CS 270

Combinatorial Algorithms and Data Structures

Lecture Notes

Druv Pai

Contents

1 Continuous Optimization

Notation 1.1
For a sequence $x = (x_n)_{n\in\mathbb{N}}$, the first n elements are denoted by $x^n = (x_i)_{i\in[n]}$.

Let $f : \mathbb{R}^n \to \mathbb{R}$. There are two kinds of optimization problems: *unconstrained optimization*:

$$x_* = \underset{x\in\mathbb{R}^n}{\text{minimize}}\ f(x)$$

and *constrained optimization*: for $X \subseteq \mathbb{R}^n$,

$$x_* = \underset{x\in X}{\text{minimize}}\ f(x).$$

Setting $X = \mathbb{R}^n$ converts the constrained problem to an unconstrained problem.

We will look at algorithms for minimizing these functions, provided certain "well-behavedness" conditions on f.

1.1 Well-Behaved Functions

Definition 1.2 (Convex Set)
A set $K \subseteq \mathbb{R}^n$ is *convex* if for every $\lambda \in [0,1]$ and $x, y \in \mathbb{R}^n$,

$$\lambda x + (1-\lambda)y \in K.$$

Geometrically, this means that the line segment connecting x and y stays wholly within K.

Suppose we are given a function $f : K \to \mathbb{R}$, for $K \subseteq \mathbb{R}^n$ convex.

Definition 1.3 (Convex Function)
The function f is convex if and only if any of the four conditions hold:

(i) (*Epigraph condition.*) Let $f \in C^0(K, \mathbb{R})$. Then the *epigraph* $\operatorname{epi}(f) = \{(x, y) \in K \times \mathbb{R} : y \geq f(x)\}$ is convex.

(ii) (*Zeroth order condition.*) Let $f \in C^0(K, \mathbb{R})$. Let $\lambda \in [0,1]$. Then for every $x, y \in \mathbb{R}^n$,

$$f\left(\lambda x + (1-\lambda)y\right) \leq \lambda f(x) + (1-\lambda) f(y).$$

(iii) (*First order condition.*) Let $f \in C^1(K, \mathbb{R})$ Then for every $x, y \in \mathbb{R}^n$,

$$f(y) - f(x) \geq \langle \nabla f(x), y - x \rangle .$$

(iv) (*Second order condition.*) Let $f \in C^2(K, \mathbb{R})$. Then for every $x \in \mathbb{R}^n$,

$$H_f(x) \succeq 0$$

where $H_f(x)$ is the *Hessian* of f evaluated at x, a matrix whose ij^{th} coordinate is $\frac{\partial^2 f(x)}{\partial x_i \partial x_j}$.

Intuitively, the epigraph condition implies that any line segment with endpoints above the graph stays above the graph.

Remark. If f is convex, any *stationary point* (x_* such that $\nabla f(x_*) = 0$) is also a *global minimum*; the first order condition of **??** gives $f(y) \geq f(x_*)$ for all y. Given a convex function, we can just solve the equation $\nabla f(x) = 0$ to compute the global minima exactly.

There are some more relevant function properties that make analysis easier.

Definition 1.4 (Lipschitz)
The function f is *L-Lipschitz* with respect to the norm $\|\cdot\|$ if and only if any of the two conditions hold:

(i) (*Zeroth order condition.*) Let $f \in C^0(K, \mathbb{R})$. Then for all $x, y \in K$,
$$|f(x) - f(y)| \leq L \|x - y\|.$$

(ii) (*First order condition.*) Let $f \in C^1(K, \mathbb{R})$. Then for all $x \in K$,
$$\|\nabla f(x)\| \leq L.$$

Definition 1.5 (α-Strong Convexity)
Let $\alpha \in \mathbb{R}_+$. Then f is *α-strongly convex* if any of the three conditions hold:

(i) (*Zeroth order condition.*) Let $f \in C^0(K, \mathbb{R})$. Then for all $\lambda \in [0, 1]$ and $x, y \in K$,
$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y) - \frac{\alpha}{2}\lambda(1 - \lambda) \|x - y\|^2.$$

(ii) (*First order condition.*) Let $f \in C^1(K, \mathbb{R})$. Then for all $x, y \in K$,
$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\alpha}{2} \|x - y\|^2.$$

(iii) (*Second order condition.*) Let $f \in C^2(K, \mathbb{R})$. Then for all $x \in K$,
$$H_f(x) \succeq \alpha I.$$

Intuitively, if f is α-strongly convex for large α then it is "not too flat".

Definition 1.6 (β-Smoothness)
Let $\beta \in \mathbb{R}_+$. Then f is *β-smooth* if any of the three conditions hold:

(i) (*Zeroth order condition.*) Let $f \in C^0(K, \mathbb{R})$. Then for all $\lambda \in [0, 1]$ and $x, y \in K$,
$$f(\lambda x + (1 - \lambda)y) \geq \lambda f(x) + (1 - \lambda) f(y) - \frac{\beta}{2}\lambda(1 - \lambda) \|x - y\|^2.$$

(ii) (*First order condition.*) Let $f \in C^1(K, \mathbb{R})$. Then for all $x, y \in K$,
$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\beta}{2} \|x - y\|^2.$$

(iii) (*Second order condition.*) Let $f \in C^2(K, \mathbb{R})$. Then for all $x \in K$,
$$H_f(x) \preceq \beta I.$$

Intuitively, if f is β-smooth for large β then it is "not too curved".

1.2 Simple Gradient Descent

Now we can analyze gradient descent. Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is convex, and we want to solve the unconstrained problem

$$x_* = \operatorname*{argmin}_{x \in \mathbb{R}^n} f(x).$$

Notice that $\mathbb{R}^n$ is closed so a minimizer either exists or is at ∞.

Algorithm 1 Gradient descent.

Input: A convex function $f : \mathbb{R}^n \to \mathbb{R}$ to minimize.
Input: An initial point $x_1 \in \mathbb{R}^n$.
Input: A step size η.
Input: A large stopping number of iterations T.
Output: A guess for the minimizer x_* of f.
  for $t \in [T]$ **do**
    $x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$
  return $\widehat{x} = \dfrac{1}{T} \displaystyle\sum_{t=1}^{T} x_t$

Remark. We return $\widehat{x}$, the average of x^T, because sometimes we overshoot the minimum region of the graph. This is particularly true when $f \notin C^2(\mathbb{R}^n, \mathbb{R})$ as the gradient can change sharply and unboundedly. The average helps us control the last few timesteps to land in the minimum region.

Proposition 1.7
The vector $-\nabla f(x)$ points towards x_*.

Proof. Since $f \in C^1(\mathbb{R}^n, \mathbb{R})$, we may use the first order condition with the points x and x_*. We obtain

$$f(x_*) \geq f(x) + \langle \nabla f(x), x_* - x \rangle \implies \langle -\nabla f(x), x_* - x \rangle \geq f(x) - f(x_*) \geq 0$$

which implies

$$\langle -\nabla f(x), x_* - x \rangle \geq 0.$$

This implies that $-\nabla f(x)$ points towards x_*. □

Theorem 1.8
Let $f \in C^1(\mathbb{R}^n, \mathbb{R})$ be convex and L-Lipschitz. Let $R = \|x_0 - x_*\|_2$ and $\eta = \frac{R}{L\sqrt{T}}$. Then after T steps of gradient descent,

$$\frac{1}{T} \sum_{t=1}^{T} f(x_t) \leq f(x_*) + \frac{1}{2} \left(\eta L^2 + \frac{R^2}{\eta T} \right).$$

Proof. Since f is convex,

$$\begin{aligned} f(x_t) - f(x_*) &\leq \langle \nabla f(x_t), x_t - x_* \rangle \\ &\leq \left\langle \frac{x_t - x_{t+1}}{\eta}, x_t - x_* \right\rangle \end{aligned}$$

$$\begin{aligned}
&\leq \frac{1}{2\eta}\left(\|x_t - x_{t+1}\|^2 + \|x_t - x_*\|^2 - \|x_{t+1} - x_t\|^2\right) \\
\sum_{t=1}^{T}(f(x_t) - f(x_*)) &\leq \frac{1}{2\eta}\sum_{t=1}^{T}\left(\|x_t - x_{t+1}\|^2 + \|x_t - x_*\|^2 - \|x_{t+1} - x_t\|^2\right) \\
&= \frac{1}{2\eta}\sum_{t=1}^{T}\|x_{t+1} - x_t\|^2 + \frac{1}{2\eta}\|x_1 - x_*\|^2 - \frac{1}{2\eta}\|x_t - x_*\|^2 \\
&\leq \frac{1}{2\eta}\sum_{t=1}^{T}\|x_{t+1} - x_t\|^2 + \frac{1}{2\eta}\|x_1 - x_*\|^2 \\
&\leq \frac{1}{2\eta}\sum_{t=1}^{T}(\eta L)^2 + \frac{1}{2\eta}\|x_1 - x_*\|^2 \\
&\leq \frac{\eta L^2 T}{2} + \frac{R^2}{2\eta} \\
\frac{1}{T}\sum_{t=1}^{T} f(x_t) &\leq f(x_*) + \frac{\eta L^2}{2} + \frac{R^2}{2\eta T}.
\end{aligned}$$

And the proof is complete. □

Proposition 1.9

Let $f \in C^1(\mathbb{R}^n, \mathbb{R})$ be convex and L-Lipschitz. Then after $T = \frac{L^2R^2}{\varepsilon^2}$ steps of gradient descent,

$$f(\widehat{x}) \leq f(x_*) + \varepsilon.$$

Proof. By convexity of f,

$$f(\widehat{x}) = f\left(\frac{1}{T}\sum_{t=1}^{T} x_t\right) \leq \frac{1}{T}\sum_{t=1}^{T} f(x_t).$$

By **??**,

$$\frac{1}{T}\sum_{t=1}^{T} f(x_t) \leq f(x_*) + \frac{1}{2}\left(\eta L^2 + \frac{R^2}{\eta T}\right).$$

When $\eta = \frac{R}{L\sqrt{T}}$, giving

$$f(\widehat{x}) \leq f(x_*) + \frac{RL}{\sqrt{T}}.$$

Finally, we use the definition of $T = \frac{L^2R^2}{\varepsilon^2}$ to obtain

$$f(\widehat{x}) \leq f(x_*) + \varepsilon.$$

□

1.3 Well-Behaved Gradient Descent

Definition 1.10 (Condition Number)

Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is α-strongly convex and β-smooth. Then we define the *condition number* of f to be

$\kappa = \frac{\beta}{\alpha}$.

Remark. The condition number is more well known as the ratio of the largest eigenvalue to the smallest eigenvalue of the Hessian, that is,

$$\kappa = \frac{\alpha}{\beta} = \max_{x \in \mathbb{R}^n} \frac{\lambda_{\max}(H_f(x))}{\lambda_{\min}(H_f(x))}.$$

This is supported by considering α as some measure of the "flatness" of f, and β as a measure of the "steepness" of f.

If κ is large then the problem is *poorly conditioned* and will take a long time to converge. If the condition number is large then *preconditioning algorithms* can scale the problem so that the condition number of the new problem is small, yet has the same optima.

Theorem 1.11

Let $f : \mathbb{R}^n \to \mathbb{R}$ have condition number κ. Then if we run (unconstrained) gradient descent for T timesteps with $\eta = \frac{1}{\beta}$,

$$\frac{1}{T} \sum_{t=1}^{T} f(x_t) - f(x_*) \le \frac{\beta R^2}{2} \exp\left(-\frac{t}{\kappa}\right).$$

Proof. For β-smooth f, we can get

$$f(x_{t+1}) \le f(x_t) - \eta \left\|\nabla f(x_t)\right\|^2 + \eta^2 \frac{\beta}{2} \left\|\nabla f(x_t)\right\|^2.$$

The right-hand side is minimized by setting $\eta = \frac{1}{\beta}$, whence we get

$$f(x_{t+1}) - f(x_t) \le -\frac{1}{2\beta} \left\|\nabla f(x_t)\right\|^2.$$

For α-strongly convex f, we get

$$\begin{aligned} f(x_t) - f(x_*) &\le \langle \nabla f(x_t), x_t - x_* \rangle - \frac{\alpha}{2} \left\|x_t - x_*\right\|^2 \\ &\le \left\|\nabla f(x_t)\right\| \left\|x_t - x_*\right\| - \frac{\alpha}{2} \left\|x_t - x_*\right\|^2 \\ &\le \frac{1}{2\alpha} \left\|\nabla f(x_t)\right\|^2 \end{aligned}$$

where we use that the right hand side is maximized when $\left\|x_t - x_*\right\| = \frac{\left\|\nabla f(x_t)\right\|}{\alpha}$. Combining these ideas, we have

$$f(x_{t+1}) - f(x_t) \le -\frac{\alpha}{\beta} \left(f(x_t) - f(x_*)\right),$$

letting $\Delta_t = f(x_t) - f(x_*)$ and rearranging, we get

$$\Delta_{t+1} \le \left(1 - \frac{\alpha}{\beta}\right) \Delta_t \le \left(1 - \frac{1}{\kappa}\right)^t \Delta_1 \le \exp\left(-\frac{t}{\kappa}\right) \Delta_1.$$

We can control the value of Δ_1 by using $x = x_*$, $y = x_1$; since $\nabla f(x_*) = 0$, we get

$$\Delta_1 = f(x_1) - f(x_*) \le \frac{\beta}{2} \left\|x_1 - x_*\right\|^2.$$

□

1.4 Projected Gradient Descent

We will now turn to constrained optimization. Suppose $K \subseteq \mathbb{R}^n$ is convex. The problem to solve obeys the form:

$$x_* = \operatorname*{argmin}_{x \in K} f(x).$$

We want to do gradient descent but stay inside the set K. For this we use *projected gradient descent*, which takes the form

Algorithm 2 Projected gradient descent.

Input: A convex function $f : \mathbb{R}^n \to \mathbb{R}$ to minimize.
Input: A convex set $K \subseteq \mathbb{R}^n$ to minimize over.
Input: An initial point $x_1 \in K$.
Input: A step size η.
Input: A large stopping number of iterations T.
Output: A guess for the minimizer x_* of f over K.

for $t \in [T]$ **do**
  $y_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$
  $x_{t+1} \leftarrow \Pi_K(y_{t+1})$
return $\widehat{x} = \dfrac{1}{T}\displaystyle\sum_{t=1}^{T} x_t$

To be precise,

$$\Pi_K(x) = \operatorname*{argmin}_{y \in K} \|x - y\|_2 .$$

Theorem 1.12
Let K be closed and convex. Let $f \in C^1(K, \mathbb{R})$ be convex and L-Lipschitz. Let $R = \|x_0 - x_*\|_2$ and $\eta = \frac{R}{L\sqrt{T}}$. Then after T steps of gradient descent,

$$\frac{1}{T}\sum_{t=1}^{T} f(x_t) \le f(x_*) + \frac{1}{2}\left(\eta L^2 + \frac{R^2}{\eta T}\right).$$

Proof. Since f is convex,

$$\begin{aligned}
f(x_t) - f(x_*) &\le \langle \nabla f(x_t), x_t - x_* \rangle \\
&\le \left\langle \frac{x_t - y_{t+1}}{\eta}, x_t - x_* \right\rangle \\
&\le \frac{1}{2\eta}\left(\|x_t - y_{t+1}\|^2 + \|x_t - x_*\|^2 - \|y_{t+1} - x_t\|^2\right) \\
&\le \frac{1}{2\eta}\left(\|x_t - y_{t+1}\|^2 + \|x_t - x_*\|^2 - \|x_{t+1} - x_t\|^2\right) \\
\sum_{t=1}^{T}(f(x_t) - f(x_*)) &\le \frac{1}{2\eta}\sum_{t=1}^{T}\left(\|x_t - y_{t+1}\|^2 + \|x_t - x_*\|^2 - \|x_{t+1} - x_t\|^2\right) \\
&= \frac{1}{2\eta}\sum_{t=1}^{T}\|y_{t+1} - x_t\|^2 + \frac{1}{2\eta}\|x_1 - x_*\|^2 - \frac{1}{2\eta}\|x_t - x_*\|^2
\end{aligned}$$

$$\begin{aligned}
&\leq \frac{1}{2\eta}\sum_{t=1}^{T}\|y_{t+1}-x_t\|^2+\frac{1}{2\eta}\|x_1-x_*\|^2\\
&\leq \frac{1}{2\eta}\sum_{t=1}^{T}\|y_{t+1}-x_t\|^2+\frac{1}{2\eta}\|x_0-x_*\|^2\\
&\leq \frac{1}{2\eta}\sum_{t=1}^{T}(\eta L)^2+\frac{1}{2\eta}\|x_0-x_*\|^2\\
&\leq \frac{\eta L^2 T}{2}+\frac{R^2}{2\eta}\\
\frac{1}{T}\sum_{t=1}^{T}f(x_t)&\leq f(x_*)+\frac{\eta L^2}{2}+\frac{R^2}{2\eta T}.
\end{aligned}$$

And the proof is complete. Here we use $\|\Pi_{\mathcal{K}}(x)-z\|\leq\|x-z\|$ for $z\in\mathcal{K}$. □

Proposition 1.13
Let $f\in C^1(\mathbb{R}^n,\mathbb{R})$ be convex and L-Lipschitz. Then after $T=\frac{L^2R^2}{\varepsilon^2}$ steps of gradient descent,

$$f(\widehat{x})\leq f(x_*)+\varepsilon.$$

Proof. By convexity of f,

$$f(\widehat{x})=f\left(\frac{1}{T}\sum_{t=1}^{T}x_t\right)\leq\frac{1}{T}\sum_{t=1}^{T}f(x_t).$$

By **??**,

$$\frac{1}{T}\sum_{t=1}^{T}f(x_t)\leq f(x_*)+\frac{1}{2}\left(\eta L^2+\frac{R^2}{\eta T}\right).$$

When $\eta=\frac{R}{L\sqrt{T}}$, giving

$$f(\widehat{x})\leq f(x_*)+\frac{RL}{\sqrt{T}}.$$

Finally, we use the definition of $T=\frac{L^2R^2}{\varepsilon^2}$ to obtain

$$f(\widehat{x})\leq f(x_*)+\varepsilon.$$

□

Remark 1.14. For all types of gradient descent that have been discussed, the number of iterations T is not dependent on the dimension. Thus it's independent of the space it's in, as long as the space is nice enough.

Remark 1.15. The error after T timesteps is $O\left(\frac{RL}{\sqrt{T}}\right)$. This is optimal for Lipschitz functions.

Remark 1.16. The performance is pseudopolynomial in the error ε.

