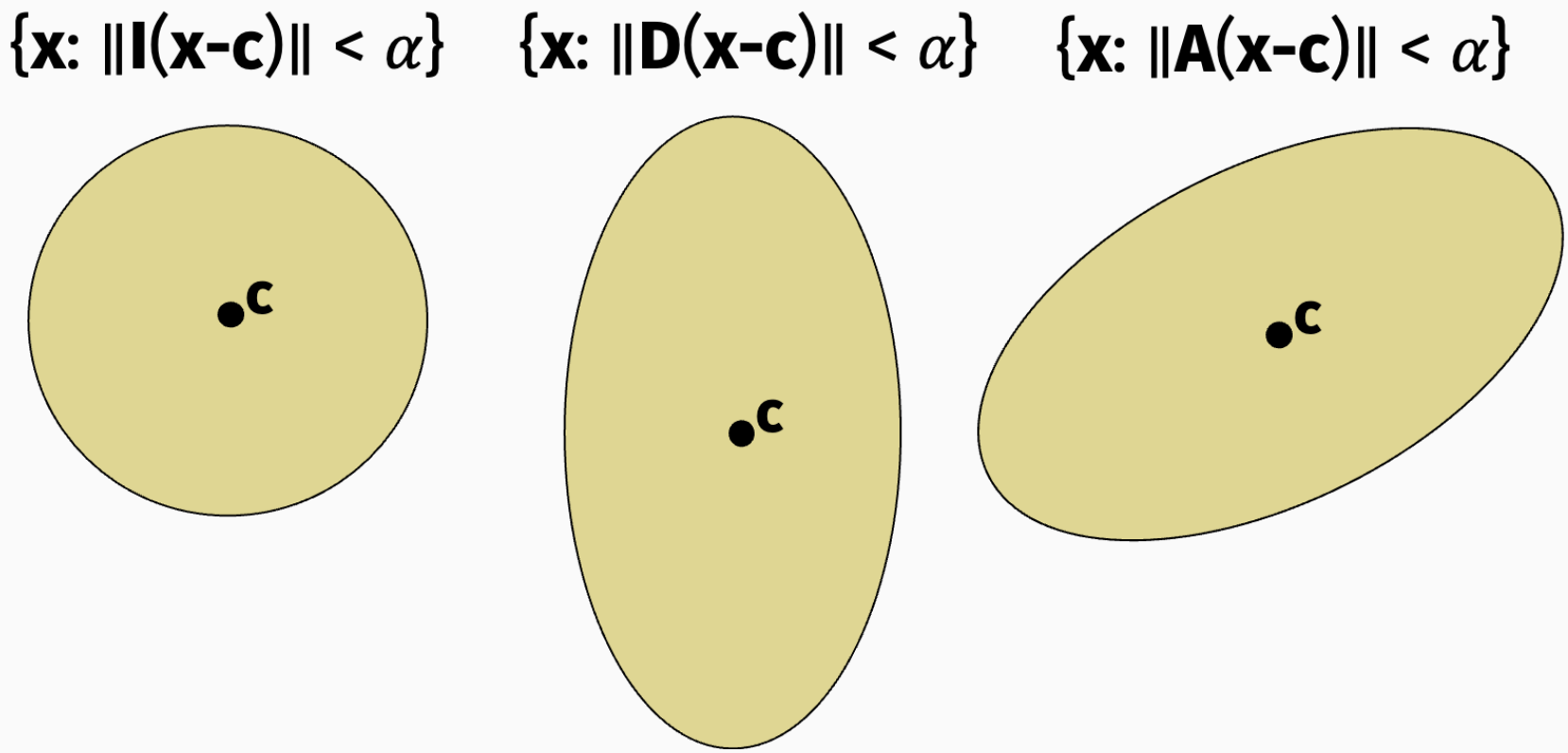
Key insight: Before moving on, approximate new search region by something that we can easily compute the centroid of. Specifically an ellipse!!



