1 Introduction

In this lecture, we finished off the low rank + sparse recovery topic from last class and laid the groundwork for fast Laplacian solvers.

2 Low Rank + Sparse

Suppose we are given a matrix $A$ which has a low rank plus sparse decomposition $A = L + S$ with suitable low rank matrix $L$ and sparse matrix $S$. In Lecture 25, we discussed methods and theory for the recovery of $L$ and $S$ via Principle Component Pursuit (PCP),

$$\min_{L,S} \|L\|_* + \lambda \|S\|_1,$$

subject to $A = L + S$.

Interestingly, this approach can be extended to situations in which a portion of the entries of $A$ are missing or unknown. This is of great importance for recommender systems, which seek to use incomplete sets of user rankings to predict user preferences [1]. For a more detailed discussion, see Section 1.2 on the Netflix Prize in Lecture 25.

Let $\Omega$ represent the set of known entries $(i,j)$ of $A$ and let $P_\Omega$ be the orthogonal projection onto the space supported on $\Omega$,

$$(P_\Omega X)_{ij} = \begin{cases} X_{ij} & (i,j) \in \Omega \\ 0 & \text{otherwise.} \end{cases}$$

If we only have partial information about $A$ and want to recover $L$ and $S$, we may write

$$Y = P_\Omega A = P_\Omega (L + S) = P_\Omega L + S'.$$

To recover the low rank component $L$ of $A$ using this incomplete data, we can reformulate [1] by relaxing the constraint on $L$ and $S$:

$$\min_{L,S} \|L\|_* + \lambda \|S\|_1,$$

subject to $P_\Omega (L + S) = Y$.

Here, $Y = P_\Omega A$ represents the known entries of $A$. For example, in the Netflix Prize example, $Y$ corresponds to a set of video rankings by a subset of users. Thus, the observation
operator $P_\Omega$ describes what data is visible to us, e.g. what movies have actually been ranked by which users. In this way, (2) seeks minimizers $L$ and $S$ that agree with the known entries of $A$.

In Lecture 25, Theorem 1, we saw that (1) enjoyed strong theoretical guarantees for the recovery of $L$ and $S$. A similar result holds for (2). Before we present this theorem, we recall the definition of an incoherence condition.

**Definition 1** (Lecture 25, Def 3) Let $L = U\Sigma V^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T$ be the singular value decomposition of $L$. $L$ satisfies an incoherence condition with parameter $\mu$ if the following conditions are satisfied.

1. $\max_i \|U^T e_i\|^2 \leq \frac{\mu k}{n}.$
2. $\max_i \|V^T e_i\|^2 \leq \frac{\mu k}{n}.$
3. $\|U V^T\|_\infty = \max_{ij} |(U V^T)_{ij}| \leq \sqrt{\frac{\mu k}{n^2}}.$

Speaking loosely, the incoherence condition ensures that $L$ is not too sparse, so as to avoid an identifiability issue between $L$ and $S$. Provided that the identifiability issue is avoided, we obtain a strong theoretical guarantee for the recovery of $L$ and $S$, even when our knowledge of $A$ is incomplete.

**Theorem 1** (Candes et al. 2011 [1]) Suppose that $L$ is $n \times n$ and satisfies an incoherence condition with $\mu$, and $\Omega$ is uniformly distributed among all sets of cardinality $p = \frac{n^2}{10}$. Now, suppose each observation is corrupted with probability $\tau$ independently. Then, there is a constant $c$ such that with probability $1 - cn^{-10}$, (2) with $\lambda = \frac{1}{\sqrt{n/10}}$ is exact. That is, if $\hat{L}$ is a minimizer of (2), then $\hat{L} = L$ provided that

$$\text{rank}(L) \leq c_1 n \mu^{-1} (\log n)^{-2} \quad \text{and} \quad \tau \leq c_2$$

Above, $c_1$ and $c_2$ are positive numerical constants.

Note the similarities and differences between the recovery guarantees for the case where $A$ is known fully, although some subset of data may be arbitrarily corrupted (Lecture 25, Theorem 1), and the case where $A$ is known only partially and, again, contains some corrupted data (Theorem 1 above). They both require only modest demands on the rank of $L$, which grows linearly with $n$ (apart from a log factor). In both cases, the constant $c$ in the probability of successful recovery depends on the constants $c_1$ and $c_2$. However, note that the size of the set of corrupted entries is fixed in the complete data case and the probability of successful recovery depends (through $c$) on this size. In the incomplete case above, the corruption is probabilistic across the known entries and the probability of successful recovery depends (through $c$) on the likelihood $\tau$ that an entry is corrupted. Remarkably, in both cases the probability of failure decreases rapidly as the size $n$ of the data increases. Finally, the choice $p = \frac{n^2}{10}$ in Theorem 1 is somewhat arbitrary. Similar statements could be proved for other choices of $p$, although the probability of success, the multiplier $\lambda$, and the constants $c_1, c_2,$ and $c$ would be impacted.
3 Introduction to Laplacian Matrices

We now change topics to discuss a certain class of linear systems that can be solved in approximately linear time. These systems are described by Laplacian matrices, which we will introduce in this section.

