Stat260/CS294: Randomized Algorithms for Matrices and Data

Lecture 1 - 09/04/2013

Lecture 1: Introduction and Overview

Lecturer: Michael Mahoney *Scribe: Michael Mahoney*

Warning: these notes are still very rough. They provide more details on what we discussed in class, but there may still be some errors, incomplete/imprecise statements, etc. in them.

1 Overview of topics and class

Today we will start with some general discussion, and next time we will start to get into the details. There is no particular reading for today, but here is an overview of the area where some of these themes are discussed and that will be a useful reference throughout the semester.

- Mahoney, "Randomized Algorithms for Matrices and Data," FnTML 2011.

1.1 Initial thoughts on randomized matrix algorithms

This course will cover recent developments in randomized matrix algorithms of interest in large-scale machine learning and statistical data analysis applications. By this, we will mean basic algorithms for fundamental matrix problems—such as matrix multiplication, least-squares regression, low-rank matrix approximation, and so on—that use randomization in some nontrivial way. This area goes by the name RandNLA (Randomized Numerical Linear Algebra) or RLA (Randomized Linear Algebra). It has led to several rather remarkable theoretical, implementation, and empirical successes so far, and a lot more is currently being developed by researchers.

Although very elementary forms of randomization are commonly used in linear algebra, e.g., the starting vector in Lanczos algorithms are typically random vectors, randomization has historically been anathema in matrix algorithms and numerical linear algebra. This stands in stark contrast to its widespread use in RandNLA in recent years, where it has proven to be a powerful resource for improved computation, and to its widespread acceptance in various forms in machine learning, statistical data analysis, etc. The recently-developed randomized matrix algorithms that we will cover in this class typically use randomness to perform random sampling, i.e., choosing, typically in a judicious manner to highlight structural properties of interest, a small number of columns or rows or elements from the matrix, or performing a random projection, i.e., projecting in a data-agnostic manner the original data to a much lower dimensional space. In either case, one typically hopes that the "sketch" of the original data that is thereby constructed is "similar" to the original full data set, so that if one runs relatively more expensive computations of interest on the sketch, then one gets a good approximation to the output of computations on the full data set.

While there has been a lot of interest in these randomized matrix algorithms, in and of themselves, much of the interest arises since many additional machine learning and data analysis methods either directly call these algorithms as black boxes or indirectly use very similar ideas within their analysis. Indeed, having been motivated by large-scale statistical data analysis problems, the area of RandNLA has received attention by and been developed by researchers from theoretical computer science, statistics, numerical linear algebra, optimization, scientific computing, data analysis, machine learning, as well as domain sciences such as astronomy, genetics, and internet data analysis. Given this diversity of approaches, the challenge is to distill out common algorithmic and statistical principles responsible for the success of these methods, to highlight commonalities and differences, so that these methods may be applied more generally. Thus, in additional to explaining the basic ideas underlying these methods and what is going on "under the hood" that makes these methods work, the course will also make connections with how these and similar ideas appear in other related machine learning and data analysis problems.

Today, we will start with a high-level description of some background ideas, mainly to set some context, and then we will describe a simple algorithmic primitive for sampling—uniformly or non-uniformly—in a not-immediately-obvious way from a large matrix. Next time, we will get into details of our first non-trivial RandNLA algorithm for a matrix problem.

1.2 Data, models of data, and approaches to computing on data

Before proceeding, it is worth remembering that, although we will often refer to the data as a matrix, matrices are just matrices, and data are whatever data are. That is, if you are already thinking of the data as consisting of m things, each of which is described by n features, or as the correlations between n pairs of things, then you have already made very strong assumptions about the data—that may or may not be appropriate in particular applications. Examples of data include discretized images, base pair information read out of a genetic sequencing machine, click logs at an internet site, records of consumer transactions, call records to a 911 site of criminal activity in a city, and so on; and in none of those examples are matrices explicitly mentioned. Matrices appear in these and other applications since they are a useful way to *model* the data. By this, we mean a useful way to encode data into a mathematical object such that we can run computations of interest on that object in a reasonable time and get answers that are useful in some sense in the application domain that generated the data.

Of course, matrices are not the only way to model data. Here are several other common ways data can be modeled.