Produce a sequence of ellipses that always contain $\mathcal{K}$ and decrease in volume: $B(\mathbf{c}_R, R) = E_1, E_2, \ldots$. Once we get to an ellipse with volume $\leq B(\mathbf{c}_r, r)$, we know that $\mathcal{K}$ must be empty.

ELLIPSE

An ellipse is a convex set of the form: $\{\mathbf{x} : \|\mathbf{A}(\mathbf{x} - \mathbf{c})\|_2^2 \leq \alpha\}$ for some constant c and matrix $\mathbf{A}$. The center-of-mass is $\mathbf{c}$.



Often re-parameterized to say that the ellipse is all $\mathbf{x}$ with $\{\mathbf{x} : (\mathbf{x} - \mathbf{c})^T \mathbf{Q}^{-1} (\mathbf{x} - \mathbf{c}) \leq 1\}$

ELLIPSOID UPDATE

There is a closed form solution for the equation of the smallest ellipse containing a given half-ellipse. I.e. let $\mathbf{E}_i$ have parameters $\mathbf{Q}_i, \mathbf{c}_i$ and consider the half-ellipse:

$$\mathbf{E}_i \cap \{\mathbf{x} : \mathbf{a}_i^T \mathbf{x} \leq \mathbf{a}_i^T \mathbf{c}_i\}.$$

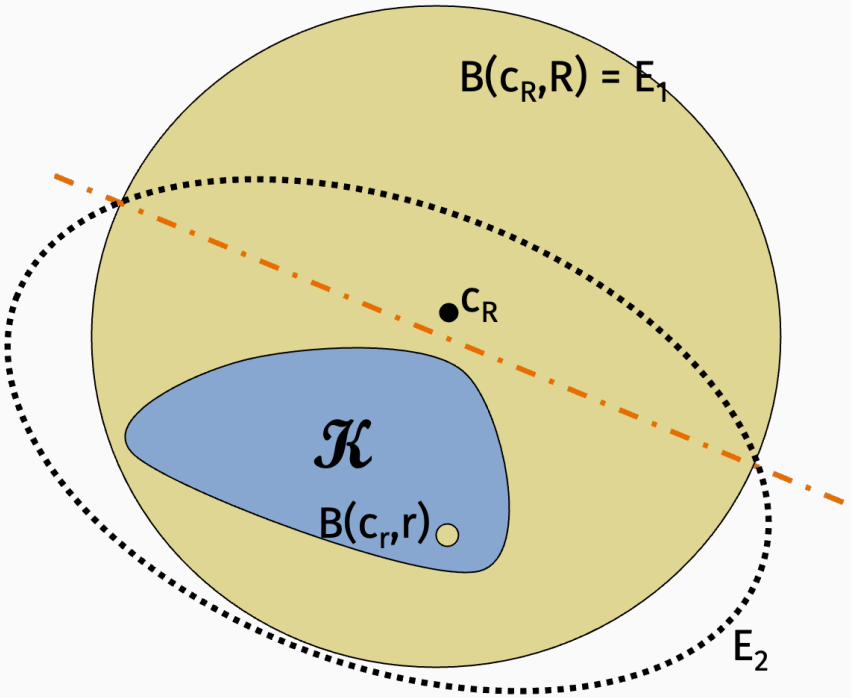
Then $\mathbf{E}_{i+1}$ is the ellipse with parameters:

$$\mathbf{Q}_{i+1} = \frac{d^2}{d^2 - 1}\left(\mathbf{Q}_i - \frac{2}{d+1}\mathbf{h}\mathbf{h}^T\right) \qquad \mathbf{c}_{i+1} = \mathbf{c}_i - \frac{1}{n+1}\mathbf{h},$$

where $\mathbf{h} = \sqrt{\mathbf{a}_i^T \mathbf{Q}_i \mathbf{a}_i} \cdot \mathbf{a}_i$.

