Lecture 4

Lecturer: Aleksander Mądry *Scribe: Jarosław Błasiok*

1 Overview

In this lecture, we will introduce a technique of *smoothing*. This technique will enable us to apply the projected gradient descent method we developed last time to optimization problems that have non-smooth objective functions. A prime example of such problem is the maximum flow problem. We will see that smoothing the ℓ_∞ objective function in this problem's formulation by a smooth "soft max" function smax_δ and then applying projected gradient descent algorithm to it allows us to obtained a better convergence rate than we previously got with the subgradient descent method.

We will then focus on the basic primitive that we somewhat neglected so far: computing a projection onto the set of valid s-t flows, which corresponds to electrical flow computations. We will see how general duality theory for non-linear optimization enables us to reduce electrical flow computations to a strictly linearly algebraic problem: solving a system of linear equations. Finally, we will start our discussion of general approaches to solving such linear systems.

2 Recap of the Last Lecture

Much of our discussion in this course revolves around the general problem of *constrained convext optimization problem*, that is, solving a problem of the following form

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & x \in \mathcal{K}, \end{aligned}$$

where the objective function f and the feasible set $\mathcal{K}$ are both convex.

As we know, these kind of problems can be solved in polynomial-time, e.g., via ellipsoid method. However, we are interested in much faster – if not necessarily exact – algorithms. In particular, in the previous lecture we have analyzed the classic algorithm called *projected gradient descent*, whose description appears below.

Algorithm 1 Projected gradient descent

Take x_1 to be an arbitrary point in $\mathcal{K}$
for $s = 1 \dots T-1$ **do**
  $x_{s+1} \leftarrow \Pi_{\mathcal{K}}\left(x_s - \eta \nabla f(x_s)\right)$
end for

Here, $\Pi_{\mathcal{K}}$ is an (ℓ_2-)projection onto a convex set $\mathcal{K}$, defined as

$$\Pi_{\mathcal{K}}(x) := \text{argmin}_{y \in \mathcal{K}} \|x - y\|. \tag{1}$$

Note that the gradient descent algorithm is well-defined only if the gradients of the objective function f are so. Moreover, in order to have good guarantees on the performance of this algorithm, we need even stronger assumptions on the function f called *L-smoothness*.

Definition 1 (L-smoothness) *We say f is L-*smooth*, for some $L \geq 0$ iff*

$$\forall\, x, y, \quad \|\nabla f(x) - \nabla f(y)\|_2 \leq L \cdot \|x - y\|_2.$$

Observe that L-smoothness corresponds to having the gradient ∇f of f be a Lipschitz function with parameter L.

Last time, we have proved the following theorem.

Theorem 2 *If f is L-smooth and we set $\eta = \frac{1}{L}$ then, after T iterations of the Algorithm 1, we have that*

$$f(x_T) - f(x^*) \leq O\left(\frac{LR^2}{T}\right), \tag{2}$$

where x^* is an optimal solution (i.e., a point $x^* \in \mathcal{K}$ that minimizes f), and $R := \|x_1 - x^*\|$ is bounded by the radius $\sup_{x,y\in\mathcal{K}} \|x - y\|$ of $\mathcal{K}$.

The convergence bound provided by this theorem should be contrasted with the $\frac{RG}{\sqrt{T}}$ convergence bound that the projected subgradient descent algorithm yielded. There, G was the Lipschitz constant of f (instead of ∇f), and the guarantee was on the convergence of the *average* $\bar{x}_T$ of all the visited points (instead of the sequence of the points itself).

So, we can see that the L-smoothness of f allows us to achieve a much better – only quadratic – dependence on T. (Although the dependence on R is worse.)