Remark 1.17. Computing the gradient can take some amount of time; we use a gradient oracle, although it's possible to do (projected) subgradient descent and get the same analysis. An oracle for the gradient could be to sample $u\in S^{n-1}$ and return

$$\frac{f(x+\delta u)}{\delta}u$$

for some tiny $\delta > 0$. We have

$$\mathop{\mathsf{E}}_{u \sim \mathrm{Uni}\left(S^{n-1}\right)}\left[\frac{f(x+\delta u)}{\delta} u\right] = \nabla f(x).$$

Another oracle is stochastic gradient descent, which gives an estimator $\theta(x)$ such that $\mathsf{E}[\theta(x)] = \nabla f(x)$ and it can be computed very quickly.

1.5 Mirror Descent

The gradient descent algorithm of the previous section is general and powerful: it allows us to (approximately) minimize convex functions over convex bodies. Moreover, it also works in the model of online convex optimization, where the convex function can vary over time, and we want to find a low-regret strategy—one which performs well against every fixed point x_*.

This power and broad applicability means the algorithm is not always the best for specific classes of functions and bodies. This suggests asking: can we somehow change gradient descent to adapt to the "geometry" of the problem?

The mirror descent framework of this section allows us to do precisely this. There are many different (and essentially equivalent) ways to explain this framework, each with its positives. We present two of them here: the *proximal point view*, and the *mirror map view*.

1.5.1 Proximal Point View

Here is a different way to arrive at the gradient descent algorithm.

Algorithm 3 Proximal gradient descent.

Input: A convex function $f : \mathbb{R}^n \to \mathbb{R}$ to minimize.
Input: An initial point $x_1 \in K$.
Input: A step size η.
Input: A large stopping number of iterations T.
Output: A guess for the minimizer x_* of f.

for $t \in [T]$ **do**

$$x_{t+1} \leftarrow \operatorname*{argmin}_x \left\{\eta \left\langle \nabla f(x_t), x \right\rangle + \frac{1}{2} \left\| x - x_t \right\|^2\right\}$$

return $\widehat{x} = \dfrac{1}{T} \displaystyle\sum_{t=1}^{T} x_t$

In this case,

$$\nabla \left(\eta \left\langle \nabla f(x_t), x \right\rangle + \frac{1}{2} \left\| x - x_t \right\|^2\right) = 0 \implies x = x_t - \eta \nabla f(x_t).$$

which recovers the gradient descent algorithm. Moreover, the intuition for this algorithm also makes sense: if we want to minimize the function $f(x)$, we could try to minimize its linear approximation $f(x_t) + \langle \nabla f(x_t), x - x_t \rangle$ instead. But we should be careful not to "overfit": this linear approximation is good only close to the point x_t, so we could add in a penalty function (a "regularizer") to prevent us from straying from the point x_t. This means we should minimize

$$x_{t+1} \leftarrow \operatorname*{argmin}_x \left\{f(x_t) + \langle \nabla f(x_t), x - x_t \rangle + \frac{1}{2} \left\| x - x_t \right\|^2\right\} = \operatorname*{argmin}_x \left\{\langle \nabla f(x_t), x \rangle + \frac{1}{2} \left\| x - x_t \right\|^2\right\}.$$

If we have a constrained problem, we can change the update step to:

$$x_{t+1} \leftarrow \operatorname*{argmin}_{x \in \mathcal{K}} \left\{ \eta \langle \nabla f(x_t), x \rangle + \frac{1}{2} \|x - x_t\|^2 \right\}.$$

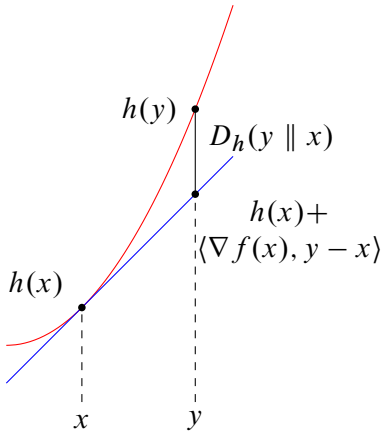
The optimality conditions are a bit more complicated now, but they again can show this algorithm is equivalent to projected gradient descent from the previous section. Given this perspective, we can now replace the squared Euclidean norm by other distances to get different algorithms. A particularly useful class of distance functions are Bregman divergences, which we now define and use.

1.5.2 Bregman Divergences

Given a *strictly* convex function h, we can define a distance based on how the function differs from its linear approximation.

Definition 1.18 (Bregman Divergence)
The *Bregman divergence* from x to y with respect to h is

$$D_h(y \parallel x) = h(y) - h(x) - \langle \nabla h(x), y - x \rangle.$$



Example 1.19
For the function $h(x) = \|x\|^2$, the associated Bregman divergence is

$$D_h(y \parallel x) = \|y - x\|^2,$$

the squared distance.

Example 1.20
For the negative entropy function

$$h(x) = \sum_{i=1}^{n} (x_i \log(x_i) - x_i) \implies D_h(y \parallel x) = \sum_{i=1}^{n} \left(y_i \log\left(\frac{y_i}{x_i}\right) - y_i + x_i \right) = D_{\mathrm{KL}}(y \parallel x)$$

supposing that $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i = 1$ in the last equality (so that x and y are probability distributions).

1.5.3 Changing the Distance

Since the distance function $\frac{1}{2}\|x-y\|^2$ is a Bregman divergence, what if we replace it by a generic Bregman divergence: what algorithm do we get in this case? Again, let us first consider the unconstrained problem, with the update

$$x_{t+1} \leftarrow \operatorname*{argmin}_x \{\eta \langle \nabla f(x_t), x\rangle + D_h(x \parallel x_t)\}.$$

Again, setting

$$\nabla\{\eta \langle \nabla f(x_t), x\rangle + D_h(x \parallel x_t)\} = 0 \implies \eta \nabla f(x_t) + \nabla h(x) - \nabla h(x_t) = 0,$$

or rephrasing,

$$\nabla h(x) = \nabla h(x_t) - \eta \nabla f(x_t) \implies x = (\nabla h)^{-1}(\nabla h(x_t) - \eta \nabla f(x_t)).$$

Note that, if h is differentiable, since f is strictly convex ∇h is invertible. This suffices when $K = \mathbb{R}^n$, and for $K \subset \mathbb{R}^n$ convex we define $\Pi_K^h(x) = \operatorname{argmin}_y D_h(y \parallel x)$ as a sort of "generalized projection" onto K. From here we also use the variable $R = D_h(x_* \parallel x_1)$ instead of $R = \|x_* - x_1\|_2$.

For our two examples:

1. When $h(x) = \frac{1}{2}\|x\|^2$, the gradient is $\nabla h(x) = x$. The update rule is
$$x_{t+1} = x_t - \eta \nabla f(x_t).$$
2. When $h(x) = \sum_{i=1}^{n}(x_i \log(x_i) - x_i)$, then $\nabla h(x) = (\log(x_1), \dots, \log(x_n))$, so
$$x_{t+1,i} = e^{\log(x_{t,i}) - \eta \nabla f(x_t)} = x_{t,i} e^{-\eta \nabla f(x_t)}.$$
Now if $f(x) = \langle \ell_t, x\rangle$ (is time-varying), its gradient is the vector ℓ_t, so we get the multiplicative weights update rule.

Put formally, the general proximal gradient descent algorithm is given by:

Algorithm 4 General proximal gradient descent.

Input: A convex function $f : \mathbb{R}^n \to \mathbb{R}$ to minimize.
Input: A convex set $K \subseteq \mathbb{R}^n$ to minimize over.
Input: A strictly convex function $h \in C^1(\mathbb{R}^n, \mathbb{R})$.
Input: An initial point $x_1 \in K$.
Input: A step size η.
Input: A large stopping number of iterations T.
Output: A guess for the minimizer x_* of f over K.
for $t \in [T]$ **do**
$x_{t+1} \leftarrow \Pi_K^h\left((\nabla h)^{-1}(\nabla h(x_t) - \nabla f(x_t))\right)$
return $\widehat{x} = \frac{1}{T}\sum_{t=1}^{T} x_t$

To summarize: this algorithm that tries to minimize the linear approximation of the function, regularized by a Bregman distance D_h, gives us vanilla gradient descent for one choice of h (which is good for quadratic-like functions

over Euclidean space), and multiplicative weights for another choice of h (which is good for linear functions over the space of probability distributions). Indeed, depending on how we choose the function, we can get different properties from this algorithm – this is the mirror descent framework.

1.6 Mirror Map

In gradient descent, at each step we set

$$x_{t+1} = x_t - \eta \nabla f(x_t).$$

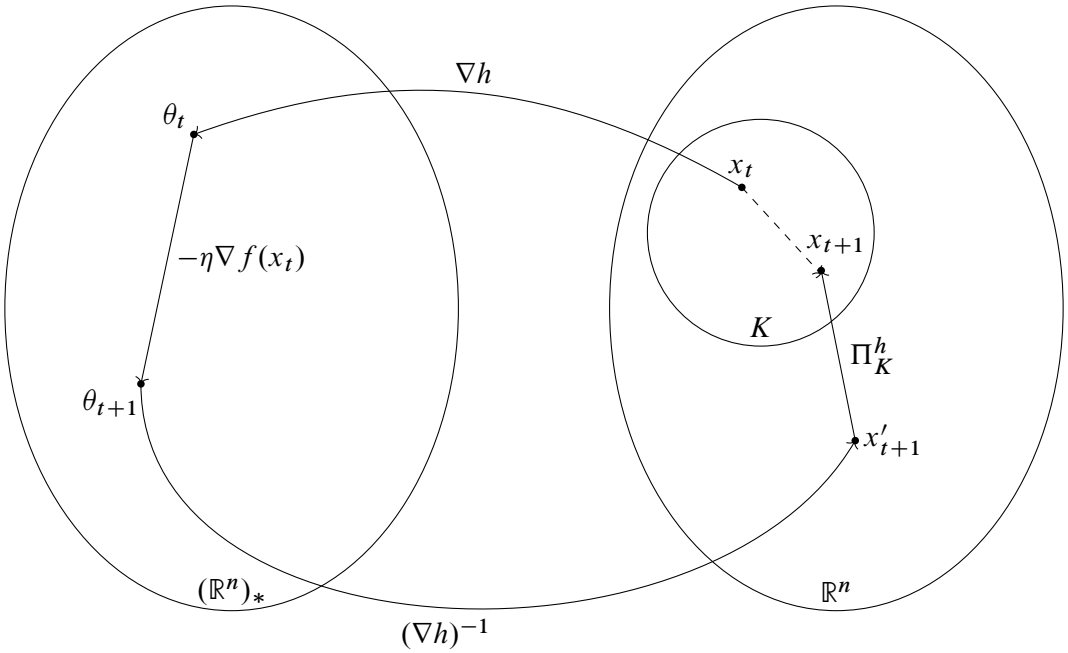
However, the gradient is defined as a linear functional on $\mathbb{R}^n$:

$$\nabla f(x) \colon \mathbb{R}^n \to \mathbb{R} \quad \text{is given by} \quad y \mapsto \langle \nabla f(x), y \rangle .$$

Hence $\nabla f(x_t) \in (\mathbb{R}^n)_*$. The fact that we represent this functional (represented as a covector) is a matter of convenience, and we should exercise care.

In the vanilla gradient descent method, we were working in $\mathbb{R}^n$ endowed with ℓ^2-norm, and this normed space is self-dual, so it is perhaps reasonable to combine points in the primal space (the iterates x_t of our algorithm) with objects in the dual space (the gradients $\nabla f(x_t)$). But when working with other normed spaces, adding a covector $\nabla f(x_t)$ to a vector x_t might not be the right thing to do. Instead, here is a proposed algorithm:

1. We map our current point x_t to a point θ_t in the dual space using a *mirror map*.
2. Next, we take the gradient step
$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla f(x_t).$$
3. We map θ_{t+1} back to a point in the primal space x'_{t+1} using the inverse of the mirror map.
4. If we are in the constrained case, this point x'_{t+1} might not be in the convex feasible region K, so we need to project x'_{t+1} back into $x_{t+1} \in K$.



The name of the process comes from thinking of the dual space as being a *mirror image* of the primal space. But how do we choose these mirror maps? Again, this comes down to understanding the geometry of the problem, the kinds of functions and the set K we care about, and the kinds of guarantees we want.

1.6.1 Defining the Mirror Map

To define a mirror map, we first fix a norm $\|\cdot\|$, and then choose a differentiable convex function $h\colon \mathbb{R}^N \to \mathbb{R}$ that is α-strongly convex with respect to this norm. Then

$$h(y) \geq h(x) + \langle \nabla h(x), y - x \rangle + \frac{\alpha}{2} \|y - x\|^2.$$

Two familiar examples are:

1. $h(x) = \frac{1}{2}\|x\|_2^2$ is 1-strongly convex with respect to $\|\cdot\|_2$, and
2. $h(x) = \sum_{i=1}^{n} x_i (\log(x_i) - 1)$ is $-\log(2)$-strongly convex with respect to $\|\cdot\|_1$.

Having fixed $\|\cdot\|$ and h, the *mirror map* is

$$(\nabla h)\colon \mathbb{R}^n \to \mathbb{R}^n.$$

Since h is differentiable and strongly convex, we can define the inverse map as well. This defines the mappings that we use:

$$\theta_t = \nabla h(x_t) \quad \text{and} \quad x'_{t+1} = (\nabla h)^{-1}(\theta_{t+1}).$$

Let's formally define the algorithm:

Algorithm 5 Mirror mapping descent.

Input: A convex function $f\colon \mathbb{R}^n \to \mathbb{R}$ to minimize.
Input: A convex set $K \subseteq \mathbb{R}^n$ to minimize over.
Input: A strongly convex function $h \in C^1(\mathbb{R}^n, \mathbb{R})$.
Input: An initial point $x_1 \in K$.
Input: A step size η.
Input: A large stopping number of iterations T.
Output: A guess for the minimizer x_* of f over K.

for $t \in [T]$ **do**
  $x_{t+1} \leftarrow \Pi_K^h\left((\nabla h)^{-1}(\nabla h(x_t) - \nabla f(x_t))\right)$
return $\widehat{x} = \frac{1}{T}\sum_{t=1}^{T} x_t$

Theorem 1.21
Let $f \in C^1(K, \mathbb{R})$ be convex and L-Lipschitz, and let $h\colon C^1(\mathbb{R}^n, \mathbb{R})$ be α-strongly convex such that ∇h is invertible. Then mirror descent with $\eta = \frac{R}{L}\sqrt{\frac{2\alpha}{T}}$ satisfies

$$\frac{1}{T}\sum_{t=1}^{T} f(x_t) - f(x_*) \leq 2RL\sqrt{\frac{2}{\alpha T}}.$$

Proof. By convexity of f,

$$\begin{aligned}
f(x_t) - f(x_*) &\le \langle \nabla f(x_t), x_t - x_* \rangle \\
&= \frac{1}{\eta} \left\langle \nabla h(x_t) - \nabla h(x'_{t+1}), x_t - x_* \right\rangle \\
&= \frac{1}{\eta} \left(D_h(x_* \parallel x_t) + D_h(x_t, x'_{t+1}) - D_h(x_* \parallel x'_{t+1}) \right) \\
&\le \frac{1}{\eta} \left(D_h(x_* \parallel x_t) + D_h(x_t \parallel x'_{t+1}) - D_h(x_* \parallel x'_{t+1}) - D_h(x_{t+1} \parallel x'_{t+1}) \right) \\
\sum_{t=1}^{T} (f(x_t) - f(x_*)) &\le \sum_{t=1}^{T} \frac{1}{\eta} \left(D_h(x_* \parallel x_t) + D_h(x_t \parallel x'_{t+1}) - D_h(x_* \parallel x'_{t+1}) - D_h(x_{t+1} \parallel x'_{t+1}) \right) \\
&\le \frac{1}{\eta} \left(D_h(x_* \parallel x_1) - D_h(x_* \parallel x_T) \right) + \sum_{t=1}^{T} \left(D_h(x_t \parallel x'_{t+1}) - D_h(x_{t+1} \parallel x'_{t+1}) \right) \\
&\le \frac{1}{\eta} \left(D_h(x_* \parallel x_1) - D_h(x_* \parallel x_T) \right) + \sum_{t=1}^{T} \frac{(\eta L)^2}{2\alpha} \\
&\le \frac{D_h(x_* \parallel x_1)}{\eta} + \frac{\eta^2 L^2 T}{2\alpha} \\
\frac{1}{T} \sum_{t=1}^{T} f(x_t) - f(x_*) &\le \frac{D_h(x_* \parallel x_1)}{\eta} + \frac{\eta^2 L^2}{2\alpha} \\
&= \frac{R}{\eta} + \frac{\eta^2 L^2}{2\alpha}
\end{aligned}$$

which shows the conclusion. □

1.7 Online Gradient Descent

Online descent is when the objective function f turns into a series of objective functions $(f_t)_{t \in \mathbb{N}}$, where each function $f_t : \mathbb{R}^n \to \mathbb{R}$ is convex. In fact, none of our algorithms or bounds depend on f being constant; each bound from the previous setting applies for this online setting also, in the sense that

$$\sum_{t=1}^{T} (f_t(x_t) - f_t(x_*)) \le \text{some bound in } R, L, T, \alpha, \beta, \text{ etc.}$$

This may seem like a paradox; for example, an adversarial problem could set f_t maximally badly for x_t and thereby incur large cost. But if this is the case, then since f_t is convex, $f_t(x_*)$ is large also. Thus it's impossible to optimize further in the timestep t.

1.8 Application: Linear Programs

Let $x \in [-B, B]^n$ and for $i \in [m]$ we have $\ell_i(x) = \langle w_i, x \rangle - b_i$. We want to find x s.t. $\ell_i(x) < 0$ for all $i \in [m]$. The algorithm has the general form:

- (Generator.) Given x_t and f_t, we use online convex optimization to get an update x_{t+1}.

- (Adversary.) Given an update x_{t+1}, we find a constraint ℓ_i s.t. $\ell_i(x_{t+1}) \geq \varepsilon$ and return $f_{t+1} = \ell_i$.

If the adversary fails, there is no violated constraint, and we're done.

Suppose there is $x_* \in [-B, B]^n$ that satisfies all the constraints. Then

$$\sum_{t=1}^{T} f_t(x_*) = \sum_{t=1}^{T} \ell_{i_t}(x_*) < 0$$

where i_t signifies the index of the constraint picked at time t. One the other hand, if the adversary runs for T steps,

$$\sum_{t=1}^{T} f_t(x_t) \geq \sum_{t=1}^{T} \varepsilon = \varepsilon T.$$

Since the average regret is ε, the generator will eventually terminate and find x_* within ε for every constraint. The runtime for this algorithm is $\text{poly}(\varepsilon^{-1})$, which is pseudopolynomial.