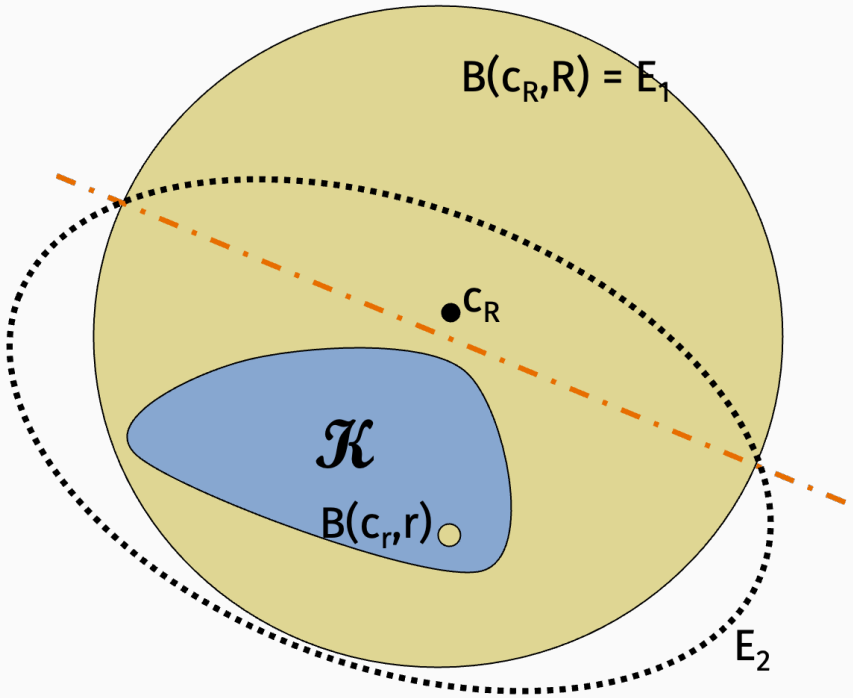
Claim: $\text{vol}(E_{i+1}) \leq (1 - \frac{1}{2d})\,\text{vol}(E_i)$.

Proof: Via reduction to the "isotropic case". I will post a proof on the course website if you are interested.



Not as good as the $(1 - \frac{1}{e})$ constant-factor volume reduction we got from center-of-gravity, but still very good!

Claim: $\mathrm{vol}(E_{i+1}) \leq (1 - \frac{1}{2d})\,\mathrm{vol}(E_i)$



After $O(d)$ iterations, we reduce the volume by a constant.

In total require $O(d^2 \log(R/r))$ iterations to solve the problem.

Theorem (Khachiyan, 1979)

Assume $n = d$. The ellipsoid method solves any linear program with L-bit integer valued constraints in $O(n^4 L)$ time. I.e. linear programming is in (weakly) polynomial time!

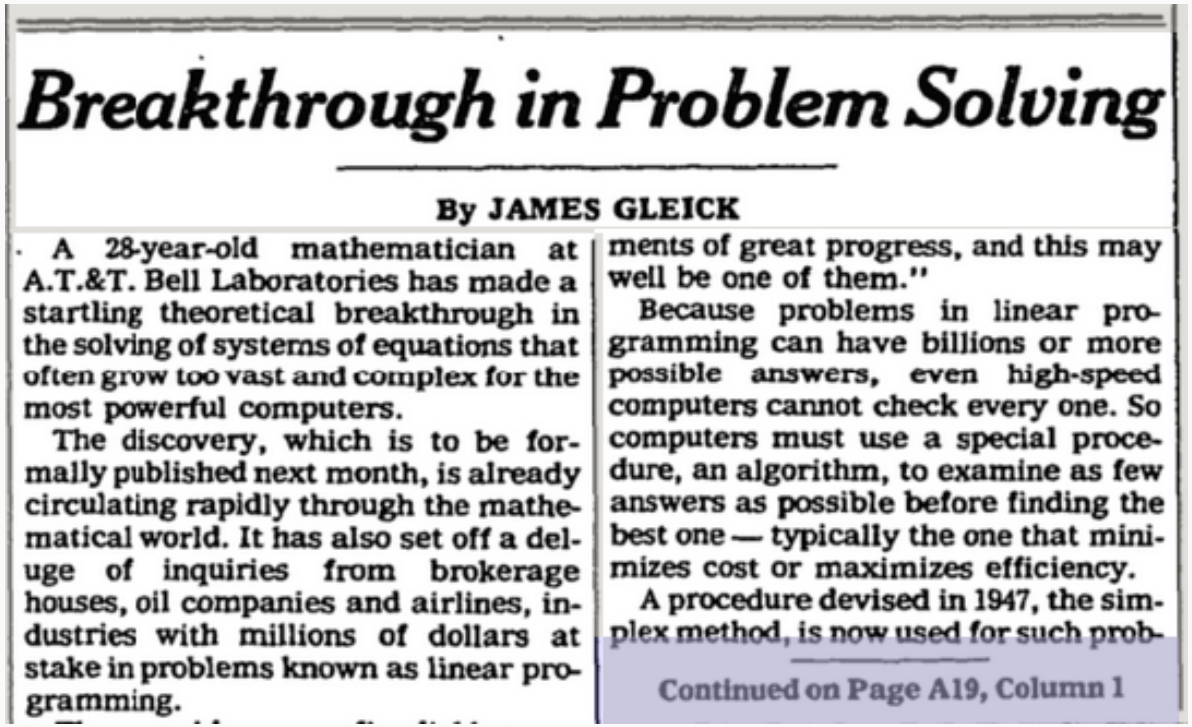
The method works for any convex program.