- **Turing machine.** This a tape with $\{0, 1\}$ entries, upon which Turing machine operations are performed. This is a popular way to model data if we are interested in characterizing the algorithmic complexity of problems, especially in the sense of polynomial-time equivalence.
- **Database table.** This a table of $(key, value)$ pairs, upon which database/logical operations are performed. This is a popular way to model data in the the field of databases, and it is of interest to us since really large-scale data are typically held in some sort of database.
- **Strings.** This is a sequence of characters, e.g., an array of bytes/words that stores a sequence of characters or more general arrays/lists. This is popular in certain theoretical areas, e.g., formal language theory, as well as in certain data analysis areas, e.g., bioinformatics.
- **Graphs.** This is a set of things and pairs of things, i.e., $G = (V, E)$, where V is a set of nodes, and $E \subset V \times V$ is a set of edges. This is a popular way to model data since data often

consist of n things and some sort of the pairwise relationship between these things.

Of course, these models are not inconsistent, and it is often helpful to model the data in different ways, depending on what one is interested in doing. For example, we may want to model the data as a real-valued matrix, since we are interested in performing matrix computations on the data, but if we are going to implement our matrix algorithms on a computer, then we need to represent those real numbers in terms of a fixed number of bits, in which case we would need to implement algorithms in a manner that is well-behaved with respect to this discretization. Alternatively, we may want to model the data as a matrix, but the data might be large enough that they are stored in a database, in which case we would have to implement those computations in a manner that respected the constraints imposed by the database query language.

One of the main points is that for each of these ways to model the data, certain types of operations tend to be relatively easy, and other types of operations tend to be relatively difficult. For example, modeling the data in a discrete way such as with a graph or a Turing machine tape makes it relatively easy to make statements about worst-case algorithmic complexity; but this is often not robust to the addition of a bit of noise. Indeed, one of the benefits of modeling the data as real-valued matrices is that the geometry of Euclidean spaces provides a robustness that, in addition to leading to relatively fast algorithms, underlies good statistical inferential properties that are often of interest. That being said, while viewing real-valued matrices as consisting of real numbers, rather than fixed precision approximations to real numbers, as they are actually stored on a computer, is convenient, it is often not robust in the presence of even very low-order bit roundoff error. To the extent that most modern researchers who use matrix algorithms have the luxury of ignoring such issues, it is because a large body of numerical analysts and scientific computers have worried about these issues for them and have black-boxed these issues from them.

As an example of one of the many challenges in providing matrix algorithms for a wide range of very large-scale statistical data analysis applications, modern variants of these problems can re-arise either when researchers who have never thought about these issues implement nontrivial matrix algorithms in large-scale settings or also when researchers apply traditional algorithms to data matrices that are structured very differently than matrices that have arisen in the past. Different fields parameterize problems in different ways, and seemingly-minor differences can have fundamental consequences for the appropriateness and applicability of different algorithms. One of the major challenges in the area, which will be a theme that arises throughout the course, is to develop algorithmic principles that allow researchers to draw strength from the experiences of matrix computations in the past, while addressing the novel and peculiar features of matrix-based data that arise in modern massive data set applications.

This is a good point to say what is "large" in large-scale data. There is a lot of hype about large-scale and massive and big data that wasn't present even a few years ago, and so it is useful to keep particular examples and categorizations in mind as we proceed, since large means different things to different people in different contexts. One of the most useful categorizations is the following:

- **Small.** A data set is small if you can look at the data and fairly-easily find solutions to problems of interest with any one of several algorithmic tools, e.g., one's favorite method. This is common, especially in areas that focus on the development of methods qua methods.
- **Medium.** A data set is medium-sized if it fits into RAM and one can fairly-easily run computations of interest in a reasonable length of time and get answers to questions of interest. This is common, especially in areas that use methods for downstream goals of primary interest.

- **Large.** A data set is large if it doesn't easily fit into RAM and/or one can't relatively-easily run computations of interest. This is increasingly common, and although it is the domain of a relatively-small set of users of data, it provides an important forcing function in general but for the development of algorithmic and statistical methods in particular.