3 Smoothing Technique

Let us get back now to our formulation of the maximum flow problem.

$$\begin{aligned} \min \quad & \|f\|_\infty \\ \text{s.t.} \quad & Bf = \chi_{s,t} \end{aligned}$$

Recall that here

$$B_{v,e} := \begin{cases} -1 & v \text{ is the head of } e \\ 1 & v \text{ is the tail of } e \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where the orientation of the edges of G is arbitrary, and $\chi_{s,t}$ is defined as

$$\chi_{s,t}(v) := \begin{cases} -1 & v = s \\ 1 & v = t \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

Given the better performance of the projected gradient method, we would like to use it – instead of projected subgradient descent method – to obtain a faster maximum flow algorithm. At first, this seems to be a hopeless task. After all, our objective function $\|\cdot\|_\infty$ is not even differentiable everywhere. Consequently, its gradient might not even exist at some points, let alone be Lipschitz.

Observe, however, that as long as the coordinates of x are all non-zero and have distinct values the gradient of $\|x\|_\infty$ exists and is very well behaved – in fact, its Lipschitz constant is only 1. Can we thus maybe "fix" this function somehow by changing it slightly to make it smooth also in the critical regions where some of the coordinates are zero? Having such a "fixed" function at our disposal we would be able to run gradient descent on it and thus hopefully take advantage of the improved convergence provided by Theorem 2.

It turns out that such a "fixing" of a function – usually referred to as *smoothing* – is indeed possible. In fact, it is a powerful and widespread tool in many areas of continuous (and not even necessarily convex) optimization.

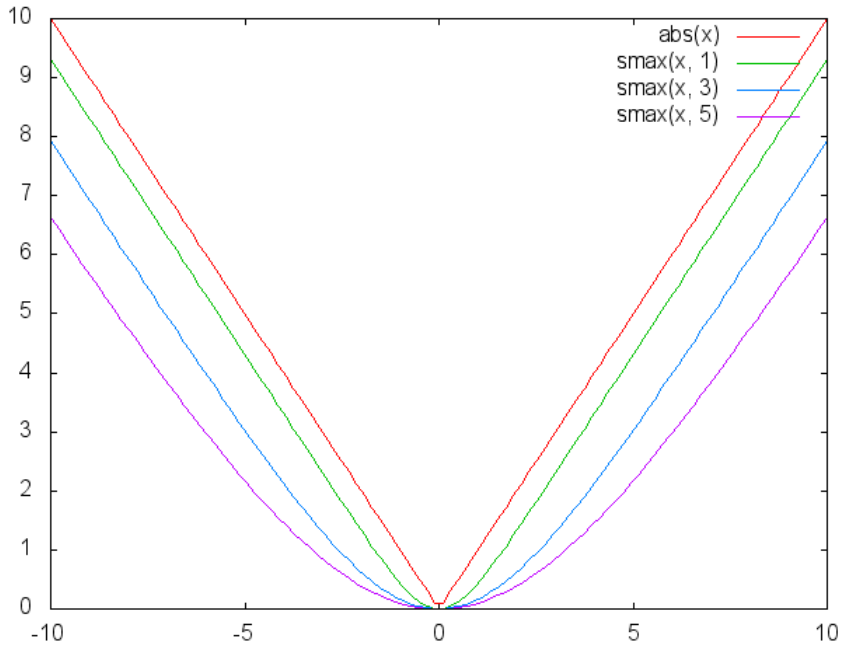

Figure 1: Plots of smax_δ functions for $\delta \in \{1, 3, 5\}$, compared to $|\cdot|$

3.1 The smax_δ Function

In the context of the ℓ_∞-norm that we are interested here, the function to consider is the so-called *soft-max* function smax_δ defined as

$$\mathrm{smax}_\delta(x) := \delta \ln \left(\frac{\sum_{i=1}^n e^{\frac{x_i}{\delta}} + e^{\frac{-x_i}{\delta}}}{2n} \right), \tag{5}$$

where δ is a parameter. See Figure 1 for a visualization of this function in one dimension.

Note that this function has all the properties that make it convenient to optimize with respect to it. It is easy to evaluate it, compute its gradient, as well as is convex and smooth.

Lemma 3 *For every $\delta > 0$, the function smax_δ is convex and $\frac{1}{\delta}$-smooth.*

(The proof is left as an exercise.)