1.9 Centroid Algorithm

The centroid algorithm is the first algorithm which gives $\text{poly}(\log(\varepsilon^{-1}))$ for ε-accurate optima. In particular

Theorem 1.22 (Linear Programming in Polynomial Time)
Linear programming is in P. In particular solving the LP

$$\underset{x \in K}{\text{minimize}}\, c^* x$$

where $K = \{x \in \mathbb{R}^n : Ax \leq b\}$, is solvable in time $\text{poly}(m, n, \max_i \text{bits}(a_i), \max_i \text{bits}(b_i), \text{bits}(c))$, where $A \in \mathbb{R}^{m \times n}$.

Let μ be a measure on $\mathbb{R}^n$. (We normally take the Lebesgue measure). Then we define the *centroid* of a set $A \in \mathcal{B}_{\mathbb{R}^n}$ as

$$\text{centroid}(A) = \frac{1}{\mu(A)} \int_A x \mathrm{d}\mu(x).$$

Algorithm 6 Centroid algorithm.

Input: A convex function $f \in C^1(K, \mathbb{R})$ to minimize.
Input: A convex set $K \subseteq \mathbb{R}^n$ to minimize over.
Input: A guess $\widehat{x}$ for the minimizer $x_* = \underset{x \in K}{\text{argmin}}\, f(x)$.

$K_1 \leftarrow K$
for $t \in [T]$ **do**
  $x_t \leftarrow \text{centroid}(K_t)$
  $K_{t+1} \leftarrow K_t \cap \{y \in \mathbb{R}^n : \langle \nabla f(x_t), y - x_t \rangle \leq 0\}$
return $x_{\text{argmin}_{t \in [T]} f(x_t)}$

The half-space added is because we want to consider the halfspace of vectors negatively correlated with the gradient.

Theorem 1.23 (Grunbaum)

For any convex set $K \in \mathbb{R}^n$ and halfspace pointing H pointing through centroid(K),

$$\frac{1}{\mathrm{e}} \leq \frac{\lambda(K \cap H)}{\lambda(K)} \leq 1 - \frac{1}{\mathrm{e}}.$$

Theorem 1.24

Suppose $B \geq 0$ such that $f \in C^1(K, [-B, B])$. If $\widehat{x}$ is the reslt of the algorithm, then

$$f(\widehat{x}) \leq f(x_*) + 4B \exp\left(-\frac{T}{3n}\right).$$

Further, for any $\varepsilon \leq 1$, and $T \geq 3n \log\left(\frac{4B}{\varepsilon}\right)$,

$$f(\widehat{x}) \leq f(x_*) + \varepsilon.$$

Proof. Define

$$K^\delta = \{(1-\delta)x_* + \delta x_t : x \in K\}$$

as a scaled-down version of K centered at x_*. Then

$$\mu(K^\delta) = \mu(K)\delta^n.$$

Furthermore, if $x \in K$ and $y = (1-\delta)x_* + \delta x \in K^\delta$, then

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

Using Grunbaum's lemma,

$$\mu(K_t) \leq \left(1 - \frac{1}{\mathrm{e}}\right)^t \mu(K).$$

Define

$$\delta = 2\left(1 - \frac{1}{\mathrm{e}}\right)^{T/n}$$

then

$$\mu(K_T) \leq \mu(K^\delta)$$

so some point of K^δ must have been cut off.

Consider a step t such that $K^\delta \subseteq K_t$ yet $K^\delta \subseteq K_{t+1}$. Let $y \in K^\delta \cap (K_t \setminus K_{t+1})$ be a point that is cut off. By convexity,

$$f(y) \geq f(x_t) + \langle \nabla f(x_t), y - x_t \rangle$$

and since $y \in K_t \setminus K_{t+1}$, $\langle \nabla f(x_t), y - x_t \rangle > 0$. Hence the centroid has value

$$f(x_t) < f(y) \leq f(x_*) + 2\delta B.$$

Since $\widehat{x}$ is the centroid such that $f(\widehat{x})$ is minimal,

$$f(\widehat{x}) \leq f(x_*) + 2B \cdot 2\left(1 - \frac{1}{\mathrm{e}}\right)^{T/n} \leq 4B \exp\left(-\frac{T}{3n}\right).$$

The second claim follows by substituting $T \geq 3n \log\left(\frac{4B}{\varepsilon}\right)$ and simplifying. □

The number of iterations T to get an error of ε depends on $\log\left(\varepsilon^{-1}\right)$; gradient descent requires $O(\varepsilon^{-2})$ steps. Thus this algorithm is truly polynomial time. But it has heavy dependence on the dimension; gradient descent does not normally explicitly depend on the dimension.

The main issue with this algorithm is that we don't know how to compute the centroid. Computers are not good at taking symbolic integrals.

1.10 Ellipsoid Algorithm

We need to discuss a definition:

Definition 1.25 (Strong Separation Oracle)
For a convex set $K \subseteq \mathbb{R}^n$, a *strong separation oracle* for K is an algorithm that takes a point $z \in \mathbb{R}^n$ and correctly outputs one of:

(i) `Yes` (i.e., $z \in K$), or

(ii) `No` (i.e., $z \notin K$), as well as a *separating hyperplane* given by $a \in \mathbb{R}^n, b \in \mathbb{R}$ such that $K \subseteq \{x : \langle a, x\rangle \leq b\}$ but $\langle a, z\rangle > b$.

Theorem 1.26 (Separation implies Optimization)
Given a LP

$$\underset{x \in K}{\text{minimize}}\, c^* x$$

for a polytope $K = \{x \in \mathbb{R}^n : Ax \geq b\}$, and given access to a strong separation oracle for K, the ellipsoid algorithm produces a vertex solution for the LP in time $\text{poly}(n, \max_i \text{bits}(a_i), \max_i \text{bits}(b_i), \text{bits}(c))$.

There is no dependence on the number of constraints in the LP; we can get a basic solution to any finite LP as long as each constraint has a reasonable bit complexity, and we can define a separation oracle for the polytope. This is often summarized by saying: "separation implies optimization".

Theorem 1.27
Given a convex set $K \in \mathbb{R}^n$ described by a strong separation oracle, and two scalars $R, r > 0$, suppose we are guaranteed that

- $K \subseteq B_R(0)$, and
- Either $K = \varnothing$, or else $B_r(c) \subseteq K$ for some $c \in \mathbb{R}^n$.

Then we need to figure out which of the two cases in condition (b) holds; moreover, if $K \neq \varnothing$ then we also need to find a point $x \in K$. We can do this task in T oracle calls, for $T = 2(n+1)\log\left(\frac{R}{r}\right)$.

Proof. We use an algorithm:

Algorithm 7 Ellipsoid algorithm for feasibility.

Input: A convex set $K \subseteq \mathbb{R}^n$
Input: A strong separation oracle S for K.
Input: A scalar $R > 0$ such that $K \subseteq B_R(0)$.
Input: A scalar $r > 0$ such that either $K = \varnothing$ or $B_c(r) \subseteq K$ for some $c \in \mathbb{R}^n$.
Output: Whether $K = \varnothing$, or not and some point in K.

```
E_1 ← B_R(0)
x_1 ← 0
T ← 2(n + 1) log(R/r)
for t ∈ [T] do
    if S(x_t, K) = Yes then
        return feasible, x_t
    else
        H_t ← S(x_t, K)                          ▷ x_t ∉ H_t but K ⊆ H_t, so K ⊆ E_t ∩ H_t.
        E_{t+1} ← ellipsoid containing E_t ∩ H_t with center x_{t+1}.   ▷ μ(E_t ∩ H_t) ≤ (1/2)μ(E_t)
return infeasible
```

To show that we make progress, we need $\mu(\mathcal{E}_{t+1}) \ll \mu(\mathcal{E}_t)$. Indeed, by a result that will follow,

$$\frac{\mu(\mathcal{E}_{t+1})}{\mu(\mathcal{E}_t)} \leq \mathrm{e}^{-\frac{1}{2(n+1)}}.$$

Thus after $2(n+1)$ iterations the ratio of the measures falls by $\frac{1}{\mathrm{e}}$. Now we are done because

$$\mu(K) \leq \mu(B_R(0)) \quad \text{and} \quad K \neq \varnothing \iff \mu(K) \geq \mu(B_r(0)).$$

Therefore, after $2(n+1)\log\left(\frac{R}{r}\right)$ steps, none of the ellipsoid centers are contained in K, we know that $K = \varnothing$. □

Now we want to solve $\min_{x \in K} f(x)$.

Theorem 1.28
Given a convex set $K \in \mathbb{R}^n$ described by a strong separation oracle, and two scalars $R, r > 0$, suppose we are guaranteed that

- $K \subseteq B_R(0)$, and
- Either $K = \varnothing$, or else $B_r(c) \subseteq K$ for some $c \in \mathbb{R}^n$.

Then we need to figure out which of the two cases in condition (b) holds; moreover, if $K \neq \varnothing$ then we also need to find $\operatorname{argmin}_{x \in K} f(x)$. We can do this task in T oracle calls, for $T = 2(n+1)\log\left(\frac{R}{r}\right)$.

Proof. We use an algorithm:

Algorithm 8 Ellipsoid algorithm for optimization.

Input: A strong separation oracle S, nominally for K.
Input: A convex set $K \subseteq \mathbb{R}^n$.
Input: A convex function $f \in C^1(K, \mathbb{R})$
Input: A scalar $R > 0$ such that $K \subseteq B_R(0)$.
Input: A scalar $r > 0$ such that either $K = \varnothing$ or $B_c(r) \subseteq K$ for some $c \in \mathbb{R}^n$.
Output: Whether $K = \varnothing$, or not and a guess for $x_* = \operatorname{argmin}_{x \in K} f(x)$.

$K_1 \leftarrow K$
$\mathcal{E}_1 \leftarrow B_R(0)$
$x_1 \leftarrow 0$
$T \leftarrow 2(n+1)\log\left(\frac{R}{r}\right)$
for $t \in [T]$ **do**
  if $S(x_t, K_t) =$ `Yes` **then**
    $H_t \leftarrow \{x \in \mathbb{R}^n : \langle \nabla f(x_t), x - x_t \rangle \le 0\}$ ▷ $K_t \cap H_t$ contains all points in K_t with value $\le f(x_t)$
  else
    $H_t \leftarrow S(x_t, K_t)$
  $K_{t+1} \leftarrow K_t \cap H_t$
  $\mathcal{E}_{t+1} \leftarrow$ ellipsoid containing $\mathcal{E}_t \cap H_t$ with center x_{t+1} ▷ $K_t \subseteq \mathcal{E}_t \implies K_{t+1} \subseteq \mathcal{E}_{t+1}$
if $S(x_t, K_t) =$ `Yes` for any t **then**
  return $\widehat{x} = x_{\operatorname{argmin}_t f(x_t)}$
else
  return `infeasible`

The algorithm is a combination of the centroid algorithm and ellipsoid algorithm for feasibility. □

One issue is that we make queries to a separation oracle for K_t, but we are only given a strong separation oracle for $K_1 = K$. We can build strong separation oracle for K_t inductively. Given a strong separation oracle for K_{t-1}, we build one for $K_t = K_{t-1} \cap H_{t-1}$ as follows: *given $z \in \mathbb{R}^n$, query the oracle for K_{t-1} at z. If $z \notin K_{t-1}$. If $z \notin K_{t-1}$, then $H_t = H_{t-1}$ separates z and K_t. Else, if $z \in K_{t-1}$, check if $z \in H_{t-1}$. If so, $z \in K_t = K_{t-1} \cap H_{t-1}$. Otherwise, the half-space $H_t = H_{t-1}$ separates z and K_t.*

Theorem 1.29 (Idealized Ellipsoid Convex Optimization)
Given K, r, R as above (and a strong separation oracle) and a convex function $f \in C^1(K, [-B, B])$ convex, the ellipsoid algorithm run for $T = 2(n+1)\log\left(\frac{R}{r}\right)$ steps either correctly reports that $K = \varnothing$ or produces a point $\widehat{x}$ such that

$$f(\widehat{x}) \le f(x_*) + \frac{2BR}{r}\exp\left(-\frac{T}{2n(n+1)}\right).$$

The last thing to do to check if this algorithm is actually viable is to examine how to pick a suitable ellipsoid $\mathcal{E}_{t+1}$ which contains $\mathcal{E}_t \cap H_t$ and yet is small enough so that $\mu(\mathcal{E}_{t+1}) \le \mu(\mathcal{E}_t)\mathrm{e}^{-\frac{1}{2(n+1)}}$. The natural first question is what exactly is an ellipsoid.

Definition 1.30 (Ellipsoid)
A *ellipsoid* is the image of the unit ball under an invertible linear transformation L. The linear transformation

yields

$$\begin{aligned} L(B_1(0)) &= \{Lx : x^*x \le 1\} = \{y : (L^{-1}y)^*(L^{-1}y) \le 1\} \\ &= \{y : y^*(LL^*)^{-1}y \le 1\} \\ &= \{y : y^*Q^{-1}y \le 1\} \\ &= \mathcal{E}(0, Q) \end{aligned}$$

where $Q = LL^*$ is a positive definite matrix. For an ellipsoid centered at c we simply write

$$\mathcal{E}(c, Q) = \{y + c : y^*Q^{-1}y \le 1\} = \{y : (y-c)^*Q^{-1}(y-c) \le 1\}.$$

In the above problems we are given an ellipsoid $\mathcal{E}_t = \mathcal{E}(x_t, Q_t)$ and a half-space H_t that does not contain x_t. We want to find a matrix Q_{t+1} and a center x_{t+1} such that the resulting ellipsoid $\mathcal{E}_{t+1} = \mathcal{E}(x_{t+1}, Q_{t+1})$ contains $\mathcal{E}_t \cap H_t$ and satifies

$$\frac{\mu(\mathcal{E}_{t+1})}{\mu(\mathcal{E}_t)} \le \mathrm{e}^{-\frac{1}{2(n+1)}}.$$

It suffices to do this when $\mathcal{E}_t$ is a unit ball; when $\mathcal{E}_t$ is a general ellipsoid, we apply $Q_t^{-1/2}$ to convert $\mathcal{E}_t$ to a ball, find the smaller ellipsoid to it, and then apply $Q_t^{1/2}$ to get the final smaller ellipsoid.

The claim for general ellipsoids follows:

Theorem 1.31 (Halving an Ellipsoid)
Given an ellipsoid $\mathcal{E}_t = \mathcal{E}(x_t, Q_t)$ and a separating hyperplane $a_t^*(x - x_t) \le 0$ through its center, the new ellipsoid $\mathcal{E}_{t+1}$ with center x_{t+1} and PD matrix Q_{t+1} is found by taking

$$x_{t+1} = x_t - \frac{h}{n+1} \quad \text{and} \quad Q_{t+1} = \frac{n^2}{n^2-1}\left(Q_t - \frac{2}{n+1}hh^*\right)$$

where $h = \sqrt{a_t^* Q_t a_t}1$, a multiple of the all-ones vector.

Proof assuming $\mathcal{E}_t = B_1(0)$. By rotational symmetry, we might as well find a small ellipsoid that contains

$$K = B_1(0) \cap \{x : x_1 \ge 0\}.$$

By symmetry, it makes sense that the center of this new ellipsoid $\mathcal{E}$ should be of the form

$$x = (x_1, 0, \dots, 0).$$

Again by symmetry, the ellipsoid can be axis-aligned, with semi-axes of length a along e_1, and $b > a$ along all the other coordinate axes. Moreover, for $\mathcal{E}$ to contain the unit ball, it should contain the points $e_1, \dots, e_n$. So

$$\frac{(1-x_1)^2}{a^2} \le 1 \quad \text{and} \quad \frac{x_1^2}{a^2} + \frac{1}{b^2} \le 1.$$

If these two inequalities are tight then

$$a = 1 - x_1 \quad \text{and} \quad b = \sqrt{\frac{(1-x_1)^2}{(1-x_1)^2 - x_1^2}} = \sqrt{\frac{(1-x_1)^2}{1-2x_1}}.$$

Moreover, the ratio of measure of the ellipsoid to that of the ball is

$$ab^{n-1} = (1-x_1)\left(\frac{(1-x_1)^2}{1-2x_1}\right)^{(n-1)/2}$$

which is minimized by setting $x_1 = \frac{1}{n+1}$ which gives us

$$\frac{\mu(\mathcal{E})}{\mu(B_1(0))} \leq \mathrm{e}^{-\frac{1}{2(n+1)}}.$$

□

2 Linear Algebraic Primitives

The most generic thing we want to do is to solve a linear system, for $A \in \mathbb{R}^{n \times n}$ and $x, b \in \mathbb{R}^n$:

$$Ax = b.$$

There are two main ways to solve such systems:

- Gaussian elimination (and optimizations such as LU decomposition). This requires $O(n^3)$ runtime complexity, $O(n^2)$ space compleixty, and no dependence on spectral properties of A.
- Iterative methods (such as gradient descent, steepest descent, conjugate gradient descent, etc.). This requires $O(n)$ space complexity, yet relies on spectral properties and sparsity of A.

2.1 Gradient Descent

Suppose $A \in \text{PSD}(n)$, and $x, b \in \mathbb{R}^n$. The reason we may assume $A \in \text{PSD}(n)$ is because

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

and the second equation always has an exact solution (if $\text{rank}(A) = n$); and the solution set of the second equation is a superset of the solution set of the first.

The basic idea is that if $f(x) = \frac{1}{2} \langle Ax, x \rangle - \langle b, x \rangle$, then $\nabla f(x) = Ax - b$. The key point is

$$x_* = \underset{x}{\text{argmin}}\, f(x) \implies Ax_* = b.$$

We minimize $f(x)$ via gradient descent.

Algorithm 9 Gradient descent to minimize quadratic objective.

Input: $A \in \text{PSD}(n)$.
Input: $b \in \mathbb{R}^n$.
Input: A time horizon T.
Input: A learning rate η.
Output: A minimizer $x_* = \text{argmin}_x \left(\frac{1}{2} \langle Ax, x \rangle - \langle b, x \rangle \right)$.

```
x_1 ← 0
for t ∈ [T] do
    return x_{t+1} ← x_t − η (A x_t − b)
return x̂ = x_T
```

The time for each iteration is one matrix-vector multiplication (which takes time $\|A\|_0$) and some vector arithmetic ($O(n)$).

Theorem 2.1
If $A \in \text{PSD}(n)$ and $x, b \in \mathbb{R}^n$, then after T iterations of gradient descent,

$$\|x_T - x_*\|_2 \leq R e^{-T/\kappa(A)}$$

where $R = \|x_1 - x_*\|_2$.