For LPs, we have an $O(nd)$ time separation oracle, and ellipsoid update take $O(d^2)$ time.

Careful analysis of the binary search method, how to set B_r and B_R, etc. leads to the final runtime bound.

Theorem (Karmarkar, 1984)

Assume $n = d$. The interior point method solves any linear program with L-bit integer valued constraints in $O(n^{3.5}L)$ time.



Breakthrough in Problem Solving

By JAMES GLEICK

A 28-year-old mathematician at A.T.&T. Bell Laboratories has made a startling theoretical breakthrough in the solving of systems of equations that often grow too vast and complex for the most powerful computers.

The discovery, which is to be formally published next month, is already circulating rapidly through the mathematical world. It has also set off a deluge of inquiries from brokerage houses, oil companies and airlines, industries with millions of dollars at stake in problems known as linear programming.

ments of great progress, and this may well be one of them."

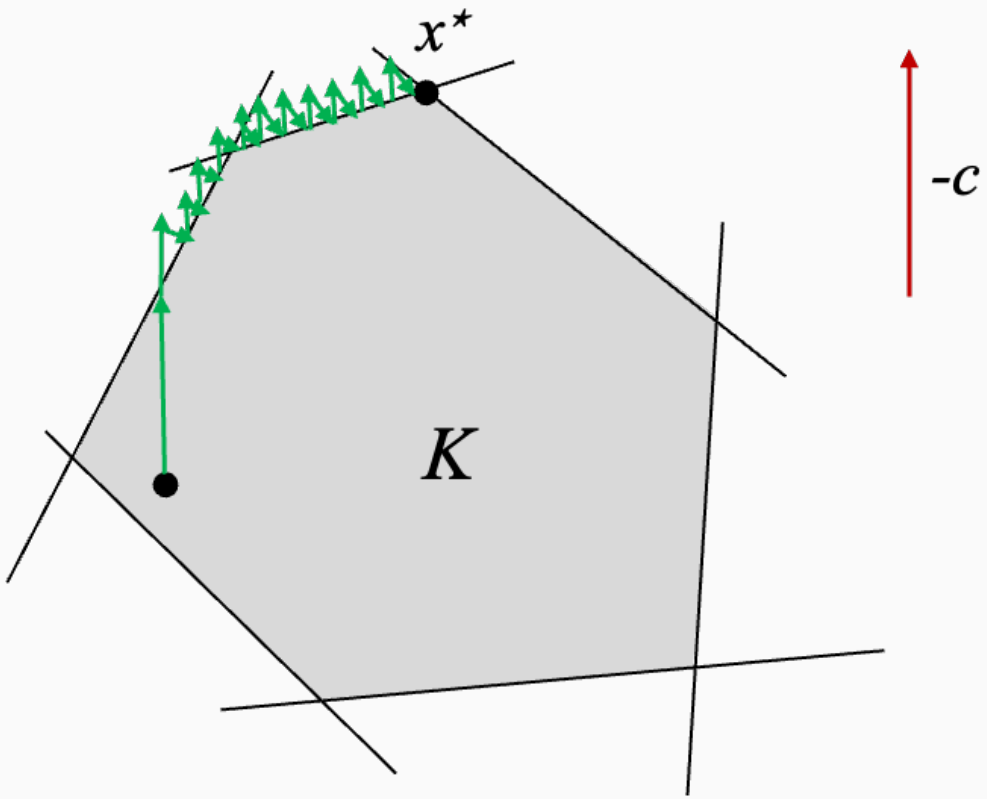
Because problems in linear programming can have billions or more possible answers, even high-speed computers cannot check every one. So computers must use a special procedure, an algorithm, to examine as few answers as possible before finding the best one — typically the one that minimizes cost or maximizes efficiency.