Of course, the fact that the soft-max function is "nice" from the optimization point of view is only one of the aspects we care about. After all, a function that is constant everywhere would be even "nicer" in this regard. So, the other important property of soft-max is that it approximates the ℓ_∞ norm very well, i.e., only up to a small additive error.

Lemma 4 *For every $x \in \mathbb{R}$, we have*

$$\|x\|_\infty - \delta \ln(2n) \leq \mathrm{smax}_\delta(x) \leq \|x\|_\infty. \tag{6}$$

Proof For an upper bound, we will just bound every summand in the sum in the argument of the natural logarithm in $\mathrm{smax}(x)$ by the maximum one.

$$\begin{aligned}
\operatorname{smax}_\delta(x) &= \delta \ln\left(\frac{\sum_{i=1}^n e^{\frac{x_i}{\delta}} + e^{\frac{-x_i}{\delta}}}{2n}\right) \\
&\leq \delta \ln\left(\frac{\sum_{i=1}^n 2e^{\|x\|_\infty/\delta}}{2n}\right) \\
&= \delta \ln e^{\|x\|_\infty/\delta} \\
&= \|x\|_\infty
\end{aligned}$$

Whereas for lower bound we will observe that the very same sum is larger than its largest summand.

$$\begin{aligned}
\operatorname{smax}_\delta(x) &= \delta \ln\left(\frac{\sum_{i=1}^n e^{\frac{x_i}{\delta}} + e^{\frac{-x_i}{\delta}}}{2n}\right) \\
&\geq \delta \ln\left(\frac{e^{\|x\|_\infty/\delta}}{2n}\right) \\
&= \|x\|_\infty - \delta \ln(2n)
\end{aligned}$$

■

Looking at Lemmas 3 and 4, we see that there is a certain tension between the two aspects of the function smax. On one hand, we want this function to be as smooth as possible, which translates to keeping δ large. On the other hand, we want it to approximate ℓ_∞-norm closely, which requires δ be small. Optimizing this trade-off between the quality of approximation and the smoothness offered is the key challenge in the context of smoothing. The better trade-off we achieve the better algorithm we get (but it is not hard to see that there are fundamental limits to how good a trade-off one can get).

3.2 Solving the Maximum Flow Problem via Smoothing

Now, once we introduced the $\operatorname{smax}_\delta$ function, we can use it to smoothen the ℓ_∞ norm and apply the gradient descent algorithm to such smoothened version of the maximum flow problem. Specifically, let us consider the following optimization problem.

$$\begin{aligned}
\min \quad & \operatorname{smax}_\delta(f) \\
\text{s.t.} \quad & Bf = \chi_{s,t}
\end{aligned}$$

where we set $\delta := \frac{\varepsilon}{\ln(2n)}$. By Lemma 4 we know that working with this smoothened version of the maximum flow problem leads to an additive error of at most $\delta \ln(2n) = \varepsilon$, which is entirely acceptable for us. (After all, our algorithm incurs an additive error of ε anyway.)

Once we are working with the $\operatorname{smax}_\delta$ objective, we are able to apply the projected gradient descent (Algorithm 1) to it. By Lemma 3, the smoothness parameter of $\operatorname{smax}_\delta$ is equal to $L = \frac{1}{\delta} = \frac{\ln(2n)}{\varepsilon}$ and, from previous lecture, we know that $R \leq \sqrt{n}$. Consequently, by Theorem 2 and our discussion above, we have that after

$$T_{PGD} = \mathcal{O}\left(\frac{LR^2}{\varepsilon}\right) = \mathcal{O}(n\varepsilon^{-2}\log n) = \widetilde{\mathcal{O}}(n\varepsilon^{-2}) \tag{7}$$

iterations we obtain a flow solution that is within 2ε additive error of the optimal one.

3.3 Why Does Smoothing Help?

Observe that the number T_{PGD} of iterations we achieved now is significantly smaller – by a factor of roughly m – than the bound of $T_{SGD} = O(mn\varepsilon^{-2})$ we got last time, when we applied the subgradient descent method directly to the maximum flow problem. Where is this improvement coming from?