Consider a weighted, undirected simple graph $G$ with vertex set $V$, edge set $E$, and weights $w_{u,v} > 0$ on each edge $(u, v) \in E$. The adjacency matrix $A$ of $G$ is defined by

$$A_{u,v} = \begin{cases} w_{u,v} & \text{if } (u, v) \in E \\ 0 & \text{else} \end{cases}.$$ 

As an example, consider the graph in Figure 1. For this graph, the adjacency matrix is

$$A = \begin{pmatrix} w_{1,2} & 0 \\ w_{1,2} & w_{2,4} \\ 0 & w_{3,4} \end{pmatrix}.$$ 

For node $u \in V$, the degree of $u$ is defined

$$d(u) = \sum_{v \in V} w_{u,v},$$

which is equivalent to a row sum of $A$. Define $D$ to be the diagonal matrix of node degrees in $G$.

The Laplacian matrix of $G$ is then $L = D - A$. It can equivalently be defined via a quadratic form: for $x \in \mathbb{R}^{|V|}$,

$$x^T L x = \sum_{(u,v) \in E} w_{u,v} (x(u) - x(v))^2.$$ 

In general, we can think of a Laplacian matrix $L$ as any matrix with the following properties:

26: Low Rank + Sparse and Fast Laplacian Solvers -3
• row sums = 0
• non-positive entries off the diagonal.

Notice that $L e = (D - A) e = 0$, where $e$ is the all ones vector. Thus the null space of $L$ is nontrivial, and $L$ has a 0 eigenvalue. We will generally be interested in solutions with prescribed entries of the pseudo-inverse.

Now that we’ve introduced Laplacian matrices, when do we actually want to solve systems of this type? We provide three common instances of these systems as motivation.

1. **Solving max-flow problems.** In this case, we have a weighted network that describes flow capacity over any given edge. We are interested in the maximum flow that is achievable from a given source node to a given target node. We can write this problem as a linear program. When solved with an interior point method, this requires many solutions of restricted Laplacian systems.

2. **Resistor networks.** In this case, edges are resistors with weights $\frac{1}{\text{resistance}}$. Then if $i_{\text{ext}}$ defines a current flow in and out of given nodes, solving $L p = i_{\text{ext}}$ gives a potential value for each node. If $i_{\text{ext}}$ describes unit flow into $u$ and unit flow out of $v$, then $p(u) - p(v)$ describes the effective resistance between $u$ and $v$.

3. **Solving PDEs.** Laplacian matrices often arise when discretizing PDEs. For instance, solving general elliptic PDEs with finite element methods yield Laplacian systems.

Note that the graphs in the last case have a lot of structure (e.g., lattice). We will be concerned with more general graphs, where we don’t assume anything about graph structure.

### 4 Fast Laplacian Solvers

In this section, we will introduce ingredients necessary for solving Laplacian systems ‘fast’. I.e., in time similar to $O\left(\text{nonzeros}(A) \log \left(\frac{1}{\varepsilon}\right) \log^c n\right)$.

Say we want to solve $L x = b$. We can do this with a Cholesky decomposition in the following way:

1. Compute $L = CC^T$, where $C$ is lower triangular.
2. Solve $Cy = b$.
3. Solve $C^Tx = y$.

Note here that $L$ is symmetric positive semi-definite. It has non-trivial null space, but if $G$ is connected, it only has one 0 eigenvalue, which Cholesky can handle. On the other hand, if the graph is not connected, then $L$ is block diagonal and can be decomposed into separate smaller Laplacian systems that describe connected subgraphs.

Steps 2 and 3 above can be solved in $O(\text{nonzeros}(C))$ time. Thus if we want to use the Cholesky decomposition in a fast solver, we need to control the sparsity of $C$. The following example illustrates how this can be a problem.
Example 1

Consider the matrix $L$ corresponding to a star graph, in which the 1st column, 1st row, and diagonal are the only nonzeros.

$$
L = \begin{bmatrix}
\times & \times & \times & \cdots & \times \\
\times & \times & & & \\
\times & & \times & & \\
\vdots & & & \ddots & \\
\times & & & & \times \\
\end{bmatrix}
$$

The sparsity of $L$ is $O(n)$ where $n$ is the number of nodes. Now consider one “step” into the Cholesky factorization, which yields the following matrix

$$
\begin{bmatrix}
\times & 0 & \cdots & 0 \\
0 & & & \\
\vdots & & & \text{Dense} \\
0 & & & \\
\end{bmatrix}
$$

In just one step we have returned to something with $O(n^2)$ nonzero entries.

If $L$ is sparse, we would like to preserve this sparsity in a Cholesky factorization. One idea is to reorder the rows and columns of $L$ in a way such that the Cholesky factorization is sparse. I.e., for a permutation matrix $P$, we want $PLP^T$ to have sparse Cholesky factors. As we will see in the next lecture, one way to get to a fast solver is to

1. Compute an approximate Cholesky factorization with controlled sparsity. I.e., a Cholesky factorization of something close to $L$: $L \approx PCC^T P^T$. We will see how to do this by randomly throwing some stuff away at each step of the factorization.

2. Use this as a preconditioner in an iterative method in order to bound the number of iterations required.

The end algorithm requires covering both fronts: maintaining sparsity of a Cholesky factorization and bounding the number of steps in an iterative method.

References