Proof. We have

$$\begin{aligned} x_{t+1} &= x_t - \eta\left(Ax_t - b\right) \\ x_{t+1} - x_* &= x_t - x_* - \eta A x_t + \eta b \\ &= \left(I - \eta A\right) x_t + \eta b - x_* \\ &= \left(I - \eta A\right) x_t + \eta A x_* - x_* \\ &= \left(I - \eta A\right)\left(x_t - x_*\right). \end{aligned}$$

Now we can write

$$x_{t+1} - x_* = \left(I - \eta A\right)^t \left(x_1 - x_*\right)$$

and taking the norm,

$$\left\|x_{t+1} - x_*\right\|_2 = \left\|\left(I - \eta A\right)^t \left(x_1 - x_*\right)\right\|_2 \leq \left\|\left(I - \eta A\right)^t\right\|_2 \left\|x_1 - x_*\right\|_2 \leq \left\|I - \eta A\right\|_2^t \left\|x_1 - x_*\right\|_2.$$

In this case $\|A\|_2$ is the operator norm. Then if $\eta = \frac{1}{\lambda_{\max}(A)}$,

$$\left\|I - \eta A\right\|_2 = \max\left\{\left(1 - \eta\lambda_i(A)\right)_{i\in[n]}\right\} = \max\left\{\left(1 - \frac{\lambda_i(A)}{\lambda_{\max}(A)}\right)_{i\in[n]}\right\} = 1 - \frac{\lambda_{\min}(A)}{\lambda_{\max}(A)} - 1 - \kappa(A).$$

Thus

$$\left\|x_{t+1} - x_*\right\|_2 \leq \left(1 - \kappa(A)\right)^t \left\|x_1 - x_*\right\|_2 \leq R\mathrm{e}^{t/\kappa(A)}.$$

□

Remark 2.2. One can choose better η values, for example a time-varying

$$\eta_t = \frac{(Ax_t - b)^*(Ax_t - b)}{(Ax_t - b)^* A (Ax_t - b)}$$

has error rate

$$\left\|x_T - x_*\right\|_2 \leq R\left(\frac{\lambda_{\max}(T) - \lambda_{\min}(T)}{\lambda_{\max}(T) + \lambda_{\min}(T)}\right)^{2T}.$$

The *conjugate gradient method*, a variant of this, has runtime proportional to $\mathrm{e}^{-t/\sqrt{\kappa(A)}}$. But everything is dependent on the condition number, which may not be helpful for analysis.

There is a variant called *preconditioned gradient descent*, which has the algorithm:

Algorithm 10 Gradient descent to minimize quadratic objective.

Input: $A \in \mathrm{PSD}(n)$ and $L \in \mathbb{R}^{n\times n}$ invertible.
Input: $b \in \mathbb{R}^n$.
Input: A time horizon T.
Input: A learning rate η.
Output: A minimizer $x_* = \mathrm{argmin}_x\left(\frac{1}{2}x^*Ax - b^*x\right)$.

```
x_1 ← 0
for t ∈ [T] do
    return x_{t+1} ← x_t − ηL^{-1}(Ax_t − b)
return x̂ = x_T
```

The analysis is the same, except we can replace $\kappa(A)$ with $\kappa(L^{-1}A)$. We want to find L such that $\kappa(L^{-1}A) < \kappa(A)$. One sticking point is that we have to do an extra multiplication per iteration in $L^{-1}(Ax_t - b)$. This means that we need to solve the system $Lw = Ax_t - b$ for w. So we need to find L such that solving linear systems in L is easy.

This usually can be done by a problem-based heuristic, in which we take L to be an approximation of A in the sense of the problem.

2.2 Principal Components Analysis

Suppose we have a large number of points $(x_i)_{i\in[n]} \in \mathbb{R}^d$. We want to find the best one-dimensional approximation for the dataset:

$$\min_v \sum_{i\in[n]} \|x_i - v\|_2^2 = \max_v \sum_{i\in[n]} \left\|\Pi_{\operatorname{span}(v)}(x_i)\right\| = \max_{v\colon \|v\|_2=1} \sum_{i\in[n]} \langle v, x_i\rangle .$$

We can find the best k-dimensional approximation:

$$\max_{W\colon \dim(W)=k} \sum_{i\in[n]} \|\Pi_W(x_i)\|_2^2 .$$

A greedy algorithm can compute the best k-dimensional subspace. Suppose $W = \operatorname{span}(v_1, \dots, v_k)$ where

$$v_j = \operatorname*{maximize}_{v\perp \operatorname{span}((v_l)_{l\in[j-1]})} \sum_{i\in[n]} \langle x_i, v\rangle^2$$

Then we can iterate from $j = 1$ to $j = k$.

This process is called *Principal Components Analysis (PCA)*.

Algorithm 11 Principal component analysis.

Input: A dataset $X = (x_i)_{i\in[n]} \in \mathbb{R}^d$.
Input: An integer $k \in [n]$.
Output: The subspace W which is the best k-dimensional approximation to X.

```
W = ∅
for j ∈ [k] do
    v_j ← maximize_{v∈ℝ^n, ‖v‖_2≤1, v⊥W} Σ_{i∈[n]} ⟨x_i, v⟩^2
    W ← W ⊕ span(v_j)
return W                                   ▷ We can return (v_j)_{j∈[k]}.
```

It remains to actually solve the optimization problem. If we write

$$A = \begin{bmatrix} x_1^* \\ \vdots \\ x_n^* \end{bmatrix}$$

then

$$\sum_{i=1}^n \langle x_i, v\rangle^2 = \|Av\|_2^2 = v^*A^*Av.$$

Then

$$\max_{v\colon \|v\|_2\le 1} v^*A^*Av = \lambda_{\max}(A^*A).$$

To obtain the largest eigenvalue, one can use Gaussian elimination (or more particularly QR decomposition), or iterative methods. The one we will focus on is power iteration.

Let $B = A^*A$ and let $0 \le \lambda_1 \le \cdots \le \lambda_n$ be the eigenvalues of B, with λ_i corresponding to v_i. We want to

compute v_n. To do this, we write

$$\begin{aligned} x &= \sum_{i \in [n]} c_i v_i \\ Bx &= \sum_{i \in [n]} c_i \lambda_i v_i \\ B^2 x &= \sum_{i \in [n]} c_i \lambda_i^2 v_i \\ &\vdots \quad \vdots \\ B^T x &= \sum_{i \in [n]} c_i \lambda_i^T v_i. \end{aligned}$$

Since λ_n is the largest eigenvector, for large T the term $c_i \lambda_i^T v_i$ dominates, and the answer $B^T x$ is approximately $c_n \lambda_n^T v_n$, which is in the direction of v_n.

Formally, the algorithm is:

Algorithm 12 Power method for calculating largest eigenvector.

Input: A matrix $A \in \text{PSD}(n)$.
Input: A time horizon T.
Output: A vector approximately in the direction of $v_n(A)$.

```
y ← any nonzero vector
for t ∈ [T] do
    y ← By
return y / ‖y‖_2.
```

To compute the top vector orthonormal to a subspace space W, we can solve this problem on the matrix

$$A = \begin{bmatrix} x_1 - \Pi_W(x_1) \\ \vdots \\ x_n - \Pi_W(x_n) \end{bmatrix}.$$

Another thing we can do is to modify the power method:

Algorithm 13 Power method for calculating the k largest eigenvector.

Input: A matrix $A \in \text{PSD}(n)$.
Input: An integer $k \in [n]$
Input: A time horizon T.
Output: Vectors which are approximately in the direction of $(v_i)_{i \in [n] \setminus [n-k]}$.

```
(y_i)_{i∈[k]} ← any non-equal, nonzero vectors
(y_i)_{i∈[k]} ← ORTHONORMALIZE((y_i)_{i∈[k]})
for t ∈ [T] do
    for i ∈ [K] do
        y_i ← By_i
    (y_i)_{i∈[k]} ← ORTHONORMALIZE((y_i)_{i∈[k]})
return (y_i)_{i∈[k]}.
```

To get accuracy δ, that is, $\|v - v_*\| \leq \delta$, it requires

$$T \geq \frac{\log(1/\delta)}{\lambda_n - \lambda_{n-1}}.$$

This denominator is called the *spectral gap*; for the general case it can be written as $\lambda_{n-k+1} - \lambda_{n-k}$.

2.3 Singular Value Decomposition

Suppose $A \in \mathbb{R}^{m \times n}$ with $\operatorname{rank}(()\, A) = r$. Then we can write

$$A = \underbrace{U}_{\mathbb{R}^{m \times d}} \underbrace{\Sigma}_{\mathbb{R}^{d \times d}} \underbrace{V^*}_{d \times n}$$

whence $U = \begin{bmatrix} u_1 & \cdots & u_d \end{bmatrix}$ is an orthonormal basis for $\operatorname{im}(A)$ and $V = \begin{bmatrix} v_1 & \cdots & v_d \end{bmatrix}$ is an orthonormal basis for $\operatorname{im}(A^*)$. We can also write

$$A = \sum_{i=1}^{d} \sigma_i u_i v_i^*$$

where $\sigma_I = \Sigma_{ii}$ is the i^{th} *singular value*. This factorization is the *Singular Value Decomposition (SVD)*. If A is an $n \times n$ symmetric matrix, then SVD reduces to diagonalization. Conventionally we order u_i, σ_i, v_i in decreasing order of σ_i, that is, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d$.

The set of rank k matrices $\mathbb{R}_k^{m \times n}$ for $k < d$ are a lower-dimensional manifold. Write

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^*.$$

Theorem 2.3 (Eckart-Young)
The matrix A_k is the "best" approximation to A in both the Frobenius norm sense and the operator norm sense:

$$\operatorname*{argmin}_{M \in \mathbb{R}_k^{m \times n}} \|A - B\|_F = \operatorname*{argmin}_{M \in \mathbb{R}_k^{m \times n}} \|A - B\|_2 = A_k.$$

2.4 Applications of PCA/SVD

2.4.1 Identifying Mixture of Gaussians

Suppose we are given two distributions $X_1 \sim \mathcal{N}(\mu_1, I_d)$ and $X_2 \sim \mathcal{N}(\mu_2, I_d)$. Let $U \in \Delta_{\{1,2\}}$ and $X = X_U$. Then X is a *Mixture of Gaussians*. We are given some $(x_i)_{i \in [n]} \sim X$, and want to find out which x_i come from the same Gaussian x_1 or x_2. Our initial algorithm to do this is quite simple.

Algorithm 14 Separating Gaussians, attempt 1.

Input: A dataset $(x_i)_{i\in[n]}$.
Output: Two disjoint sets $S \cup T = \{x_i\}_{i\in[n]}$ which partition $(x_i)_{i\in[n]}$ depending on the distribution the points are sampled from.
$U_1 \leftarrow \operatorname{argmin}_{I\subseteq[n]} \max_{(i,j)\in I^2} \|x_i - x_j\|_2$
$U_2 \leftarrow [n] \setminus U_1$
$S \leftarrow \{x_u : u \in U_1\}$
$T \leftarrow \{x_u : u \in U_2\}$
return S, T

In this case $\widehat{\mu}(Z)$ is some estimator of $\mathsf{E}_{Z\sim X}[X]$.

We analyze this algorithm. Most probability mass lies on an annulus of width $O(1)$ at radius $\sqrt{d}$. Also $\mathrm{e}^{-\|x\|_2^2/2} = \prod_{i=1}^d \mathrm{e}^{-x_i^2/2}$ and almost all of the mass is within the slab $\left\{x \in \mathbb{R}^d : -c \le x_1 \le c\right\}$, for $c \in O(1)$. Pick a point x from this Gaussian. After picking x, rotate the coordinate system to make the first axis align with x. Independently pick a second point y from the Gaussian. The fact that almost all of the probability mass of the Gaussian is within the slab $\left\{x \in \mathbb{R}^d : -c \le x_1 \le c\right\}$ at the equator implies that y's component along x's direction is $O(1)$ with high probability. Thus y is nearly orthogonal to x, so $\|x - y\|_2 \approx \sqrt{\|x\|_2^2 + \|y\|_2^2}$.

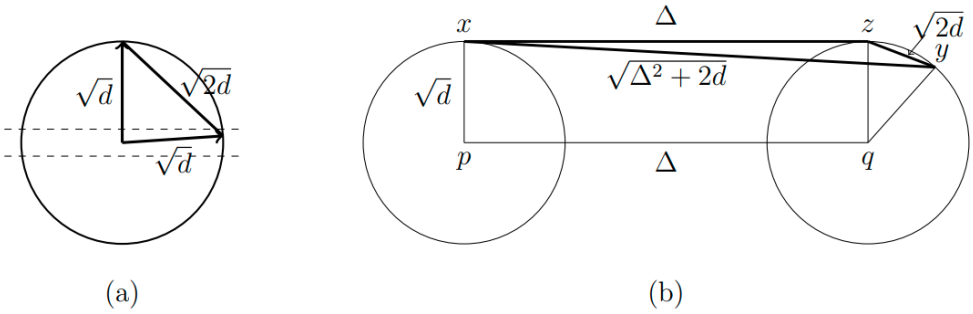


Figure 2.1: (a) indicates that two randomly chosen points in high dimension are surely almost nearly orthogonal. (b) indicates the distance between a pair of random points from two different unit balls approximating the annuli of two Gaussians.

More precisely, since the coordinate system has been rotated so that x is at the North Pole, $x = (\sqrt{d} \pm O(1), 0, \ldots, 0)$. Since y is almost on the equator, further rotate the coordinate system so that the component of y that is perpendicular to the axis of the North Pole is in the second coordinate. Then $y = (O(1), \sqrt{d} \pm O(1), 0, \ldots, 0)$. Thus,

$$\|x - y\|_2^2 = d \pm O(\sqrt{d}) + d \pm O(\sqrt{d}) = 2d \pm O(\sqrt{d})$$

which implies $\|x - y\|_2 = \sqrt{2d} \pm O(1)$ with high probability.

Consider if $\|\mu_1 - \mu_2\|_2 = \Delta$. Then if $x \sim X_1, y \sim X_2$, then since $x - p$, $p - q$, and $q - y$ are nearly mutually orthogonal,

$$\|x - y\|_2^2 = \|(x - p) + (p - q) + (q - y)\|_2^2 \approx \|x - p\|_2^2 + \|p - q\|_2^2 + \|q - y\|_2^2 = \Delta^2 + 2d + O(\sqrt{d}) \implies \|x - y\|_2 \approx \sqrt{}$$

To ensure that the distance between two points picked from the same Gaussian are closer to each other than two points picked from different Gaussians requires that the upper limit of the distance between a pair of points from the

same Gaussian is at most the lower limit of distance between points from different Gaussians. This requires that

$$\sqrt{2d} + O(1) \leq \sqrt{2d + \Delta^2} - O(1) \quad \text{or} \quad 2d + O(\sqrt{2}) \leq 2d + \Delta^2$$

which holds when $\Delta \; in \omega(d^{1/4})$. Thus, mixtures of spherical Gaussians can be separated in this way, provided their centers are separated by $\omega(d^{1/4})$. If we have n points and want to correctly separate all of them with high probability, we need our individual high-probability statements to hold with probability $1 - \frac{1}{\text{poly}(n)}$, which means our $O(1)$ terms become $O(\sqrt{\log(n)})$. So we need to include an extra $O(\sqrt{\log(n)})$ in the separation distance.

One can actually separate Gaussians where the centers are much closer. The trick is to use a coordinate system learned from data instead of the standard axes. Indeed, projecting all points onto $W = \text{span}(\mu_1, \mu_2)$ does the trick, and makes $d = 2$ so that $\Delta \in \omega(1)$ works. To estimate W, we take the first two principal components of the data matrix, and take the projections onto the first two principal components. Hence, PCA.

2.4.2 Robust Mean Estimation

In this section, we illustrate the main insights underlying recent robust high-dimensional learning algorithms by focusing on the problem of robust mean estimation.

The objective of this section is to provide the intuition and background required to develop robust learning algorithms in an accessible way. As such, we will not attempt to optimize the sample or computational complexities of the algorithms presented, other than to show that they are polynomial in the relevant parameters

In the problem of robust mean estimation, we are given an ε-corrupted set of samples from a distribution X on $\mathbb{R}^d$ (that is, n samples, εn which are corrupted in some way and $(1-\varepsilon)n$ which are not) and our goal is to approximate the mean of X, within small error in ℓ_2-norm. In order for such a goal to be information-theoretically possible, it is required that X belongs to a suitably well-behaved family of distributions. A typical assumption is that X belongs to a family whose moments are guaranteed to satisfy certain conditions, or equivalently, a family with appropriate concentration properties. In our initial discussion, we will use the running example of a spherical Gaussian, although the results presented here hold in greater generality. That is, the we imagine that $X = \mathcal{N}(\mu, I)$ for some $\mu \in \mathbb{R}^d$.

Arguably the most natural idea to robustly estimate the mean of a distribution would be to identify the outliers and output the empirical mean of the remaining points. The key conceptual difficulty is the fact that, in high dimensions, the outliers cannot be identified at an individual level even when they move the mean significantly. In many cases, we can easily identify the "extreme outliers" via a pruning procedure exploiting the concentration properties of the inliers. Alas, such naive approaches typically do not suffice to obtain non-trivial error guarantees.

The simplest example illustrating this difficulty is that of a high-dimensional spherical Gaussian. Typical samples $x \sim \mathcal{N}(\mu, I)$ have $\|x - \mu\|_2 \approx \Theta(\sqrt{d})$. That is, we can certainly identify as outliers all points of our dataset at distance more than $\Omega(\sqrt{d})$ from the cooridnate-wise median of the dataset. All other points cannot be removed via such a procedure, as this could result in removing many inliers as well. However, by placing an ε-fraction of outliers at distance $\sqrt{d}$ in the same direction from the unknown mean, an adversary can corrupt the sample mean by as much as $\Omega(\varepsilon\sqrt{d})$. This leaves the algorithm designer with a dilemma of sorts. On the one hand, potential outliers at distance $\Theta(\sqrt{d})$ from the unknown mean could lead to large ℓ_2-error, scaling polynomially with d. On the other hand, if the adversary places outliers at distance approximately $\Theta(\sqrt{d})$ from the true mean in *random directions*, it may be information-theoretically impossible to distinguish them from the inliers. The way out is the realization that *it is in fact not necessary to detect and remove all outliers*. It is only required that the algorithm can detect the "consequential outliers", i.e., the ones that can significantly impact our estimates of the mean.

Let us assume without loss of generality that there no extreme outliers (as these can be removed via pre-processing). Then *the only way that the empirical mean can be far from the true mean is if there is a "conspiracy" of many outliers, all producing errors in approximately the same direction.* Intuitively, if our corrupted points are at distance $O(\sqrt{d})$ from the true mean in random directions, their contributions will on average cancel out, leading to a small error in the sample mean. In conclusion, it suffices to be able to detect these kinds of conspiracies of outliers.