After all, the general approach that we employed is exactly the same in both cases – we are using local informations provided by the (sub) gradient to guide our greedy improvement in every step. The

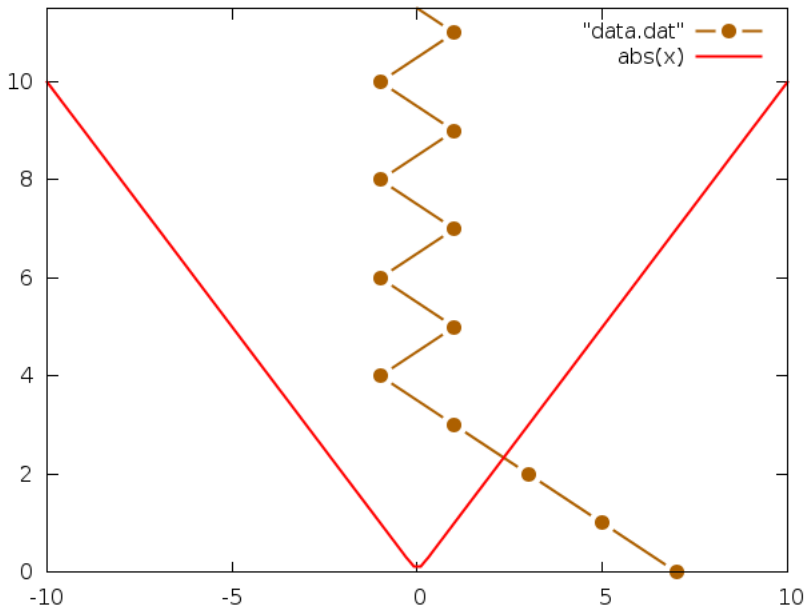

Figure 2: Intuitive picture presenting a oscillation around optimum of subgradient descent for minimizing $|x|$. Brown dots represents subsequent steps of algorithm — the position along x axis is a value x_t, where the y-coordinate of every such dot represents the iteration t. Red line depicts the objective function $|x|$.

functions we apply this procedure to are different, but only slightly. Was the previous, worse, bound just an artifact of our analysis, or is our new algorithm different in some fundamental way?

To get more intuition about this, let us consider a very simple optimization problem: *unconstrained* minimization of the ℓ_∞-norm in *one* dimension. Clearly, in this case, $\|x\|_\infty$ is simply the absolute value $|x|$ of x.

Let us think now what happens when we apply the subgradient descent method to this problem with our starting point being some $x_1 > 0$ – see Figure 2. At first, the algorithm will take steady steps toward the optimum point, $x = 0$. As long as $x \neq 0$, the gradient (which is now simply a derivative) is either 1 or -1, depending on whether x is positive or not. So, each step of this algorithm will have exactly the same length η. Once this sequence of steps reaches the neighborhood of the optimum point, i.e., $0 \leq x_s \leq \eta$, the point x_s will start osculating around $x = 0$ indefinitely – each step "overshooting" the optimum and never converging.

This constant overshooting and corresponding indefinite oscillation illustrates the key problem with applying gradient descent–type approaches to non-smooth functions. Namely, the local information on such function, i.e., the (sub)gradients, might not give us any information on how far from the optimum we are – only what is the direction in which the optimum lies. This is a big problem when we need to choose the step size. After all, on one hand, we want to take large steps when we are far away from the optimum – so as make sufficient progress. But, on the other hand, we want to take only small steps once we are in the optimum's vicinity – in order to avoid too large oscillations around it. The lack of any hints on the current distance to the optimum forces us to make one, suboptimal, fit-it-all choice of the step size. This need to make a fixed choice already leads to a slower convergence. Also, the possibility of indefinite oscillations around the optimum necessitates focusing on the, slower, convergence of the sequence of averages of all the intermediate points instead of on the direct convergence of the intermediate points, as the latter sequence might not even converge.