The main point of this informal categorization is that as one goes from medium-sized to large-scale one does not have random access to the data, and so details of memory access become increasingly important. A related issue is that communication is often a more precious resource than computation, and so this must be taken into account at the start of designing algorithms. The details of the memory access issues can vary in different application areas—e.g., streaming settings, moderate-sized databases, MapReduce or Hadoop-style environments, multi-core settings, etc. From the perspective of this class, however, we will see that often similar algorithmic ideas or algorithmic principles hold in multiple settings, basically since those ideas exploit basic structural properties of vector spaces, but that those ideas have to be instantiated in somewhat different ways depending on the particular setting. For example, a random projection algorithm might solve a given problem in an idealized sense, but one should work with versions of random projections that optimize what matters most, e.g., one should consider an Hadamard-based or Gaussian-based projection depending on whether one is interested in optimizing FLOPs or communication. The coarse algorithmic ideas are similar in both of those settings, and it's often not so useful to "over-optimize" to the details of an idealized computational model, since it can hide the breadth of applicability of a basic algorithmic idea.

Randomization can be thought about in different ways:

- Vague philosophical hope that you will find something useful if you randomly sample.
- Statistical approach: observed data are a noisy/random version of ground truth.
- Algorithmic approach: randomness is a computational resource for faster algorithms on a given observed data set.

This parallels two major perspectives on the data that will be a common theme:

- Algorithmic perspective (common in computer science theory, databases, etc.): the data/algorithms are typically discrete; do worst-case analysis on a given data set; interested in optimizing running time and other resources; models for computation and data access.
- Statistical perspective (common in statistics, machine learning, natural sciences, etc.): the data/algorithms are often continuous; make reasonable niceness assumptions on the data; ultimately interested in inferences about the world and not data per se; models for the data to help inference.

Randomization can be useful in several ways:

- Faster algorithms: worst-case theory, numerical implementation, clock time
- Simpler algorithms: to state, implement, and analyze
- More interpretable algorithms and output: select actual columns

- Implicit regularization: randomness in the algorithms helps to avoid overfitting to a given data set
- Organize algorithms to modern computational architectures better.

1.3 Examples of matrix-based data and matrix computations

If we are going to develop algorithms for data modeled as a matrix, then here are several examples of matrix-based data to keep in mind.

- **Object-feature data matrix.** This is perhaps the most common way matrices arise, in which case one has m objects or things, each of which are described by n features, e.g., term-document data, people-SNPs data, etc.
- **Correlation matrices.** This is basically $X^T X$, where X or X^T is an object-feature data matrix, perhaps appropriately normalized.
- **Kernels and similarity matrices.** These are popular in machine learning. The former are basically SPSD matrices, while the latter are basically entry-wise non-negative matrices, and there are several common procedures to go from one to the other.
- **Laplacians or Adjacency matrices of graphs.** These are central to spectral graph theory where one considers eigenvectors and eigenvalues and related quantities of matrices associated with a graph.
- **PDEs and discretization of continuum operators.** Matrices that arise here can come in one of several forms, and implementations of (low-rank, in particular) RandNLA algorithms are often developed for these matrices, but they often come with relatively strong domain-specific niceness assumptions.

And here is a motivating application (one of many, but one to which I am partial) to keep in mind as we develop and analyze algorithms over the semester. Recall that *the* human genome consists of roughly $3B$ base pairs, but every individual is distinct, and so there are differences. Of the many types of differences, perhaps the most amenable to large-scale data analysis are SNPs, which are single locations in the genome where a non-negligible fraction of the population has one base pair and a non-negligible fraction has another different base pair. Very roughly, e.g., depending on how one defined the minor-allele frequency, these SNPs occur in 1/1000 base paris, and so there are roughly $3M$ SNPs. HapMap considered roughly 400 people, and subsequent studies considered $1000s$ or $10,000s$ of people. So, we can easily get matrices of size roughly, say, $10^4 \times 10^6$. There are typically one of two goals of interest: either do population genetics, or do some sort of personalized medicine.

Among the many algorithmic/statistical challenges—for this particular motivating application, as well as much more generally—here are two prototypical examples.

- **Low-rank matrix approximation.** Do PCA or SVD to get a good low-rank approximation, and do stuff. That is, compute a full/partial SVD/PCA/QR to get a small number of eigenvectors; appeal to a model selection rule to determine the number of eigenvectors to keep; and use those eigenvectors to cluster or classify. This is no problem if the matrices are of size $10^2 \times 10^4$; it is challenging but possible if the matrices are of size $10^4 \times 10^6$; and it is essentially impossible if the matrices are of size $10^4 \times 10^8$.