The next key insight is simple and powerful. Let T be an ε-corrupted set of points drawn from $\mathcal{N}(\mu, I)$. If such a conspiracy of outliers substantially moves the empirical mean $\widehat{\mu}(T)$, it must move $\widehat{\mu}(T)$ in some direction. That is, there is a unit vector v such that these outliers cause $\langle v, \widehat{\mu}(T) - \mu \rangle$ to be large. For this to happen, it must be the case that these outliers are on average far from μ in the v-direction. In particular, if an ε-fraction of corrupted points in T move the sample average of $\langle v, u - \mu \rangle$ where $u \sim \mathrm{Uni}(T)$, by more than δ (in this case δ is small but larger than ε), then on average these corrupted points x must have $\langle v, x - \mu \rangle \geq \frac{\delta}{\varepsilon}$. This means that these corrupted points will have a contribution of at least $\varepsilon \cdot (\frac{\delta}{\varepsilon})^2 = \frac{\delta^2}{\varepsilon}$ to the variance of $\langle v, u \rangle$. Fortunately, this condition can actually be algorithmically detected! In particular, by computing the top eigenvector of the sample covariance matrix, we can efficiently determine whether or not there is any direction v for which the variance of $\langle v, u \rangle$ is abnormally large.

This discussion leads us to the overall structure of the algorithm.

Algorithm 15 Robust mean estimation, attempt 1

Input: An ε-corrupted set of points $T \in \mathbb{R}^{n \times d}$ where $t_i \sim X$.
Output: An estimate $\widehat{\mu}(T)$ for $\mathsf{E}[X]$.

$\widehat{\Sigma} \leftarrow \dfrac{T^* T}{n}$

$\lambda_*, v_* \leftarrow$ largest eigenvalue and corresponding eigenvector of $\widehat{\Sigma}$

if $\lambda_* \gg \lambda_{\max}(\mathsf{Cov}(X))$ **then**

$T \leftarrow T \setminus \left(\text{points } t_i \in T \text{ where } \left\| \Pi_{\mathrm{span}(v_*)}(t_i) \right\|_2 \text{ is large} \right)$

return $\widehat{\mu} = \dfrac{1}{|T|} \displaystyle\sum_{i=1}^{|T|} t_i$

To quantify the performance of this algorithm, we require some definitions.

Definition 2.4 (Stability Condition)
Fix $0 < \varepsilon < \frac{1}{2}$ and $\delta \geq \varepsilon$. A finite set $S \subset \mathbb{R}^d$ is (ε, δ)-stable (w.r.t. X) if for every unit vector $v \in \mathbb{R}^d$ and every $S' \subseteq S$ with $|S'| \geq (1 - \varepsilon)|S|$, the following conditions hold:

- $\left| \dfrac{1}{|S'|} \displaystyle\sum_{x \in S'} \left\langle v, x - \mathsf{E}[X] \right\rangle \right| \leq \delta$, and
- $\left| \dfrac{1}{|S'|} \displaystyle\sum_{x \in S'} \left\langle v, x - \mathsf{E}[X] \right\rangle^2 - 1 \right| \leq \dfrac{\delta^2}{\varepsilon}$.

Theorem 2.5 (Certificate for Empirical Mean)
Let S be an (ε, δ)-stable set with respect to a distribution X, for some $\delta \geq \varepsilon > 0$. Let T be an ε-corrupted version of S. If $\|\Sigma_T\|_2 \leq 1 + \lambda$, then

$$\left\| \mu(T) - \mathsf{E}[X] \right\|_2 \leq O(\delta + \sqrt{\varepsilon \lambda})$$

2.5 Tensors

Definition 2.6 (Tensor)
A *tensor* of order o is an element of $\mathbb{R}^{\times_{i=1}^{o} n_i}$, that is, it's a collection of numbers with t coordinate indices.

Notation 2.7
If we can write

$$T_s = \prod_{i=1}^{o} u_{i,s_i}$$

for all $s \in \bigtimes_{i=1}^{o} [n_i]$, then we use the shorthand

$$T = \bigotimes_{i=1}^{p} u_i.$$

For example,

$$T_{i,j,k} = u_i v_j w_k \implies T = u \otimes v \otimes w.$$

Definition 2.8
The rank of a p-order tensor T is the smallest integer r so that we can write

$$T = \sum_{i=1}^{r} \bigotimes_{j=1}^{p} u_j^{(i)}.$$

This is a natural generalization of the rank of a matrix M is the smallest integer r so that M can be written as the sum of r rank one matrices.

Remark 2.9. Tensors can also be linear maps. Let $p + q = o$. Then a tensor T can be written as a linear map from $\mathbb{R}^p \times (\mathbb{R}^q)^* \to \mathbb{R}$.

Remark 2.10. The $\operatorname{rank}(T)$ is a very finicky quantity. Almost every tensors $T \in \mathbb{R}^{(n^3)}$ have $\operatorname{rank}(T) \geq n^2$, but all explicit matrices we can construct have $\operatorname{rank}(T) = \Theta(n)$. Also, the rank depends on whether we are working over $\mathbb{R}$ or $\mathbb{C}$. Finally, we can take a sequence of low rank tensors whose limit has high rank. So the notion is not very robust. Further, it is NP-hard to compute for an arbitrary tensor.

We can still make a good eigenvalue decomposition. Suppose a tensor T can be written as

$$T = \sum_{i=1}^{n} \bigotimes_{j=1}^{o} a_i = \sum_{i=1}^{n} a_i^{\otimes 3}$$

where $\{a_i\}_{i \in [n]}$ are orthonormal vectors. Then $\operatorname{rank}(T) = n$. Pick a random vector $g \in \mathbb{R}^n$. Then we can use the notation

$$T(g, :) = \sum_{i=1}^{n} \langle a_i, g \rangle \, a_i^{\otimes (o-1)}.$$

This notation is also useful in other ways, for example

$$T(g, :, h) = \sum_{i=1}^{n} \langle a_i, g \rangle \, \langle a_i, h \rangle \, a_i^{\otimes (o-2)}$$

and so on.

Going back to this notation, consider

$$T(g_1, \dots, g_{o-2}, :, :) = \sum_{i=1}^{n} \left(\prod_{j=1}^{o-2} \langle a_i, g_j \rangle \right) a_i^{\otimes 2}.$$

This is a matrix with eigenvalues $\left\{ \prod_{j=1}^{o-2} \langle a_i, g_j \rangle \right\}_{i \in [n]}$ and corresponding eigenvectors $\{a_i\}_{i \in [n]}$. If T is as described, then

$$T(x, \dots, x) = \sum_{i=1}^{n} \langle a_i, x \rangle^o$$

with local optima $\{a_i\}_{i \in [n]}$ on the unit ball S^{n-1}.

2.6 Jennrich's Algorithm

Suppose we have a tensor $T \in \mathbb{R}^{m \times n \times p}$ that can be written as

$$T = \sum_{i=1}^{r} u_i \otimes v_i \otimes w_i$$

where $\{u_i\}_{i \in [r]}$ are linearly independent, $\{v_i\}_{i \in [r]}$ are linearly independent, and $\{w_i\}_{i \in [r]}$ are distinct.

Algorithm 16 Jennrich's algorithm

Input: A tensor $T \in \mathbb{R}^{m \times n \times p}$
Output: $\{u_i\}_{i \in [r]}$ linearly independent, $\{v_i\}_{i \in [r]}$ linearly independent, $\{w_i\}_{i \in [r]}$ distinct

$a, b \leftarrow$ random vectors in S^{p-1}
$T^{(a)} \leftarrow \sum_{i=1}^{p} a_i T_{:,:,i}$
$T^{(b)} \leftarrow \sum_{i=1}^{p} b_i T_{:,:,i}$
$U \leftarrow$ eigenvectors $\left(T^{(a)} \left(T^{(b)} \right)^{-1} \right)$
$V \leftarrow$ eigenvectors $\left(\left(T^{(a)} \right)^{-1} T^{(b)} \right)$
Solve for W given U, V
return U, V, W

This works since

$$T^{(a)} = \sum_{i=1}^{p} a T_{:,:,i} = \sum_{i=1}^{r} \langle w_i, a \rangle \, u_i \otimes v_i$$

and analogously for $T^{(b)}$, so

$$U = \begin{bmatrix} u_1 & \cdots & u_r \end{bmatrix} \qquad V = \begin{bmatrix} v_1^* \\ \vdots \\ v_r^* \end{bmatrix} \qquad D^{(a)} = \operatorname{diag}\left(\{ \langle a, w_i \rangle \}_{i \in [r]} \right) \qquad D^{(b)} = \operatorname{diag}\left(\{ \langle b, w_i \rangle \}_{i \in [r]} \right)$$

implies

$$T^{(a)} = U D^{(a)} V^* \quad \text{and} \quad T^{(b)} = U D^{(b)} V^*.$$

Then

$$T^{(a)} \left(T^{(b)} \right)^{-1} = U D^{(a)} V^* \left(U D^{(b)} V^* \right) = U D^{(a)} V^* V (D^{(b)})^{-1} U^* = U D^{(a)} (D^{(b)})^{-1} U^*$$

so we can recover U from the eigendecomposition. Also,

$$\left(T^{(a)}\right)^{-1} T^{(b)} = \left(UD^{(a)}V^*\right)^{-1} UD^{(b)}V^* = V(D^{(a)})^{-1}U^*UD^{(b)}V^* = V(D^{(a)})^{-1}D^{(b)}V^*$$

which proves the algorithm.

2.7 Independent Component Analysis

Suppose we have the model

$$y = Ax + b$$

for $A \in \mathbb{R}^{n\times n}$ unknown, $b \in \mathbb{R}^n$ unknown, $y \in \mathbb{R}^n$ known. Finally $x \in \mathbb{R}^n$ is a random variable such that $\mathsf{E}[x] = 0$ and $\mathsf{E}\left[x_i^2\right] = 1$ for all i. Further $\{x_i\}_{i\in[n]}$ are independent scalar random variables. We will give an algorithm for recovering A and b under the assumptions that A is nonsingular and x is not Gaussian (that is, $\mathsf{E}\left[x_i^4\right] \neq 3$). We rule out the case that x is Gaussian, because if x is Gaussian then for any rotation R, $ARx \sim Ax$.

Suppose we have some data $\{y_t\}_{t\in[T]}$. A natural thing to do is to consider the centered data using the transformation

$$\widehat{y} = y - \mathsf{E}[y] = y - \mathsf{E}[Ax + b] = y - b.$$

The new samples $\{\widehat{y}_t\}_{t\in[T]}$ have expectation 0. Thus without loss of generality our model becomes

$$y = Ax$$

with the same assumptions. The other thing to do is to whiten the data. To do that, we need to compute

$$\mathsf{Cov}(y) = \mathsf{E}\left[\left(y - \mathsf{E}[y]\right) y - \mathsf{E}[y]^*\right] = \mathsf{E}\left[yy^*\right] = \mathsf{E}\left[Axx^*A^*\right] = A\,\mathsf{E}\left[xx^*\right]A^* = A\,\mathsf{Cov}(x)\,A^*.$$

By our assumption that $\{x_i\}_{i\in[n]}$ are mutually independent and thus uncorrelated, and that $\mathsf{E}\left[x_i^2\right] = 1$, we know $\mathsf{Cov}(x) = I_n$. Thus

$$\mathsf{Cov}(y) = AA^*.$$

We are a rotation away from recovering A, since $AA^* = (AR)(AR)^*$ for any rotation R. But this is not quite what we want, we actually want to recover A up to permutation. The key here is to estimate

$$M_4 = \mathsf{E}\left[y^{\otimes 4}\right] - T$$

where

$$T_{a,b,c,d} = \mathsf{E}[y_a y_b]\,\mathsf{E}[y_c y_d] + \mathsf{E}[y_b y_c]\,\mathsf{E}[y_a y_d] + \mathsf{E}[y_a y_c]\,\mathsf{E}[y_b y_d].$$

Then we can estimate T by taking lots of samples and averaging. Then one can derive

$$M_4 = \sum_{i=1}^{n} \left(\mathsf{E}\left[x_i^4\right] - 3\right)\left(A_i^{\otimes 4}\right)$$

where A_i is the i^{th} column of A. Then M_4 is a rank n 4-tensor; we can run Jennrich's algorithm to recover each A_i.

3 Embeddings

Embeddings are loosely where our problem lives in some domain $\mathcal{D}$ which is easier to work with. We find a map ϕ which maps $\mathcal{D}$ to a range $\mathcal{R}$, whence the problem is easier to solve in $\mathcal{R}$ for some reason. For example, $\mathcal{R}$ could have nice structure, lower dimensionality, or nice properties that lend themselves to the problem. We would like ϕ to preserve properties of the original points in $\mathcal{D}$ in $\mathcal{R}$, for example the distances or angles or norms.

Throughout this section, we will discuss *metric embeddings*. Let (X, d_X) and (Y, d_Y) be metric spaces.

Definition 3.1 (Isometry)
A map $f : X \to Y$ is an *isometry on* $S \subseteq X$ if

$$d_Y(f(x), f(y)) = d_X(x, y) \quad \text{for all } x, y \in S.$$

Two metric spaces are *isometric* if there exists a bijective isometry between them.

If we don't specify the subset S, we assume $S = X$.

It is many times impossible to construct exact isometries beteween two metric spaces. We are okay in principle with some finite distortion of distances.

Definition 3.2 (Distortion)
An injection $f : X \to Y$ is a *D-embedding* for $S \subseteq X$, where $D \geq 1$, if there exists $r > 0$ such that

$$r d_X(x, y) \leq d_Y(f(x), f(y)) \leq D r d_X(x, y) \quad \text{for all } x, y \in S.$$

The *distortion of f on S* is

$$D_f = \inf\{D \in \mathbb{R} : f \text{ is a } D\text{-embeddding for } S\}.$$

If Y is a normed vector space, we can re-scale the data at will, and so we can choose the scaling factor r to be a convenient number, i.e, $r = 1$.

We now discuss a related notion of irregularity in an isometry, which will be more useful in the following sections.

Definition 3.3 (ε-Almost Isometry)
A map $f : X \to Y$ is an *ε-almost isometry* for $S \subseteq X$, where $\varepsilon > 0$, if

$$(1 - \varepsilon) d_X(x, y) \leq d_Y(f(x), f(y)) \leq (1 + \varepsilon) d_X(x, y) \quad \text{for all } x, y \in S.$$

For small ε, you can see by Taylor series that in a normed vector space this is similar to saying that f is a $(1 + 2\varepsilon)$-embedding.

3.1 Dimension Reduction via Random Projection

For now, we work in finite-dimensional normed vector spaces. Consider $(\mathbb{R}^n, \|\cdot\|_2)$ as the prototypical example.

3.1.1 Johnson-Lindenstrauss Lemma

The Johnson-Lindenstrauss lemma is the following fact:

Theorem 3.4 (Johnson-Lindenstrauss Lemma)
Let $\varepsilon \in (0,1)$ and let $P = \{p_i\}_{i\in[n]} \in \mathbb{R}^n$. Let $k \geq C\varepsilon^{-2}\log(n)$, where C is a large constant independent of ε. Then there is a mapping $f : \mathbb{R}^n \to \mathbb{R}^k$ which is an ε-almost isometry for P.

All proofs of this theorem rely on the following fact:

Lemma 3.5 (Random Projection Lemma – Informal)
Let $\varepsilon, \delta \in (0,1)$. Let $n, k \in \mathbb{N}$ with $k = O\left(\frac{\log(1/\delta)}{\varepsilon^2}\right)$. Let $T : \mathbb{R}^n \to \mathbb{R}^k$ be a "normalized random linear map". Then for every vector $x \in \mathbb{R}^n$ we have

$$\mathsf{P}[(1-\varepsilon)\|x\|_2 \leq \|T(x)\|_2 \leq (1+\varepsilon)\|x\|_2] \geq 1-\delta.$$

Proof of Johnson-Lindenstrauss Lemma assuming Random Projection Lemma. Choose k large enough (in particular $k = O\left(\frac{\log(n)+\log(1/\delta)}{\varepsilon^2}\right)$) so that the Random Projection Lemma yields

$$\mathsf{P}[(1-\varepsilon)\|x\|_2 \leq \|T(x)\|_2 \leq (1+\varepsilon)\|x\|_2] \geq 1-\frac{\delta}{n^2} \quad \text{for all } x \in \mathbb{R}^n.$$

Then we apply this to the $\binom{n}{2}$ vectors $\{p_i - p_j\}_{1\leq i<j\leq n}$, and use the union bound. Then T is an ε-almost isometry for P with probability at least $1-\delta$. Thus a suitable T exists. □

The obvious question is "what is a normalized random linear map"? Here are some examples.

(a) We can pick a random k-dimensional linear subspace of $\mathbb{R}^n$, and take T as the orthogonal projection onto it, scaled by the factor of $\sqrt{n/k}$.

(b) The Gaussian case. Define T by $T(x) = \frac{1}{\sqrt{k}}Ax$, where A is a random $k \times n$ matrix whose entries are i.i.d. $\mathcal{N}(0,1)$.

(c) The ± 1 case. Choose T as in (b) except that the entries of A are i.i.d. Bern(± 1).

We will prove the Gaussian case.

Theorem 3.6 (Random Projection Lemma with Independent Gaussian Coefficients)
Let $\varepsilon, \delta \in (0,1)$. Let $n, k \in \mathbb{N}$, with $k \leq \frac{8\log(2/\delta)}{\varepsilon^2}$. Define a randomized linear map $T : \mathbb{R}^n \to \mathbb{R}^k$ by

$$T(x)_i = \frac{1}{\sqrt{k}}\sum_{j=1}^{n} Z_{ij}x_j \quad \text{for } i \in [k] \quad ,$$

where $\{Z_{ij}\}_{i\in[k],j\in[n]} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$. Then for all $x \in \mathbb{R}^n$,

$$\mathsf{P}[(1-\varepsilon)\|x\|_2 \leq \|T(x)\|_2 \leq (1+\varepsilon)\|x\|_2] \geq 1-\delta.$$

Proof. Let $Y_i = \sum_{j=1}^n Z_{ij}x_j$. Then $\|T(x)\|_2^2 = \frac{1}{k}\sum_{i=1}^k Y_i^2$. Then $Y_i \sim \mathcal{N}\left(0, \|x\|_2^2\right)$. Homogenizing does not change the conclusion of the lemma, so we may w.l.o.g. choose $\|x\|_2 = 1$, and thus $\{Y_i\}_{i\in[k]} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$. Thus $\mathsf{E}\left[\|T(x)\|_2^2\right] = 1$. It remains to prove that $\|T(x)\|_2^2$ is concentrated around 1.