The above discussion highlights the key goal of smoothing: encoding the information about the critical non-smooth regions of the function in the local (gradient) structure of these region's neighborhood. In

particular, in our example of $|x|$ minimization, if we work with the $\text{smax}_\delta(x)$ function instead, then the gradients of $\text{smax}_\delta(x)$ will become increasingly smaller the closer we are to the optimum. Consequently, the resulting gradient descent step size will be always appropriately attuned – large when far away from the optimum, smaller when in optimum's vicinity – and will yield fast and direct convergence. We thus see that applying the smoothing technique, even if seemingly innocent, might have a dramatic effect both on the algorithm itself and its performance.

4 Computing a Projection on the Space of s-t Flows

The performance of all the algorithms for the maximum flow problem that we developed so far was measured in terms of the number of (sub)gradient-descent steps that the algorithm takes. However, taking each such steps involved not only computing a gradient (that was so far a straight-forward operation) but also computing a projection $\Pi_{\mathcal{F}_{s,t}}$ of our new point on the feasible set $\mathcal{F}_{s,t}$ of all the valid s-t flows of value 1. The latter operation seems non-trivial. So, in order to bound the actual running time of our algorithms we need to discuss now how to perform it efficiently.

4.1 Electrical Flows

By definition, computation of the projection $\Pi_{\mathcal{F}_{s,t}}$ of a given point (flow) g is captured by the following ℓ_2-minimization problem.

$$\begin{aligned} \text{argmin} \quad & \|g - f\|_2 \\ \text{s.t.} \quad & Bf = \chi_{s,t}, \end{aligned}$$

where the matrix B (defined in (3)) and the vector $\chi_{s,t}$ (defined in (4)) describe the affine space $\mathcal{F}_{s,t}$.

Taking $h := g - f$, we can consider an equivalent problem

$$\begin{aligned} \text{argmin} \quad & \|h\|_2 \\ \text{s.t.} \quad & Bh = \sigma, \end{aligned}$$

where $\sigma := \chi_{s,t} - Bg$ is a fixed vector of demands that the flow h has to obey in order to have $f = h - g$ be a valid s-t flow.

In other words, our problem is to find a flow h that minimizes ℓ_2 norm over all the flow that obey the flow demand pattern described by σ. It turns out that this kind of question is very well studied, both in optimization and in physics and corresponds to the notion of *electrical flow*.

Electrical flow problem:

- *Input:* A graph $G = (V, E)$, a demand vector σ, and a positive resistance r_e associated with each edge e.
- *Goal:* Find the optimal solution h^* to the following minimization problem

$$\begin{aligned} \min \quad & \frac{1}{2} h^T R h \\ \text{s.t.} \quad & Bh = \sigma \end{aligned}$$

Here, the matrix R is an $m \times m$ diagonal matrix defined as

$$R_{e,e'} := \begin{cases} r_e & \text{if } e = e' \\ 0 & \text{otherwise} \end{cases}, \tag{8}$$

and thus we have that

$$h^T R h = \sum_e r_e h_e^2 \tag{9}$$

is just the *energy* of the flow h, as we know it from physics.

Observe that by introducing the resistances r_e we made the above definition slightly more general. However, we recover our original problem by taking all resistances to be equal to 1. (Note that the $\frac{1}{2}$ scaling factor in front of the energy in the objective function does not change the problem – it is there just for notational convenience.)

4.2 Electrical Flows and Linear Systems

Once we formalized the problem we need to solve, let us try to understand how we can go about solving it. Clearly, electrical flow computation corresponds to a convex optimization problem, so as usual we might just use some off-the-shelf method, such as ellipsoid method, to solve it. Needless to say, this answer is hardly satisfying, as the whole point of our approach was to reduce the maximum flow computations – which also can be solved via ellipsoid method – to solving a simpler task that admits much more efficient algorithms.

As we will see soon, our hopes are not ill-founded. Electrical flow problem is indeed a much simpler optimization question. In fact, it boils down to linear algebra – more precisely, to solving a *linear system*, i.e., a system of linear equations, that has a very special structure.