- **Column subset selection problem.** Select a small number of representative SNPs, and do stuff. That is, compute a full/partial SVD/PCA/QR to get a small number of eigenvectors; appeal to a model selection rule to determine the number of eigenvectors to keep; and "interpret" those eigenvectors i.t.o. processes generating the data or use them to select actual SNPs. This is no problem if the data really are generated from a Gaussian process, since in that case the eigenvectors mean something in terms of the data, but otherwise the reification is typically no good.

There are many variants of these basic methods, e.g. "nonlinear" kernel methods, "sparse" SVD, other feature selection methods, etc.; and nearly all, either structurally or in terms of the algorithms to implement them have similar challenges to those two just outlined.

1.4 A simple model for accessing large-scale matrix data

Now, we informally define the Pass-Efficient Model, which is a computational model in which the computational resources are the number of passes over the data and the additional space and additional time required, and we use it to present several technical sampling lemmas that illustrate how to draw uniform and nonuniform samples when the data are not in RAM.

Such data streaming models have been widely-studied in recent years, especially in the theory of algorithms, and the basic idea is that the data are so large that they stream by and one must compute on the stream without storing the entire data set. The Pass-Efficient Model, in particular, is motivated by the observation that in many applications one has often the ability to generate and store very large amounts of data, but one often does not have random access to that data. For example, the data may be stored on external storage such as a tape, or the data may be stored in a distributed data center. To model this phenomenon, consider a situation in which the three scarce computational resources of interest are number of passes over the data and the additional space and time required. Although this model is quite idealized, understanding how matrix algorithms behave with respect to it will help us get beyond a vanilla RAM model in which any element of a data matrix can be accessed at random.

Definition 1 *In the* Pass-Efficient Model*, the only access an algorithm has to the data is via a pass, where a* pass *over the data is a sequential read of the entire input data set. In addition to the external storage space to store the data and to a small* number of passes *over the data, an algorithm in the Pass-Efficient Model is permitted to use* additional RAM space *and* additional computation time*. An algorithm is considered* pass-efficient *if it requires a small constant number of passes and additional space and time which are sublinear in the length of the data stream in order to compute the solution (or a "description" of the solution).*

Recall that, if the data are represented by a $m \times n$ matrix, e.g., n vectors $a_i \in \mathbb{R}^m$, $i = 1, \ldots, n$, then the data can be presented column-wise or row-wise or in some other arbitrary order. The *sparse-unordered representation* of data is a form of data representation in which each element of the data stream consists of a pair $((i, j), A_{ij})$ where the elements in the data stream may be unordered with respect to the indices (i, j) and only the nonzero elements of the matrix A need to be presented. This very general form is suited to applications where, e.g., multiple agents may write parts of a matrix to a central database and where one cannot make assumptions about the rules for write-conflict resolution. In the simplest form, the data stream read by algorithms in the Pass-Efficient Model is assumed to be presented in the *sparse-unordered representation*, but in many cases stronger results can be obtained when the data matrix is assumed to be presented, e.g., column-wise or row-wise.

Next, we present two related sampling lemmas that will be used by our subsequent algorithms. Since many of our subsequent algorithms will involve constructing random samples, from either uniform or non-uniform distributions, we would like to be able to select random samples in a pass-efficient manner. To this end, consider the SELECT algorithm, described below, which does just this. The algorithm reads a stream, assumed to consist of non-negative entries, and in constant additional space and time (where we assume that we are representing the real numbers in the stream in constant size) it selects and returns an element from that stream with a probability proportional to its size.

Algorithm 1 The SELECT Algorithm.

```
Input: $\{a_1, \dots, a_n\}$, $a_i \geq 0$, read in one pass, i.e., one sequential read, over the data.
Output: $i^*, a_{i^*}$.
1: $D = 0$.
2: for $i = 1$ to $n$ do
3:     $D = D + a_i$.
4:     With probability $a_i/D$, let $i^* = i$ and $a_{i^*} = a_i$.
5: end for
6: Return $i^*, a_{i^*}$.
```

The following lemma establishes that in one pass over the data one can sample an element according to certain probability distributions.