We have

$$\mathsf{Var}\left(\|T(x)\|_2^2\right) = \frac{1}{k^2}\sum_{i=1}^{k}\mathsf{Var}\left(Y_i^2\right) = \frac{1}{k}\mathsf{Var}\left(Y^2\right) = O(k^{-1})$$

for $Y \sim \mathcal{N}(0,1)$. Let's set $W = \frac{1}{\sqrt{k}} \sum_{i=1}^{k} \left(Y_i^2 - 1\right)$, so $\mathsf{E}[W] = 0$ and $\mathsf{Var}(W) = \Theta(1)$. By probability theory one can bound

$$\mathsf{E}\left[\mathrm{e}^{uW}\right] = \prod_{i=1}^{k} \mathsf{E}\left[\mathrm{e}^{u(Y_i^2-1)/k}\right] \leq \left(\mathrm{e}^{C(u/k)^2}\right)^k = \mathrm{e}^{Cu^2}$$

where u is bounded above by some constant $u_0/\sqrt{k}$, and similarly for $\mathsf{E}\left[\mathrm{e}^{-uW}\right]$. Then

$$\mathsf{P}[|W| \geq t] \leq 2\mathrm{e}^{-ct^2} \quad \text{for } t \in \left[0, \sqrt{k}\right].$$

From here it is simple to finish. In particular $\|T(x)\|_2^2 - 1$ for $x \in \mathrm{S}_2^{n-1}$ is distributed as $k^{-1/2}W$, so

$$\begin{aligned}
\mathsf{P}\left[1-\varepsilon \leq \|T(x)\|_2^2 \leq 1+\varepsilon\right] &= \mathsf{P}\left[(1-\varepsilon)^2 \leq \|T(x)\|_2^2 \leq (1+\varepsilon)^2\right] \\
&\geq \mathsf{P}\left[1-\varepsilon \leq \|T(x)\|_2^2 \leq 1+\varepsilon\right] \\
&= \mathsf{P}\left[|W| \leq \varepsilon\sqrt{k}\right] \\
&\geq 1 - 2\mathrm{e}^{-c\varepsilon^2 k}.
\end{aligned}$$

A more thorough analysis shows that $c = 1/8$ suffices. □

3.1.2 Oblivious Subspace Embedding

Definition 3.7 (Oblivious Subspace Embedding)
A (k, ε, δ)-oblivious subspace embedding is a map T such that, for a fixed k-dimensional vector subspace V of X, with probability at least $1-\delta$, T is an ε-almost isometry for V.

The same matrices which satisfy Johnson-Lindenstrauss also satisfy oblivious subspace embedding. In particular, if $k_{\mathrm{JL}} = O\left(\frac{k_{\mathrm{OSE}} + \log(1/\delta)}{\varepsilon^2}\right)$ then

$$\texttt{OSE}(k_{\mathrm{OSE}}, \varepsilon, \delta) = \texttt{JohnsonLindenstrauss}(k_{\mathrm{JL}}, \varepsilon, \delta)\,.$$

3.2 Sparse Dimensionality Reduction

Before now, we have been talking about dimensionality reduction that preserves lengths or distances. Now, we will discuss dimensionality reduction that enables us to find sparse vectors that are mapped under the reduction to a target vector.

It is useful to have a definition of sparsity.

Definition 3.8 (Sparsity)
A vector $x \in \mathbb{R}^n$ is *r-sparse* if $\|x\|_0 \leq r$.

Specifically, let $A \in \mathbb{R}^{k \times n}$, where $k \ll n$. We want to find r-sparse solutions to the linear system $Ax = b$, and moreover we want k small.

Proposition 3.9
If every $2r$ rows of A are linearly independent, then there is at most one r-sparse solution $\widetilde{x}$ to $Ax = b$.

Computing this sparse solution is NP-hard in general. However, methods have been invented that find the sparse solution efficiently for a wide class of matrices. Roughly speaking, while the condition above for uniqueness of a sparse solution requires every $2r$ columns of A to be linearly independent, a sufficient condition for efficient computability of the sparse solution is that every $3r$ columns of A are nearly orthogonal. In other words, the linear mapping $\mathbb{R}^{3r} \to \mathbb{R}^k$ defined by these columns should be a Euclidean ε_0-almost isometry for some small constant ε_0.

As we will prove, for a matrix A satisfying the condition just stated, a sparse solution x can be found as a solution to the following minimization problem:

$$\begin{aligned} &\underset{x \in \mathbb{R}^n}{\text{minimize}} \ \|x\|_1 \\ &\text{subject to} Ax = b. \end{aligned} \tag{BP}$$

Instead of looking for a solution x with the smallest number of nonzero components, we look for a solution with the smallest ℓ_1 norm. This method of searching for sparse solutions is called the *basis pursuit* in the literature.

Definition 3.10 (BP-Exactness)
Let us call the matrix A *BP-exact* for sparsity r if for all b such that $Ax = b$ has an r-sparse solution $\widetilde{x}$, the problem (BP) has $\widetilde{x}$ as the unique minimum.

The set of all r-sparse vectors is a union of r-dimensional coordinate subspaces. We will consider only r-sparse $\widetilde{x}$ with $\|\widetilde{x}\|_1 = 1$ (without loss of generality, since we can re-scale the right-hand side b of the considered linear system). These vectors constitute exactly the union of all $(r-1)$-dimensional faces of the unit ℓ_1 ball B_1^n.

Let A be a $k \times n$ matrix of rank k. Then $\dim(\ker(A)) = n - k = m$. A given r-sparse vector $\widetilde{x}$ satisfies the linear system $Ax = b_{\widetilde{x}}$, where $b_{\widetilde{x}} = A\widetilde{x}$, and the set of all solutions of this system is a translate of $\ker(A)$, namely $\ker(A) + \widetilde{x}$.

When the affine subspace $\ker(A) + \widetilde{x}$ touches B_1^n at $\widetilde{x}$, then $\widetilde{x}$ uniquely minimizes the ℓ_1 norm among all points of $\ker(A) + \widetilde{x}$. Then $\ker(A) + \widetilde{x}$ touches B_1^n at $\widetilde{x}$ exactly if

$$\Pi^{-1}_{\ker(A)^\perp}\left(\Pi_{\ker(A)^\perp}(\widetilde{x})\right) = \{\widetilde{x}\}.$$

In particular, $\Pi_{\ker(A)^\perp}(\widetilde{x})$ has to lie on the boundary of the projected ℓ_1 ball.

Thus BP exactness of A can be re-phrased as follows. Every point $\widetilde{x}$ in each $(r-1)$ face of the unit ℓ_1 ball B_1^n should project to the boundary of $\Pi_{\ker(A)^\perp}\left(\mathrm{B}_1^n\right)$, and should have a unique preimage in the projection.

Theorem 3.11 (BP-Exactness of Random Matrices)
There are constants C and $c > 0$ such that if n, k, r are integers with $1 \le r \le n/C$ and $k \ge Cr\log\left(\frac{n}{r}\right)$, and if A is a random $k \times n$ matrix (with entries i.i.d. $\mathrm{Bern}(\pm 1)$ or $\mathcal{N}(0,1)$), then A is BP-exact for sparsity r with probability at least $1 - \mathrm{e}^{-ck}$.

It is known that the theorem is asymptotically optimal in the following sense. For $k = o\left(r\log\left(\frac{n}{r}\right)\right)$, *no* $k \times n$ *matrix at all* can be BP-exact for sparsity r.

Definition 3.12 (2-RIP)
A matrix A has the property *r-restricted ℓ_2/ℓ_2 ε-almost isometry* if the corresponding linear mapping satisfies the condition of ε-almost isometry with respect to the ℓ_2 norm for every sparse x; that is, if

$$(1-\varepsilon)\|x\|_2 \le \|Ax\|_2 \le (1+\varepsilon)\|x\|_2 \quad \text{for all } r\text{-sparse } x \in \mathbb{R}^n.$$

Lemma 3.13
There is a constant $\varepsilon_0 > 0$ such that if a matrix A has the property of $3r$-restricted ℓ_2/ℓ_2 ε_0-almost isometry, then it is BP-exact for sparsity r.

Remark 3.14. The same proof works for restricted ℓ_2/ℓ_1 ε_0-almost isometry, i.e., assuming $(1-\varepsilon)\|x\|_2 \le \|Ax\|_1 \le (1+\varepsilon)\|x\|_2$ for all $3r$-sparse x.

Proof of BP-Exactness of Random Matrices Result Assuming RIP Result. Let B be a matrix consisting of some $3r$ distinct columns of A. Then the linear mapping $\mathbb{R}^{3r} \to \mathbb{R}^k$ given by an appropriately scaled B fails to be an ε_0-almost isometry with probability at most $\mathrm{e}^{-c_1\varepsilon_0^2 k}$ for some positive constant c_1.

The number of possibilities of B is

$$\binom{n}{3r} \le \left(\frac{\mathrm{e}n}{3r}\right)^{3r} \le \left(\frac{n}{r}\right)^{3r} = \mathrm{e}^{3r\log(n/r)}.$$

Thus A fails to have the $3r$-restricted ε_0-isometry property with probability at most $\mathrm{e}^{3r\log(n/r)}\mathrm{e}^{-c_1\varepsilon_0^2 k} \le \mathrm{e}^{-ck}$ for r, k, n as in the theorem. □

Proof of RIP Result. Suppose A has the property of $3r$-restricted ℓ_2/ℓ_2 ε_0-almost isometry, and let $\widetilde{x}$ is an r-sparse solution of $Ax = b$ for some b.

For contradiction, we assume that $\widetilde{x}$ is not the unique minimum of (BP), and so there is another solution of $Ax = b$ with smaller or equal ℓ_1 norm. Write this solution as $\widetilde{x} + \Delta$, so

$$A\Delta = 0 \qquad \|\widetilde{x} + \Delta\|_1 \le \|\widetilde{x}\|_1 .$$

Assume $\Delta \neq 0$. We aim to reach a contradiction.

Note that if A were an almost-isometry, then $\Delta \neq 0$ would imply $A\Delta \neq 0$ and we would have a contradiction immediately. We cannot expect the whole of A to be an almost-isometry – we have control only over small blocks of A.

First let $S = \{i : \widetilde{x}_i \neq 0\}$. We observe that

$$\|\Delta_S\|_1 \ge \left\|\Delta_{[n]\setminus S}\right\|_1 ,$$

where Δ_S denotes the vector consisting of the components of Δ indexed by S. When Δ is added to $\widetilde{x}$, its components outside S only increase the ℓ_1 norm, and since $\|\widetilde{x} + \Delta\|_1 \le \|\widetilde{x}\|_1$, the components in S must at least compensate for this increase.

Since the r-restricted ℓ_2/ℓ_2 ε-isometry property of A concerns the ℓ_2 norm, we will need to argue about the ℓ_2 norm of various pieces of Δ. For simpler notation, assume $\|\Delta\|_1 = 1$ (the argument is scale-invariant). Then $\|\Delta_S\|_1 \ge \frac{1}{2}$ and thus $\|\Delta_S\|_2 \ge \frac{1}{2\sqrt{r}}$ by Cauchy-Schwarz.

Let $B_0 \subseteq [n] \setminus S$ consist of the indices of the $2r$ largest components of $\Delta_{[n]\setminus S}$, B_1 are the indices of the next $2r$ largest components, and so on (the last block may be smaller). We have

$$\left\|A_{S\cup B_0}\Delta_{S\cup B_0}\right\|_2 \ge (1-\varepsilon_0)\left\|\Delta_{S\cup B_0}\right\|_2 \ge (1-\varepsilon_0)\|\Delta_S\|_2 \ge \frac{1-\varepsilon_0}{2\sqrt{r}}.$$

We know that

$$\sum_{j\ge 0}\left\|\Delta_{B_j}\right\|_1 = \left\|\Delta_{[n]\setminus S}\right\|_1 \le \frac{1}{2}.$$

Moreover, by the choice of the blocks, the components belonging to B_j are no larger than the average of those in

B_{j-1}, and thus

$$\left\|\Delta_{B_j}\right\|_2 \le \left(2r\left(\frac{\left\|\Delta_{B_{j-1}}\right\|_1}{2r}\right)^2\right)^{1/2} = \frac{\left\|\Delta_{B_j}\right\|_1}{\sqrt{2r}}.$$

Summing over $j \ge 1$, we have

$$\sum_{j\ge 1}\left\|\Delta_{B_j}\right\|_2 \le \frac{1}{\sqrt{2r}}\sum_{j\ge 0}\left\|\Delta_{B_j}\right\|_1 \le \frac{1}{2\sqrt{2r}}.$$

Then using r-restricted ℓ_2/ℓ_2 ε_0-almost isometry on $S \cup B_0$ and on each of $B_1, B_2, \ldots,$

$$\|A\Delta\|_2 \ge \left\|A_{S\cup B_0}\Delta_{S\cup B_0}\right\|_2 - \sum_{j\ge 1}\left\|A_{B_j}\Delta_{B_j}\right\|_2 \ge \frac{1-\varepsilon_0}{2\sqrt{r}} - \frac{1+\varepsilon_0}{2\sqrt{2r}} > 0$$

providing the contradiction for a suitable ε_0. □

3.3 Nearest Neighbors

We are given a set P of n points in (X, d_X). We will mostly consider $\mathbb{R}^k$. Given a query point $x \in X$, we want to find $p \in P$ that is the closest to x.

The main idea is that we first *preprocess* the s et P and store the results in a data structure, and then use this data structure to answer the queries faster.

Definition 3.15 (C-Approximate Nearest Neighbor)
Let $C \ge 1$. A C-approximate algorithm for the nearest neighbor problem is one that returns $p \in P$ such that

$$d(p, x) \le C \inf_{q\in P} d_X(x, q).$$

This is very much in the spirit of approximate metric embeddings, with C being similar to the distortion.

Definition 3.16 (C-Approximate r-Near Neighbor Problem)
Let $C > 1$. The *C-approximate r-near neighbor problem* asks us to do the following. Given an input point set P and a number $r > 0$, and a query point x, the algorithm should return either a point $p \in P$ at most distance Cr to x, or the answer `NONE` if the distance of x to all points of P exceeds r.

It is known that an algorithm for the C-approximate r-near neighbor problem can be transformed into an algorithm for the $(1+\eta)C$-approximate nearest neighbor problem. Such a transformation is very simple unless the ratio $\Delta = d_{\max}/d_{\min}$ is very large, where $d_{\max} = \sup_{p,q\in P} d_X(p, q)$ and $d_{\min} = \inf_{p,q\in P:\, p\ne q} d_X(p, q)$. The idea is to build several data structures for the C-approximate r-near neighbor problem for suitably chose values of r, and to query all of them. One can take $d_{\min}/((1+\eta)C)$ as the smallest value of r and then keep doubling it until it exceeds $d_{\max}$. This incurs an extra factor of at most $O(\log(\Delta))$ both in the storage and query time (here C is considered fixed).

The preprocessing phase of the algorithm is going to be randomized (while query answering is deterministic). Moreover, the algorithm is allowed to fail (i.e., say `NONE` even when it should really return a point) with some small probability δ. The failure probability is taken with respect to the internal random choices made during the preprocessing. If we fix P and a query point x, then the probability that the algorithm fails for x is at most δ.

In the algorithm given below, we will bound the failure probability by $\frac{3}{4}$. By building and querying t independent data structures, the failure probability can be reduced to $\left(\frac{3}{4}\right)^t$, and thus we can make it as small as desired, while paying a reasonable price in storage and query time.

The algorithm we will consider is based on *locality-sensitive hashing*. We assume that the points live in a metric space (X, d_X), and we want that *two close points are more likely to be hashed to the same bucket than two far-away points*. Let J be an index set. We consider a family $\mathcal{H}$ of hash functions $h\colon X \to J$, together with a probability distribution on $\mathcal{H}$.

Definition 3.17 (Locality-Sensitive Hash Family)
A family $\mathcal{H}$ of hash functions from X to J is called $(r, Cr, p_{close}, p_{far})$-*sensitive*, where $r > 0$, $C \geq 1$, and $0 \leq p_{\text{far}} < p_{\text{close}} \leq 1$, if for every two points $x, y \in X$ we have

(i) If $d_X(x, y) \leq r$, then $\mathsf{P}_h[h(x) = h(y)] \geq p_{\text{close}}$.

(ii) If $d_X(x, y) > Cr$, then $\mathsf{P}_h[h(x) = h(y)] \leq p_{\text{far}}$.

Assume that a $(r, Cr, p_{\text{close}}, p_{\text{far}})$-sensitive hash family $\mathcal{H}$ is available for some constants $p_{\text{close}}, p_{\text{far}}$. Define

$$\alpha = \frac{\log(p_{\text{close}})}{\log(p_{\text{far}})}.$$

We will achieve query time roughly $O(n^{\alpha})$ and storage roughly $O\left(n^{1+\alpha}\right)$.

First we need to amplify the gap between p_{close} and p_{far}. We define a new family $\mathcal{G}$, consisting of all t-tuples of hash functions from $\mathcal{H}$, where t is a parameter to set later. That is,

$$\mathcal{G} = \left\{g = (h_1, \ldots, h_t)\colon X \to J^t, h_1, \ldots, h_t \in \mathcal{H}\right\},$$

and for choosing $g \in \mathcal{G}$ at random, we pick $h_1, \ldots, h_t \in \mathcal{H}$ randomly and independently. Then, clearly, $\mathcal{G}$ is $(r, Cr, p_{\text{close}}^t, p_{\text{far}}^t)$-sensitive.

In the preprocessing phase of the algorithm, we choose L random hash functions $g_1, \ldots, g_L \in \mathcal{G}$, where L is another parameter to be determined in the future. For each $i \in [L]$, we construct a hash table storing all elements of the point set P, where each $p \in P$ is stored in the bucket $g_t(p)$ of the i^{th} hash table.

Note that the set J^t indexing the buckets in the hash tables may be very large or even infinite. We can employ ordinary hashing, and further hash the bucket indices into a compact table of size $O(n)$. Thus the total space occupied by the hash tables is $O(nL)$.

To process a query x, we consider the points stored in the bucket $g_i(x)$ of the i^{th} hash table, $i \in [L]$, one by one, for each of them we compute the distance to x, and we return the first point with distance at most Cr to x. If no such point is found in these buckets, we return the answer `NONE`. Moreover, if these L buckets together contain more than $3L$ points, we abort the search after examining $3L$ points unsuccessfully, and also return `NONE`.

Thus, for processing a query, we need at most kL evaluations of hash functions from the family $\mathcal{H}$ and at most $3L$ distance computations.