In order to establish this connection we will need to make a very brief detour through so-called *duality theory*. Duality theory is a very fundamental and useful concept in optimization. Very roughly speaking, it relates to the fact that for every optimization problem $\mathcal{P}$ of the general form:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & x \in \mathcal{K}, \end{aligned}$$

which is usually referred to as *primal problem*, there exists a different optimization problem $\mathcal{D}$

$$\begin{aligned} \max \quad & \bar{f}(y) \\ \text{s.t.} \quad & y \in \bar{\mathcal{K}}, \end{aligned}$$

which is called the *dual problem* that if very closely related to $\mathcal{P}$.

Specifically, for *any* feasible solution x to the primal problem $\mathcal{P}$ and *any* feasible solution y to the dual problem $\mathcal{D}$, we have that

$$f(x) \geq \bar{f}(y). \tag{10}$$

This phenomena is called *weak duality*. It implies, in particular, that any feasible primal solution provides an upper bound on the value $\bar{f}(y^*)$ of an optimal solution y^* to the dual problem and any feasible dual solution provides a lower bound on the value $f(x^*)$ of the optimal solution x^* to the primal problem.

Even more interestingly, provided certain technical conditions are satisfied (which will be always the case in our discussion), we also have *strong duality*:

$$f(x^*) = \bar{f}(y^*). \tag{11}$$

That is, the value $f(x^*)$ of the primal optimal solution x^* is equal to the value $\bar{f}(y^*)$ of the dual optimal solution.

The connection of electrical flow problem to linear system solving is a consequence of the strong duality for the so-called *Lagrangian* formulation of that problem that is given via the following min-max optimization problem

$$\min_h \max_\varphi \mathcal{L}(h, \varphi), \tag{12}$$

where

$$\mathcal{L}(h, \varphi) := \frac{1}{2} h^T R h - (Bh - \sigma)^T \varphi \tag{13}$$

and $\varphi \in \mathbb{R}^n$ is a vector of dual variables that are sometimes referred to as *vertex potentials*.

Observe that the problem (12) is an unconstrained problem, that is, both h and φ can be arbitrary vectors. However, this problem turns out to be completely equivalent to the constrained optimization problem that served as a definition of the electrical flow problem.

To see why this is the case, one should think about the min-max problem (12) as a two player game. In this game, the first player, let's call him/her the *minimization* player, has to first commit to some vector h. Then, seeing this choice of h, the second player, let's call him/her the *maximization* player, chooses a vector φ so as to make the value of the Lagrangian $\mathcal{L}(h, \varphi)$ maximal. The optimal value of this optimization problem is exactly the minimum value of $\mathcal{L}(h, \varphi)$ that the minimization player can guarantee why playing that game.

With this interpretation in mind, it is not hard to see why this Lagrangian formulation indeed captures electrical flow problem. Observe that if the minimization player chooses h that is not feasible for our flow problem, i.e., $Bh \neq \sigma$, then the vector $Bh - \sigma$ is non-zero, which enables the maximization player to choose φ that will make the value of $\mathcal{L}(h, \varphi)$ arbitrarily large. So, despite the apparent freedom to choose arbitrary h, the minimization player has to stick only to feasible hs. This means though that we always have that $Bh - \sigma = 0$ and thus the maximization player has no impact on the value of $\mathcal{L}(h, \varphi)$ anymore. Our problem boils down therefore to finding a feasible flow h that minimizes

$$\mathcal{L}(h, \varphi) = \frac{1}{2} h^T R h - (Bh - \sigma)^T \varphi = \frac{1}{2} h^T R h,$$

which is exactly our electrical flow formulation.

Once we convinced ourselves that the problem (12) indeed captures the electrical flow problem, let us consider its *Lagrangian dual*

$$\max_{\varphi} \min_{h} \mathcal{L}(h, \varphi), \tag{14}$$

which is obtained by simply changing the order of maximization and minimization. This new program can be interpreted as a game as well, the only difference is that it is the maximization player who has to commit first to his/her choice. We thus have that

$$\min_{h} \max_{\varphi} \mathcal{L}(h, \varphi) \geq \max_{\varphi} \min_{h} \mathcal{L}(h, \varphi),$$

as the changing of the order can only give more power to the minimization player. In other words, the weak duality holds (cf. (10)).