Lemma 1 *Suppose that* $\{a_1, \dots, a_n\}$, $a_i \geq 0$, *are read in one pass, i.e., one sequential read over the data, by the* SELECT *algorithm. Then the* SELECT *algorithm requires* $O(1)$ *additional storage space and returns* i^* *such that* $\mathbf{Pr}\left[i^* = i\right] = a_i / \sum_{i'=1}^{n} a_{i'}$.

Proof: First, note that retaining the selected value and the running sum requires $O(1)$ additional space. The remainder of the proof is by induction. After reading the first element a_1, $i^* = 1$ with probability $a_1/a_1 = 1$. Let $D_\ell = \sum_{i'=1}^{\ell} a_{i'}$ and suppose that the algorithm has read $a_1, \dots, a_\ell$ thus far and has retained the running sum D_ℓ and a sample i^* such that $\mathbf{Pr}\left[i^* = i\right] = a_i/D_\ell$. Upon reading $a_{\ell+1}$ the algorithm lets $i^* = \ell + 1$ with probability $a_{\ell+1}/D_{\ell+1}$ and retains i^* at its previous value otherwise. At that point, clearly $\mathbf{Pr}\left[i^* = \ell + 1\right] = a_{\ell+1}/D_{\ell+1}$; furthermore for $i = 1, \dots, \ell$, $\mathbf{Pr}\left[i^* = i\right] = \frac{a_i}{D_\ell}\left(1 - \frac{a_{\ell+1}}{D_{\ell+1}}\right) = \frac{a_i}{D_{\ell+1}}$. By induction this results holds when $\ell + 1 = n$ and the lemma follows.

◇

Clearly, in a single pass over the data this algorithm can be run in parallel with $O(s)$ total memory units to return s independent samples $i_1^*, \dots, i_s^*$ such that for each i_t^*, $t = 1, \dots, s$, we have $\mathbf{Pr}\left[i_t^* = i\right] = a_i / \sum_{i'=1}^{n} a_{i'}$. Also, one can clearly use this algorithm to sample with respect to other distributions that depend (or don't depend) on the a_i, e.g., the uniform distribution, probabilities proportional to a_i^2, etc.

The next lemma is a modification of the previous lemma to deal with the case where a matrix is read in the sparse-unordered representation and one wants to choose a row label with a certain probability. Note that a trivial modification would permit choosing a column label.

Lemma 2 *Suppose that $A \in \mathbb{R}^{m \times n}$, is presented in the sparse-unordered representation and is read in one pass, i.e., one sequential read over the data, by the* Select *algorithm. Then the algorithm requires $O(1)$ additional storage space and returns i^*, j^* such that $\mathbf{Pr}\left[i^* = i \wedge j^* = j\right] = A^2_{i^*j^*} / \|A\|_F^2$ and thus $\mathbf{Pr}\left[i^* = i\right] = \left\|A_{(i^*)}\right\|_2^2 / \|A\|_F^2$.*

Proof: Since $A^2_{i^*j^*} > 0$ the first claim follows from Lemma 1; the second follows since

$$\mathbf{Pr}\left[i^* = i\right] = \sum_{j=1}^{n} \mathbf{Pr}\left[i^* = i \wedge j^* = j\right] = \sum_{j=1}^{n} \frac{A^2_{i^*j^*}}{\|A\|_F^2} = \frac{\left\|A_{(i^*)}\right\|_2^2}{\|A\|_F^2}.$$

◇

Note that, in particular, this lemma implies that we can select columns and rows from a matrix according to a probability distribution that is proportional to the squared Euclidean norms, and that we can do it in two "passes" over the data. More precisely, in one pass and $O(1)$ additional space and time, we can choose choose the index of a column with a probability proportional to the Euclidean norm squared of that column (and with $O(s)$ additional space and time, we can choose the indices of s columns), and in the second pass we can pull out that column—provided, of course, that we have $O(n)$ additional space.