Here is a summary of the algorithm:

Algorithm 17 C-Approximate r-Near Neighbor via Locality-Sensitive Hashing

Input: A point set P.
Input: A number $r > 0$.
Output: A query function which, given a point x, solves the C-approximate r-near neighbor problem.

```
procedure PREPROCESSING(P, r)
    for i ∈ [L] do
        g_i = (h_{i,1}, ..., h_{i,t}) ~ Uni(𝒢)
        T_i ← CreateHashTable(P, g_i)
procedure QUERY(x)
    N ← 0
    for i ∈ [L] do
        for p ∈ g_i(x) do
            if N ≥ 3L then
                return NONE
            if d_X(x, p) ≤ C r then
                return p
            N ← N + 1
    return NONE
```

The algorithm fails if it answer `NONE` even though there is a point $p_0 \in P$ with $d_X(x, p_0) \le r$. This may have two reasons:

(a) In none of the L hash tables, x was hashed to the same bucket as p_0.

(b) The L buckets searched up by the algorithm contain more than $3L$ "far" points, i.e., points p with $d_X(x, p) > Cr$.

We want to set the parameters t and L so that the failure probability is bounded by a constant $\delta < 1$.

For points p_0 and x with $d_X(x, p_0) \le r$ we have $\mathsf{P}_h[g(x) = g(p_0)] \ge p_{\text{close}}{}^t$, and so the probability of type (a) failure is at most $(1 - p_{\text{close}}^t)^L$.

As for the type (b) failure, the probability that a far point q goes to the same bucket as x is at most p_{far}^t, and so the expected number of far points in the L searched buckets is no more than nLp_{far}^t. By Markov's inequality, the probability that we have more than $3L$ far points there is is at most $np_{\text{far}}^t/3$.

First we set t so that $np_{\text{far}}^t/3 \le 1/3$; this requires $t = \log(n) / \log(1/p_{\text{far}})$. Then using $(1 - p_{\text{close}}^t)^L \le \mathrm{e}^{-p_{\text{close}}^t L}$, we see that for $L = p_{\text{close}}^{-t}$, the probability of part (a) failure is at most e^{-1}, and the total failure probability doesn't exceed $\frac{1}{3} + \mathrm{e}^{-1} < \frac{3}{4}$. For the value of L this gives,

$$L = p_{\text{close}}^{-t} = \mathrm{e}^{\log(p_{\text{close}}) \log(n) / \log(p_{\text{far}})} = n^{\alpha}$$

as claimed.

This general algorithm can be used with α arbitrarily close to $1/C$. We need space $O(n^{1+\alpha})$ for storing the hash tables, $O(kn)$ space for the points themselves, and $O(ktL) = O(kn)$ space for representing the hash functions $g_1, \ldots, g_L$, so $O(kn + n^{1+\alpha})$ in total.

The cost of distance computation is $O(k)$, and evaluating a hash function g_i needs $O(tk) = O(k \log(n))$. The total query time is $O(kn^{\alpha} \log(n))$.

3.4 Tree Embeddings

Fix an undirected graph $G = (V, E)$ with distance d_G denoting shortest path in G. We hope

- Graphs have spanning trees that preserve their distances. That is, there exists a subtree $T = (V, E_T)$ with distance d_T and $E_T \subseteq E$ such that

$$d_G(u, v) \approx d_T(u, v) \quad \text{for all } u, v \in V.$$

- We can solve NP-hard problems on trees.

Definition 3.18 (Low-Diameter Decomposition)
A *low-diameter decomposition* with parameter β for a finite metric space (V, d_V) is a randomized algorithm that, given a bound $\Delta > 0$, partitions the point set V into $\bigcup_i P_i$ such that, if $P(v)$ be the unique cluster that contains v.

(i) For $i \in [t]$, $\operatorname{diam}(P_i) \leq \Delta$ (Δ-bounded), and

(ii)

$$\mathsf{P}[P(x) \neq P(y)] \leq \beta \cdot \frac{d_V(x, y)}{\Delta} \quad \text{for all } x, y \in V \text{ such that } x \neq y.$$

Here is an algorithm that gives a random Δ-bounded partition.

Algorithm 18 Low-Diameter Decomposition.

Input: A finite metric space (V, d_V).
Output: A Δ-bounded partition of V.

$R \sim \operatorname{Uni}([\Delta/4, \Delta/2])$
$V_O = \{v_i\}_{i \in [n]} \leftarrow$ random ordering of V.
for $i \in [n]$ **do**
    $V_i \leftarrow \mathbb{B}(x_i, R) \setminus \bigcup_{j<i} B(x_j, R)$
return $\{V_i\}_{i \in [n]}$

Lemma 3.19 (Random Partition Lemma)
For every $\Delta > 0$, the Δ-bounded partition P returned by the low-diameter decomposition has the property such that for every $r \leq \Delta/8$,

$$\mathsf{P}[\mathbb{B}(x, r) \subseteq P(x)] \geq \exp\left(-\frac{8r}{\Delta} \log\left(\frac{|\mathbb{B}(x, \Delta)|}{|\mathbb{B}(x, \Delta/8)|}\right)\right).$$

Proof. Fix $r \leq \Delta/8$ and observe that

$$\mathsf{P}[\mathbb{B}(x, r) \subseteq P(x) \mid R] \geq \frac{|\mathbb{B}(x, r - r)|}{|\mathbb{B}(x, R + r)|}.$$

This follows because if we condition on R, then only centers in $\mathbb{B}(x, R + r)$ can decide the fate of $\mathbb{B}(x, r)$, and the corresponding cluster will contain all of $\mathbb{B}(x, r)$ if the center lies in $\mathbb{B}(x, R - r)$. Thus

$$\begin{aligned}
\mathop{\mathsf{P}}_R[\mathbb{B}(x, r) \subseteq P(x)] &\geq \mathop{\mathsf{E}}_R\left[\frac{|\mathbb{B}(x, R - r)|}{|\mathbb{B}(x, R + r)|}\right] \\
&= \mathop{\mathsf{E}}_R\left[\exp\left(-\log\left(\frac{|\mathbb{B}(x, R + r)|}{|\mathbb{B}(x, R - r)|}\right)\right)\right] \\
&\geq \exp\left(\mathop{\mathsf{E}}_R\left[-\log\left(\frac{|\mathbb{B}(x, R + r)|}{|\mathbb{B}(x, R - r)|}\right)\right]\right)
\end{aligned}$$

$$
\begin{aligned}
&= \exp\left(-\mathop{\mathsf{E}}_{R}\left[\log\left(\frac{|\mathbb{B}(x,R+r)|}{|\mathbb{B}(x,R-r)|}\right)\right]\right)\\
&= \exp\left(-\frac{4}{\Delta}\int_{\Delta/4}^{\Delta/2}\log\left(\frac{|\mathbb{B}(x,R+r)|}{|\mathbb{B}(x,R-r)|}\right)\mathrm{d}R\right)\\
&\geq \exp\left(-\frac{8r}{\Delta}\log\left(\frac{\left|\mathbb{B}\left(x,\frac{\Delta}{2}+r\right)\right|}{\left|\mathbb{B}\left(x,\frac{\Delta}{4}-r\right)\right|}\right)\right)\\
&\geq \exp\left(-\frac{8r}{\Delta}\log\left(\frac{|\mathbb{B}(x,\Delta)|}{\left|\mathbb{B}\left(x,\frac{\Delta}{8}\right)\right|}\right)\right).
\end{aligned}
$$

□

Remark 3.20. If $d_V(x,y) \leq r$, then

$$\mathsf{P}[P(x) \neq P(y)] \geq \exp\left(-\frac{8r}{\Delta}\log\left(\frac{|\mathbb{B}(x,\Delta)|}{\left|\mathbb{B}\left(x,\frac{\Delta}{8}\right)\right|}\right)\right).$$

Consider a partition P of V such that P_j is 8^j-bounded for every j. Define the metric

$$D_P(x,y) = \max\left\{8^{j+1} : P_j(x) \neq P_j(y)\right\}.$$

Then D_P is a metric on V, and moreover

$$D_P(x,y) \geq d_V(x,y) \quad \text{for all } x,y \in V.$$

This follows from $D_P(x,y) \leq 8^j$ implies $P_j(x) = P_j(y)$ implies $d_V(x,y) \leq 8^j$.

Fix $x \neq y \in V$. Let $j_0 = \min\left\{j : 8^j \geq d_V(x,y)\right\}$ denote the LCA of x, y. Then

$$\mathop{\mathsf{E}}_{R}[D_P(x,y)] \leq 8^{j_0+1} + \sum_{j>j_0}\mathsf{P}\left[P_j(x) \neq P_j(y)\right]8^{j+1},$$

and using

$$\mathsf{P}\left[P_j(x) = P_j(y)\right] \geq \mathsf{P}\left[\mathbb{B}(x, d_V(x,y)) \subseteq P_j(x)\right]$$

yields

$$
\begin{aligned}
\sum_{j>j_0}\mathsf{P}\left[P_j(x) \neq P_j(y)\right]8^{j+1} &\leq \sum_{j\geq 1} 8^{j+2}\frac{d_V(x,y)}{8^j}\log\left(\frac{\left|\mathbb{B}\left(x,8^j\right)\right|}{\left|\mathbb{B}\left(x,8^{j-1}\right)\right|}\right)\\
&\leq 64 d_V(x,y)\sum_{j\geq 1}\log\left(\frac{\left|\mathbb{B}\left(x,8^j\right)\right|}{\left|\mathbb{B}\left(x,8^{j-1}\right)\right|}\right)\\
&= 64 d_V(x,y)\log(n).
\end{aligned}
$$

This induces a hierarchical tree decomposition.

3.5 Graph Embeddings and Applications

4 Spectral Graph Theory

The idea of this chapter is to discuss graphs via the spectrum of their associated matrices. Throughout this chapter, fix a weighted graph $G = (V, E, d)$.

Definition 4.1 (Rayleigh Quotient)
The Rayleigh quotient of a vector $x \neq 0$ w.r.t. a square matrix M is

$$\frac{x^* M x}{x^* x} = \frac{\langle Mx, x\rangle}{\langle x, x\rangle}.$$

Remark 4.2. If $Mv = \lambda v$, then

$$\frac{\langle Mv, v\rangle}{\langle v, v\rangle} = \frac{\langle \lambda v, v\rangle}{\langle v, v\rangle} = \lambda \frac{\langle v, v\rangle}{\langle v, v\rangle} = \lambda.$$

The Rayleigh quotient of an eigenvector is hence its eigenvalue.

Theorem 4.3 (Courant-Fischer)
Let $M \in \mathrm{Sym}(n)$ with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Then

$$\lambda_k = \sup_{\substack{S \leq \mathbb{R}^n \\ \dim(S) = k}} \inf_{\substack{x \in S \\ x \neq 0}} \frac{\langle Mx, x\rangle}{\langle x, x\rangle} = \inf_{\substack{T \leq \mathbb{R}^n \\ \dim(T) = n-k+1}} \sup_{\substack{x \in T \\ x \neq 0}} \frac{\langle Mx, x\rangle}{\langle x, x\rangle}.$$

Remark 4.4. The notation $S \leq \mathbb{R}^n$ means S is a vector subspace of $\mathbb{R}^n$, and similarly for T.

Proof of Courant-Fischer Theorem. We will verify the first assertion. The second is similar. Fix $M \in \mathrm{Sym}(n)$ with eigenvalues $\lambda_1, \ldots, \lambda_n$ and corresponding orthonormal eigenvectors $v_1, \ldots, v_n$. Fix $S = \mathrm{span}(v_1, \ldots, v_n)$ and$x \in S \setminus \{0\}$. Write

$$x = \sum_{i=1}^{k} c_i v_i.$$

Then

$$\frac{\langle Mx, x\rangle}{\langle x, x\rangle} = \frac{\left\langle M \sum_{i=1}^k c_i v_i, \sum_{i=1}^k c_i v_i \right\rangle}{\left\langle \sum_{i=1}^k c_i v_i, \sum_{i=1}^k c_i v_i \right\rangle} = \frac{\left\langle \sum_{i=1}^k c_i \lambda_i v_i, \sum_{i=1}^k c_i v_i \right\rangle}{\left\langle \sum_{i=1}^k c_i v_i, \sum_{i=1}^k c_i v_i \right\rangle} = \frac{\sum_{i=1}^k \lambda_i c_i^2}{\sum_{i=1}^k c_i^2} \geq \frac{\sum_{i=1}^k \lambda_k c_i^2}{\sum_{i=1}^k c_i^2} = \lambda_k.$$

So

$$\inf_{\substack{x \in S \\ x \neq 0}} \frac{\langle Mx, x\rangle}{\langle x, x\rangle} \geq \lambda_k.$$

We want to show the other required bound for all S s.t. $\dim(S) = k$, that is, $\inf_{x \in S \setminus \{0\}} \frac{\langle Mx, x\rangle}{\langle x, x\rangle} \leq \lambda_k$. Fix such an (arbitrary) S. Let $T = \mathrm{span}(v_k, \ldots, v_n)$. Then $\dim(T) = n - k + 1$, so $\dim(S \cap T) \geq 1$. So

$$\inf_{\substack{x \in S \\ x \neq 0}} \frac{\langle Mx, x\rangle}{\langle x, x\rangle} \leq \inf_{\substack{x \in S \cap T \\ x \neq 0}} \frac{\langle Mx, x\rangle}{\langle x, x\rangle} \leq \sup_{\substack{x \in T \\ x \neq 0}} \frac{\langle Mx, x\rangle}{\langle x, x\rangle}.$$

Since $x \in T \setminus \{0\}$, write

$$x = \sum_{i=k}^{n} c_i v_i.$$

Then

$$\frac{\langle Mx, x\rangle}{\langle x, x\rangle} = \frac{\sum_{i=k}^{n} \lambda_i c_i^2}{\sum_{i=k}^{n} c_i} \leq \frac{\sum_{i=k}^{n} \lambda_k c_i^2}{\sum_{i=k}^{n} c_i^2} = \lambda_k.$$

Thus

$$\inf_{\substack{x \in S \\ x \neq 0}} \frac{\langle Mx, x\rangle}{\langle x, x\rangle} \leq \lambda_k$$

and the claim is proved. □

4.1 The Laplacian

Notation 4.5

Let's re-order the eigenvalues, so that given a list of eigenvalues and eigenvectors, we order them so that

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

Definition 4.6 (Laplacian)

The *Laplacian* L_G is defined by the quadratic form

$$x^* L_G x = \langle L_G x, x\rangle = \sum_{(i,j)\in E} d(i,j)\left(x_i - x_j\right)^2.$$

It can be expressed in the form

$$L_G = D_G - A_G$$

where the *degree matrix* D_G is given by

$$(D_G)_{ij} = \begin{cases} \sum_{k \in V} d(i,k) & i = j \\ 0 & i \neq j \end{cases}$$

and the *adjacency matrix* A_G is given by

$$(A_G)_{ij} = d(i,j) \quad \text{for all } i, j \in V.$$

Remark 4.7. The Laplacian $L_G \in \mathrm{PSD}(n)$.

Notation 4.8

Just as we have fixed a graph G, we fix its Laplacian L_G. Let $\lambda_1 \leq \cdots \leq \lambda_{|V|}$ be the eigenvalues of L_G, with associated orthonormal eigenvectors $v_1, \ldots, v_{|V|}$. This is justified by the spectral theorem for real symmetric matrices.

Remark 4.9. $\lambda_1 = 0$, with eigenvector 1_V.

Theorem 4.10

$\lambda_k = 0$ if and only if L_G has $\geq k$ connected components.

Proof. We first show that if G has $\geq k$ connected components then $\lambda_k = 0$. Write $G = \bigcup_{i=1}^{k} G_i$ where $G_i = (V_i, E_i)$ are disjoint and may be connected. Then L_G has at least k eigenvectors with eigenvalue 0, namely 1_{V_i}. Thus $\lambda_1 = \cdots = \lambda_k = 0$.

On the other hand, suppose that $\lambda_k = 0$. Let v_k be its normalized eigenvector. Then

$$0 = \langle L_G v_k, v_k \rangle = \sum_{(i,j)\in E} d(i,j) \left(v_{k,i} - v_{k,j}\right)^2$$
$$\implies v_{k,i} = v_{k,j} \quad \text{for all } (i,j) \in E.$$

Thus v_k is a indicator function for unions of connected components. Since $\{v_i\}_{i\in V}$ are orthogonal, $v_1, \ldots, v_k$ are indicator vectors for (disjoint) connected components. Thus G has $\geq k$ connected components. □

4.2 Graph Projections

Given a Laplacian L_G (which contains distance and edge connectivity information), we want to find an embedding of G into $\mathbb{R}^k$. In the case $k = 1$, we solve the problem

$$x_* = \operatorname*{minimize}_{x\in\mathbb{R}^{|V|}} \langle L_G x, x \rangle .$$

The solution to this is $x^* = 0_{|V|}$, which is a denegerate embedding. To fix this we mandate that x is normalized:

$$x_* = \operatorname*{minimize}_{\substack{x\in\mathbb{R}^{|V|} \\ \|x\|_2^2 = 1}} \langle L_G x, x \rangle .$$

This also results in a degenerate solution, $x_* = \frac{1_{|V|}}{\sqrt{n}}$. We could impose the additional constraint

$$x_* = \operatorname*{minimize}_{\substack{x\in\mathbb{R}^{|V|} \\ \|x\|_2^2 = 1 \\ \langle 1_{|V|}, x\rangle = 0}} \langle L_G x, x \rangle = v_2(L_G) .$$

This gives a one-dimensional embedding, that is, if i is a vertex in G then the embedding is $\phi\colon i \mapsto x_{*,i}$. To get it for arbitrary k, we solve the problem

$$x_* = \operatorname*{minimize}_{\substack{x\in\mathbb{R}^{k\times|V|} \\ \|x_i\|_2^2 = 1 \\ \langle 1_{|V|}, x_i\rangle = 0 \ \forall i \\ \langle x_i, x_j\rangle = 0 \ \forall i<j}} \sum_{(i,j)\in E} \left\| x_{:,i} - x_{:,j} \right\|_2^2 = \operatorname*{minimize}_{\substack{x\in\mathbb{R}^{k\times|V|} \\ \|x_i\|_2^2 = 1 \\ \langle 1_{|V|}, x_i\rangle = 0 \ \forall i \\ \langle x_i, x_j\rangle = 0 \ \forall i<j}} \sum_{i=1}^{k} \langle L_G x_i, x_i \rangle = \begin{bmatrix} v_2^* \\ \vdots \\ v_{k+1}^* \end{bmatrix}.$$

Thus the eigenvectors form the embedding up to normalization constants.

4.3 Graph Partitioning

Definition 4.11 (Isoperimetric Ratio)
Let $S \subseteq V$. The *boundary* of S is

$$\partial S = \{(i,j) \in E : i \in S, j \notin S\}.$$

The *isoperimetric ratio of* S is

$$\theta(S) = \frac{|\partial(S)|}{|S|}.$$

The *isoperimetric ratio of* G is

$$\theta_G = \inf_{\substack{S \subseteq V \\ |S| \leq |V|/2}} \theta(S).$$

We now derive a lower bound for θ_G in terms of $\lambda_2(L_G)$.