Furthermore, the strong duality holds as well and it can be used to prove the following theorem.

Theorem 5 *There exists primal and dual solutions h^* and φ^* such that:*

(i) $\mathcal{L}(h^*, \varphi^*) = \max_{\varphi} \mathcal{L}(h^*, \varphi)$;

(ii) $\mathcal{L}(h, \varphi^*) = \min_{h} \mathcal{L}(h, \varphi^*)$;

(iii) $\nabla \mathcal{L}(h^*, \varphi^*) = 0$.

The first two conditions in the above theorem imply, in particular, that h^* and φ^* are optimal solutions to the primal and dual problems. However, these conditions also provide us with a much deeper and non-trivial statement.

Namely, going back to our game theoretic interpretation of these optimization problems, h^* and φ^* are optimal strategies for both players *irrespectively* of the order in which they need to reveal their choices. Even if one of these players can see first what the other one played, as long as the latter player chose to play according to these strategies, there is no gain for the former player in not following the strategy too. In other words, the strategies h^* and φ^* are *Nash equilibrium* of the underlying game. This can be viewed as a generalization of the von Neumann's MinMax theorem that proves existence of such Nash equilibrium for any zero-sum game and connects this fact to duality theory for *linear* programs.

Finally, the third condition in the theorem above is so-called *Karush-Kuhn-Tucker (KKT) condition* that is a necessary (but not always sufficient) condition for the solutions h^* and φ^* to be optimal.

Now, let us examine what does this KKT condition imply for our particular problem. By definition, the gradient $\nabla \mathcal{L}(h, \varphi)$ has $n+m$ coordinates, first part corresponds to φ and the second one corresponds to h.

For the first part, the KKT condition implies that

$$0 = \nabla_{\varphi} \mathcal{L}(h^*, \varphi^*) = -(Bh^* - \sigma), \tag{15}$$

which implies that h^* has to be a flow whose demand pattern is exactly σ.

For the second part, the KKT condition implies that

$$0 = \nabla_h \mathcal{L}(h^*, \varphi^*) = Rh^* - B^T\varphi^*,$$

which is equivalent to the statement that

$$h^* = R^{-1}B^T\varphi^*. \tag{16}$$

Let us take a moment to understand better what the above condition means. If we consider one coordinate of this equation, corresponding to some edge $e = (v, u)$, (16) will imply that

$$h_e^* = \frac{(\varphi_u - \varphi_v)}{r_e},$$

which is exactly the Ohm's law that we know from physics. So, (16) tells us that Ohm's law is just a primal-dual optimality condition for electrical flows. This also justifies referring to dual variables φ as vertex potentials.

Putting (15) and (16) together, we must have that

$$\sigma = Bh^* = B\left(R^{-1}B^T\varphi^*\right) = \left(BR^{-1}B^T\right)\varphi^* = L\varphi^*, \tag{17}$$

where $L := BR^{-1}B^T$.

Equation (17) implies that φ^* correspond to a solution to a linear system in a matrix $L = BR^{-1}B^T$. Moreover, once we compute φ^* we can easily get the corresponding electrical flow h^* by applying Ohm's law (16) to φ^*. Therefore, indeed computing the electrical flow h^* corresponds to solving a linear system.

Before we continue, let us remark that the matrix L is an extremely important matrix – it is the *Laplacian* matrix of the graph G. This matrix is a central object of the field of *spectral graph theory* that aims to tie the linear algebraic properties of this matrix to the combinatorial properties of the graph it describes. (Hopefully, we will be able to explore this topic in a bit more detail later in the semester.)