A procedure devised in 1947, the simplex method, is now used for such prob-

Continued on Page A19, Column 1

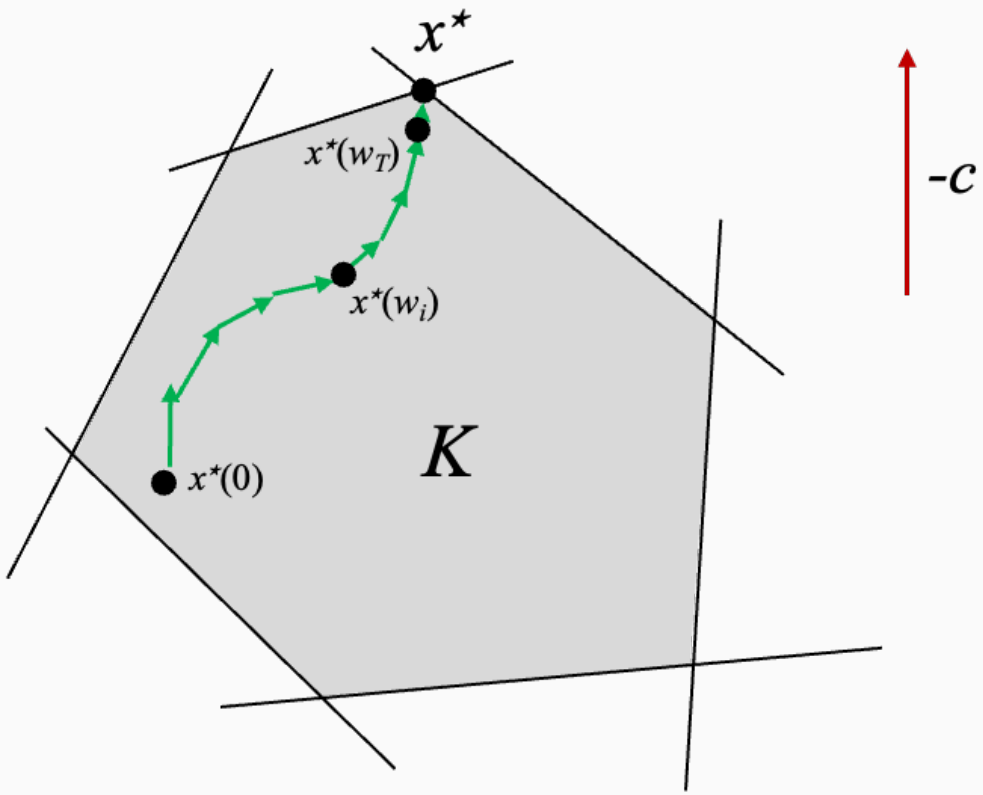
Front page of New York Times, November 19, 1984.

Will post lecture notes on the website (optional reading).



Projected Gradient Descent Optimization Path

Will post lecture notes on the website (optional reading).



Ideal Interior Point Optimization Path

Both results had a huge impact on the theory of optimization, although at the time neither the ellipsoid method or interior point method were faster than a heuristic known at the Simplex Method.

These days, improved interior point methods compete with and often outperform simplex.