Theorem 4.12
For every $S \subseteq V$, if $s = \frac{|S|}{|V|}$ then

$$\theta(S) \geq \lambda_2 (1 - s).$$

In particular

$$\theta_G \geq \frac{\lambda_2}{2}.$$

Proof. As

$$\lambda_2 = \inf_{\substack{x \in \mathbb{R}^V \\ \langle 1_V, x \rangle = 0}} \frac{\langle L_G x, x \rangle}{\langle x, x \rangle}$$

we know for every non-zero x orthogonal to 1_V that

$$\langle L_G x, x \rangle \geq \lambda_2 \langle x, x \rangle.$$

Take $x = 1_S$, so that

$$\langle L_G 1_S, 1_S \rangle = \sum_{(i,j) \in E} \left(1_{S,i} - 1_{S,j}\right)^2 = |\partial(S)|.$$

However, 1_S is not orthogonal to 1. Trying $x = 1_S - s1_V$, we have

$$\langle 1_V, x \rangle = \langle 1_V, 1_S - s1_V \rangle = 0,$$

and

$$\langle L_G x, x \rangle = \langle L_G(1_S - s1_V), 1_S - s1_V \rangle = \sum_{(i,j) \in E} \left(\left(1_{S,i} - s\right) - \left(1_{S,j} - s\right)\right)^2 = \sum_{(i,j) \in E} \left(1_{S,i} - 1_{S,j}\right)^2 = |\partial(S)|.$$

Also

$$\langle x, x \rangle = \langle 1_S - s1_V, 1_S - s1_V \rangle = n\left(s - s^2\right).$$

Thus

$$\lambda_2 \leq \frac{\langle L_G 1_S, 1_S \rangle}{\langle 1_S, 1_S \rangle} = \frac{|\partial(S)|}{|S|(1 - s)}.$$

Re-arranging terms slightly,

$$\theta(S) = |V| \frac{|\partial(S)|}{|S||V \setminus S|} \geq \lambda_2.$$

□

This theorem says that if λ_2 is big, then G is very well-connected. The boundary of every small set of vertices is at least λ_2 times something slightly smaller than $|S|$.

We want to normalize this formula for weighted graphs. Instead of counting the edges on the boundary, we count the sum of their weights. Write

$$\deg(S) = \sum_{v \in S} \deg(v) \quad \text{for all } S \subseteq V \qquad \text{and} \qquad d(T) = \sum_{e \in T} d(e) \quad \text{for all } T \subseteq E$$

Thus $\deg(V) = 2d(E)$.

Definition 4.13 (Conductance)
The *conductance* of $S \subseteq V$ is

$$\phi(S) = \frac{d(\partial(S))}{\min\{\deg(S), \deg(V \setminus S)\}}$$

The conductance of G is

$$\phi_G = \inf_{S \subseteq V} \phi(S)$$

Usually conductance is more useful when discussing edges; isoperimetric ratio is more useful when discussing vertices.

It seems natural to relate the conductance to the following generalized Rayleigh quotient:

$$\frac{\langle L_G y, y\rangle}{\langle D_G y, y\rangle}.$$

Making the substitution $D_G^{1/2} y = x$, this quotient becomes

$$\frac{\left\langle L_G D_G^{-1/2} x, D_G^{-1/2} x\right\rangle}{\left\langle D_G^{-1/2} D_G x, D_G^{-1/2} x\right\rangle} = \frac{\left\langle D_G^{-1/2} L_G D_G^{-1/2} x, x\right\rangle}{\left\langle D_G^{-1} D_G x, x\right\rangle} = \frac{\left\langle D_G^{-1/2} L_G D_G^{-1/2} x, x\right\rangle}{\langle x, x\rangle}.$$

This matrix quantity is called the *normalized Laplacian*

$$N_G = D_G^{-1/2} L_G D_G^{-1/2}.$$

We let $0 = \nu_1 \leq \cdots \leq \nu_{|V|}$ denote the eigenvalues of N_G, with corresponding orthonormal eigenvectors $w_1, \ldots, w_n$.

The eigenvector corresponding to $0 = \nu_1$ of N_G is $x = \text{diagonal}\left(D_G^{1/2}\right)$. Then

$$N_G x = D_G^{-1/2} L_G D_G^{-1/2} x = D_G^{-1/2} L_G 1_V = D_G^{-1/2} 0_V = 0_V.$$

Then

$$\nu_2 = \inf_{\substack{x \in \mathbb{R}^V \\ \left\langle x, \text{diag}\left(D_G^{1/2}\right)\right\rangle = 0}} \frac{\langle N_G x, x\rangle}{\langle x, x\rangle}.$$

Going back into the variable y and observing that

$$\left\langle x, \text{diag}\left(D_G^{1/2}\right)\right\rangle = \left\langle D_G^{1/2} y, \text{diag}\left(D_G^{1/2}\right)\right\rangle = \left\langle y, \text{diag}(D_G)\right\rangle,$$

we obtain

$$\nu_2 = \inf_{\substack{y \in \mathbb{R}^V \\ \langle y, \text{diag}(D_G)\rangle = 0}} \frac{\langle L_G y, y\rangle}{\langle D_G y, y\rangle}.$$

The conductance is related to ν_2 as the isoperimetric number is related to λ_2:

$$\frac{\nu_2}{2} \leq \phi_G.$$

Lemma 4.14
For every $S \subseteq V$,

$$\frac{d(\partial(S)) \deg(V)}{\deg(S) \deg(V \setminus S)} \geq \nu_2.$$

Proof. We would like to use 1_S as a test vector. But $\langle \operatorname{diag}(D_G), 1_S \rangle \neq 0$. To fix this, we do the same idea as a fix. Let

$$y = 1_S - \sigma 1_V \quad \text{where} \quad \sigma = \frac{\deg(S)}{\deg(V)}.$$

We need to check that $\langle y, \operatorname{diag}(D) \rangle = 0$:

$$\left\langle y, \operatorname{diag}(D) \right\rangle = \left\langle 1_S, \operatorname{diag}(D) - \sigma 1_V \right\rangle = \deg(S) - \frac{\deg(S)}{\deg(V)} \deg(V) = 0.$$

We already know that

$$\langle L_G y, y \rangle = |\partial(S)|.$$

It remains to compute $\langle D_G y, y \rangle$. In particular

$$\begin{aligned} \langle Dy, y \rangle &= \sum_{u \in S} \deg(u) (1-\sigma)^2 + \sum_{u \notin S} \deg(u) \sigma^2 \\ &= \deg(S) (1-\sigma)^2 + \deg(V \setminus S) \sigma^2 \\ &= \deg(S) - 2 \deg(S) \sigma + \deg(V) \sigma^2 \\ &= \deg(S) - \deg(S) \sigma = \frac{\deg(S) \deg(V \setminus S)}{\deg(V)}. \end{aligned}$$

So

$$\nu_2 \leq \frac{\langle L_G y, y \rangle}{\langle D_G y, y \rangle} = \frac{d(\partial(S)) \deg(V)}{\deg(S) \deg(V \setminus S)}.$$

□

Corollary 4.15
For every $S \subset V$,

$$\phi(S) \geq \frac{\nu_2}{2}.$$

Proof. As

$$\max \left\{ \deg(S), \deg(V \setminus S) \right\} \geq \frac{\deg(V)}{2},$$

we find

$$\nu_2 \leq 2 \frac{d(\partial(S))}{\min \{ \deg(S), \deg(V \setminus S) \}}.$$

□

4.4 Cheeger's Inequality

Cheeger's inequality proves that if we have a vector y, orthogonal to $\operatorname{diag}(D_G)$, for which the generalized Rayleigh quotient is small, then one can obtain a set of small conductance from y. We obtain such a set by choosing $\tau \in \mathbb{R}$, and setting

$$S_\tau = \{ i \in V : y_i \leq \tau \}.$$

We want to derive y from an eigenvector ν_2 of N_G. If w_2 is an eigenvector of ν_2, then $y = D_G^{1/2} \nu_2$ is orthogonal to $\operatorname{diag}(D_G)$ and the generalized Rayleigh quotient of y with respect to L_G and D_G equals ν_2. But the theorem can make use of any vector that is orthogonal to $\operatorname{diag}(D_G)$ that makes the generalized Rayleigh quotient small.

Definition 4.16

A vector y is centered with respect to $\operatorname{diag}(D_G)$ if

$$\sum_{\substack{i\in V\\ y_i>0}} D_{ii} \le \frac{\deg(V)}{2} \quad \text{and} \quad \sum_{\substack{i\in V\\ y_i<0}} D_{ii} \le \frac{\deg(V)}{2}.$$

By renumbering the vertices, we may assume without loss of generality that

$$y_1 \le \cdots \le y_{|V|}.$$

To center y, let

$$j = \inf\left\{k : \sum_{i=1}^{k} D_{ii} \ge \frac{\deg(V)}{2}\right\}.$$

Then we set

$$z = y - y_j 1_V.$$

Then $z_j = 0$.

Lemma 4.17

Let $v_s = y + s1_V$. Then the minimum of $\langle D_G v_s, v_s\rangle$ is achieved at the s for which $\langle v_s, \operatorname{diag}(D)\rangle = 0$.

Proof. Write

$$\begin{aligned}\frac{\mathrm{d}\langle D_G v_s, v_s\rangle}{\mathrm{d}s} &= \left\langle \frac{\mathrm{d}v_s}{\mathrm{d}s}, \frac{\mathrm{d}\langle D_G v_s, v_s\rangle}{\mathrm{d}v_s}\right\rangle = \left\langle \frac{\mathrm{d}(y+s1_V)}{\mathrm{d}s}, 2D_G v_s\right\rangle = \langle 1_V, 2D_G v_s\rangle = 21_V^* D_G v_s \\ &= 2\left\langle \operatorname{diag}(D_G), v_s\right\rangle.\end{aligned}$$

This is 0 at the minimum. □

Theorem 4.18

Let z be a vector that is centered with respect to $\operatorname{diag}(D_G)$. Then there is a number τ such that

$$\phi(S_\tau) \le \sqrt{2\frac{\langle L_G z, z\rangle}{\langle D_G z, z\rangle}}.$$

We assume without loss of generality that

$$z_1^2 + z_n^2 = 1.$$

This can be achieved by multiplying z by a constant. We want to show that there is a τ for which

$$\phi(S_\tau) \le \sqrt{2\frac{\langle L_G z, z\rangle}{\langle D_G z, z\rangle}}.$$

We define a distribution on τ for which we can prove that

$$\frac{\mathsf{E}_\tau[d(\partial(S_\tau))]}{\mathsf{E}_\tau[\min\{\deg((S_\tau), \deg(V\setminus S_\tau))\}]} \le \sqrt{2\frac{\langle L_G z, z\rangle}{\langle D_G z, z\rangle}}.$$

This implies by linearity that there is some τ for which

$$\frac{d(\partial(S_\tau))}{\min\{\deg(S_\tau), \deg(V \setminus S_\tau)\}} \leq \sqrt{2\frac{\langle L_G z, z\rangle}{\langle D_G z, z\rangle}}.$$

The proof of this is tedious. In particular we first show

$$\mathop{\mathbb{E}}_{\tau}[d(\partial(S_\tau))] = \sum_{e \in E} d(e) \mathop{\mathbb{P}}_{\tau}[e \in \partial(S_\tau)] \overset{\text{Cauchy-Schwarz}}{\leq} \sqrt{2\frac{\langle L_G z, z\rangle}{\langle D_G z, z\rangle}} \langle D_G z, z\rangle.$$

And then we show

$$\mathop{\mathbb{E}}_{\tau}\left[\min\left\{\deg(S_\tau), \deg(V \setminus S_\tau)\right\}\right] = \langle D_G z, z\rangle.$$

From here the solution is straightforward algebraic manipulation.

To summarize, our result is

Theorem 4.19 (Cheeger's Inequality)

$$\frac{\nu_2}{2} \leq \phi_G \leq \sqrt{2\nu_2}.$$

A randomized algorithm to find a suitable cut that reaches this inequality is to embed G in $\mathbb{R}^{|V|}$, i.e., map each $v \in V$ to a spot in $\mathbb{R}$. Then we can just take the $|V| - 1$ possible thresholds, and take as a cut all points which are on one side of the threshold.

Theorem 4.20 (Generalized Cheeger Inequality)
Suppose $V = \bigcup_{i=1}^{k} V_i$ is a partition of V. Then

$$\sup_{i \in [k]} \phi(V_i) \leq \sqrt{\lambda_k \cdot \text{poly}(k)}.$$

The equivalent randomized algorithm is to embed G in $\mathbb{R}^{k \times |V|}$ and take similar thresholds, $|V| - 1$ in each coordinate, taking the best cut.

4.5 Expander Graphs

Definition 4.21 (Expander Graph)
A *combinatorial expander grpah* is a d-regular graph G such that

$$\phi(G) \geq \Omega(1).$$

A *spectral expander graph* is a d-regular graph G such that

$$\nu_2 \geq \Omega(1).$$

The latter property is verifiable, and by Cheeger's inequality, combinatorial expanders are equivalent to spectral expanders.

Theorem 4.22

For all planar d-regular graphs G,

$$\nu_2 \leq \frac{8d}{|V|}.$$

Random walks mix fast on expanders. Fix $\{X_t\}_{t\geq 0}$ a random walk, and let

$$P_t(x) = \mathsf{P}[X_t = x].$$

Then

$$P_t = \left(D_G^{-1/2} A_G D_G^{-1/2}\right)^t P_0.$$

It turns out that the steady-state for expanders is close to $P_\infty = 1_V$. After conditioning on X_0 one can consider the difference of P_t and P_∞, showing that convergence occurs exponentially fast. For a d-regular graph,

$$t \geq \frac{\log(1/\varepsilon)}{\log\left(\sup_{i\in V}\left|\lambda_i\left(D_G^{-1/2} A_G D^{-1/2}\right)\right|\right)} \implies \|P_t - P_\infty\|_2 \leq \varepsilon.$$

Notice that this only works for a connected graph. This yields a fairly obvious randomized algorithm which solves s-t connectivity in $O(\log(|V|))$ space (and $O(|V|^3 \log(|V|))$ time).

4.6 Solving Laplacian Systems

Given a graph G with Laplacian L_G, and a vector $b \in \mathbb{R}^V$, we want to solve the system

$$L_G x = b$$

for x. There is a near-linear time solver for these systems, by Spielman and Teng.

A fast starting point for this is the Kacmarz Method.

Algorithm 19 Kacmarz Method

Input: A matrix $A \in \mathbb{R}^{m\times n}$.
Input: A vector $b \in \mathbb{R}^m$.
Input: A time horizon T.
Output: A vector $x \in \mathbb{R}^n$ such that $Ax = b$.

```
x_1 ← starting point in R^n
for t ∈ [T] do
    i ← random index in m
    x_{t+1} ← x_t + (b_i − ⟨a_i, x⟩) a_i / ‖a_i‖_2^2
return x_T
```

This is a sort of randomized projected gradient descent onto the constraint set; the update rule projects x_t onto the hyperplane $\{x : \langle a_i, x\rangle = b_i\}$. We can improve the analysis by the following optimization rule:

Algorithm 20 Randomized Kacmarz Method

Input: A matrix $A \in \mathbb{R}^{m \times n}$.
Input: A vector $b \in \mathbb{R}^m$.
Input: A time horizon T.
Output: A vector $x \in \mathbb{R}^n$ such that $Ax = b$.

$x_1 \leftarrow$ starting point in $\mathbb{R}^n$
for $t \in [T]$ **do**
  $i \leftarrow$ random index in m, selecting index j with probability $\frac{\|a_j\|_2^2}{\|A\|_F^2}$
  $x_{t+1} \leftarrow x_t + (b_i - \langle a_i, x \rangle) \frac{a_i}{\|a_i\|_2^2}$
return x_T

The analysis is that

$$\begin{aligned}
x_{t+1} &= x_t + (b_i - \langle a_i, x \rangle) \frac{a_i}{\|a_i\|_2^2} \\
x_{t+1} - x_* &= x_t - x_* + (b_i - \langle a_i, x \rangle) \frac{a_i}{\|a_i\|_2^2} \\
&= \left(I - (\overline{a}_i)(\overline{a}_i)^*\right)(x_t - x_*) && \text{(using } \overline{a}_i = \tfrac{a_i}{\|a_i\|_2} \text{ and } \langle a_i, x^* \rangle = b_i\text{)} \\
\|x_{t+1} - x_*\|_2^2 &= \left\|(I - \overline{a}_i \overline{a}_i^*)(x_t - x_*)\right\|_2^2 \\
\mathop{\mathsf{E}}_{i \in [m]}\left[\|x_{t+1} - x_*\|_2^2\right] &= \mathop{\mathsf{E}}_{i \in [m]}\left[\left\|(I - \overline{a}_i \overline{a}_i^*)(x_t - x_*)\right\|_2^2\right] \\
&= \mathop{\mathsf{E}}_{i \in [m]}\left[\|x_t - x_*\|_2^2\right] - \mathop{\mathsf{E}}_{i \in [m]}\left[\langle \overline{a}_i, x_t - x_* \rangle^2\right] \\
&= \|x_t - x_*\|_2^2 - \mathop{\mathsf{E}}_{i \in [m]}\left[\langle \overline{a}_i, x_t - x_* \rangle^2\right] \\
&= \|x_t - x_*\|_2^2 - \sum_{i \in [m]} \mathsf{E}\left[\langle \overline{a}_i, x_t - x_* \rangle^2 \,\middle|\, \text{chooses } i\right] \mathop{\mathsf{P}}_{i \in [m]}[\text{chooses } i] \\
&= \|x_t - x_*\|_2^2 - \sum_{i \in [m]} \langle \overline{a}_i, x_t - x_* \rangle^2 \mathop{\mathsf{P}}_{i \in [m]}[\text{chooses } i] \\
&= \|x_t - x_*\|_2^2 - \sum_{i \in [m]} \langle \overline{a}_i, x_t - x_* \rangle^2 \frac{\|a_i\|_2^2}{\|A\|_F^2} \\
&= \|x_t - x_*\|_2^2 \left(1 - \frac{1}{\|A\|_F^2} \frac{\|A(x_t - x_*)\|_2^2}{\|x_t - x_*\|_2^2}\right). \\
&\le \|x_t - x_*\|_2^2 \left(1 - \frac{\|A\|_{\text{op}}^2}{\|A\|_F^2}\right).
\end{aligned}$$

Thus $\|x_{t+1} - x_*\|_2^2 \to 0$ in expectation, or with large enough T.