Also, as we will see soon, this matrix turns out to have a very interesting structure that will enable us to solve linear system in it, i.e., a *Laplacian system*, extremely fast – that is, in nearly-linear time. As a result, electrical flow computation ends up being a nearly-linear time computable primitive.

5 Fast Linear System Solving

Our quest for developing fast maximum flow algorithm brought us to another fundamental computational problem: solving a linear system. Specifically, we have shown that in order to perform a projection step in the projected gradient descent algorithm for the maximum flow problem, we need to solve a Laplacian system $L\varphi = \sigma$, where L is Laplacian matrix of the underlying graph.

Let us, therefore, switch gears and put the maximum flow problem aside to focus our attention on a new topic: fast algorithms for solving general linear systems. We will discover that there is a lot of beautiful ideas and intimate connections to convex optimization there.

5.1 Direct Methods

Consider a system of linear equations

$$Ax = b.$$

How could we go about solving it?

The first instinct is to simply compute A^{-1} and then multiply both sides of the linear system through it, obtaining an answer $x^* = A^{-1}b$.

Although this approach would certainly work, it has a couple of undesirable properties. First of all, computing an A^{-1} is quite costly computationally. Applying Gaussian elimination requires $O(n^3)$ time and if we resort to, fairly impractical, fast matrix multiplication algorithms we would obtain a running time of $O(n^\omega)$, where $2 \leq \omega \leq 2.373$. In either case, the running time would be super-quadratic and thus

prohibitive from our point of view. Also, another problem with this approach relates to numerical issues. Computing A^{-1} might involve division by small numbers, which is quite problematic when working with real-world finite precision arithmetic.

A slightly better solution is to find a decomposition of A into $A = LU$ where L is lower triangular and U is upper triangular – so-called *LU-decomposition of A*. Once we have found such a decomposition, the problem reduces to finding an x such that $Ax = L(Ux) = b$. This can be easily done in two steps — first we would find z such that $Lz = b$, then we would find x, such that $Ux = z$. Each of those steps consist of solving a system of linear equation in a triangular matrix — this can be easily done in time $\mathcal{O}(n^2)$ via simple back-substitution. (Other decompositions also could be used, most notably one can use QR-decomposition: decompose $A = QR$, where Q is orthonormal and R is upper triangular. Again, solving a linear system both in R as well as in Q is quite easy.) Unfortunately, computing such decompositions also requires at least $\Omega(n^\omega)$ time.

Finally, one other problem with this kind of approaches is that they do not exploit sparsity of the matrix A. That is, many real-world matrices of interest are *sparse*, i.e., the number of their non-zero entries m is much smaller than the total number of entries $O(n^2)$. In such situations, one would hope to have methods whose complexity can take advantage of this fact. However, both A^{-1} and the LU-decomposition of A might be dense, i.e., have $\Omega(n^2)$ non-zero entries, even if A is very sparse, i.e., m is $O(n)$. This ability to exploit sparsity is particularly relevant in the context of Laplacian systems, as the sparsity of a Laplacian is within a constant of the sparsity of the underlying graph.

5.2 Iterative Approaches

Given all these drawbacks of the above approaches (which are usually called *direct methods*), we focus our attention on a different family of algorithms: *iterative approaches*.

These approaches, much in the spirit of this class, solve the linear system in a sequence of steps. They start from some initial guess x_1, and then iteratively refine it, which each consecutive answer x_s being increasingly better. The guiding principle here – and one of the key advantages of these method – is making each refinement step very simple and easy to compute. In fact, each step boils down to a small number of multiplications of A by some vector y. Note that this basic operation not only can be implemented fast, i.e., in $O(m)$ time, but it also exploits the sparsity of the problem.

On the other hand, one of the less desirable features of iterative methods is that they never really compute the exact solution. Instead, they provide a sequence of answer that only converges to the exact solution. Furthermore, the rate of this convergence will largely depend on numerical properties of the matrix A (usually, on its eigenvalue or singular values). Consequently, for some matrices the convergence will be very fast, while for others it might be extremely slow.

In the next lecture, we will embark on the study of these methods to get a much better grasp of both their power and limitations.