1.5 Quick overview of ideas and topics to be covered in the class

The ideas to be discussed in this class arose and were developed in several related research areas, and they have been applied in various forms in a wide range of theoretical and practical applications. Not surprisingly, then, there has been some reinvention of the wheel, and it can be difficult for even experts in one area to understand the contributions and developments from other areas. That being said, the ideas have already proven remarkably fruitful: they have led to qualitatively improved worst-case bounds for fundamental matrix problems, they have led to numerical implementations that beat state-of-the-art solvers, and they have led to improved machine learning and data analysis that have been used in a wide range of scientific and internet applications. In this course, we will try to distill out the basic algorithmic and statistical ideas that make these methods work. We will do so by focusing on a few very basic linear algebraic problems that underlie all or nearly all of the extensions and applications.

To illustrate this, a fundamental primitive and the first matrix problem that we will consider will be that of approximating the product of two matrices. Say that we have an $m \times n$ matrix A and an $n \times p$ matrix B, and assume that we are interested in computing the product AB.

- **Traditional perspective on matrix multiplication.** The obvious well-known way to compute the product AB is with the usual three-loop algorithm. In this case, one views an element of AB as an *inner product between a row of A and an column of B.*

- **RandNLA perspective on matrix multiplication.** A less obvious way is to view the product AB is as a sum of n terms, each of which is an *outer product between a column of A and a row of B*. Viewed this latter way, we can try to construct some sort of "sketch" of the columns of A and the rows of B—let's represent those sketches as matrices C and R, respectively—and approximate the product AB by the product CR.

If the sketches are linear, as they almost always are and as they will be in this class, then we can represent the sketches themselves as a matrix. Let's say that the $n \times c$ matrix S represents the sketching operation. Then, observe that $C = AS$ and $R = S^T B$, in which case $AB \approx CR = ASS^T B$. We will quantify the quality of CR by bounding the norm of the error, i.e., by providing an upper bound for $\left\|AB - ASS^T B\right\|_\xi$, where ξ represents some matrix norm such as the spectral or Frobenius or trace norm. This fundamental matrix multiplication primitive will appear again and again in many different guises. Of course, described this way, the sketching matrix S can be anything—deterministic or randomized, efficient or intractable to compute, etc. It turns out that if the sketches are randomized—basically consisting of random sampling or random projection operations—then in many cases we can obtain results that are "better" than with deterministic methods.

(This comment about "better" requires some clarification and comes with a number of caveats. First, the randomized methods might simply fail and return an answer that is extremely bad—after all one could flip a fair coin heads 100 times in a row—and so we will have to control for that. Second, lower bounds in general are hard to come by, and since they often say more about the computational model being considered than about the problem being considered, and they are often not robust to minor variations in problem statement or minor variations in models of data access. There are lower bounds in this area, but because of this non-robustness and because these lower bounds have yet to have impact on numerical, statistical, machine learning, or downstream scientific and internet applications of these algorithms, we will not focus on them in this class. Third, "better" might mean faster algorithms in worst-case theory, or numerical implementations that are faster in terms or wall-clock time, or implementations in parallel and distributed environments where traditional methods fail to run, or applications that are more useful to downstream scientists who tend to view these methods as black boxes. We will consider all of these notions of better throughout the class.)

The randomized sketches we will consider will come in one of two flavors: random sampling sketches, in which each column of S has one nonzero entry, which defines which (rescaled) columns of A (remember, here we are post-multiplying A by S—we won't use this convention throughout—which means that S is working on the columns of A, which also means that S^T is working on the rows of B); and random projection sketches, in which case S is dense or nearly dense and consists of i.i.d. random variables drawn from, e.g., a Gaussian distribution. The random projections are data-agnostic, in the sense that they can be constructed without looking at the data, while the random sampling typically needs to be performed in a way that identifies and extracts relevant structure in the data. Vanilla versions of both of these procedures can often be shown to perform well, in the sense that they return relatively-good answers, but they are often not faster than solving the original problem. Thus, more complex versions also exist and will be considered. For example, S could consist of the composition of a Hadamard-like rotation and a uniform sampling or sparse projection matrix. In this case, the projection or sampling quality is nearly as good as the vanilla version, while being much faster.

Thus, the main themes in RandNLA algorithms will be that we need to construct a sketch in one of two complementary ways.

- **Randomly sample.** Here, one identifies some sort of uniformity structure and use that to construct an importance sampling distribution with which to construct a random sample.
- **Randomly project.** Here, one performs a random projection which rotates to a random basis where the nonuniformity structure is uniformized and so where we can sample u.a.r.

Then, in either case, RandNLA algorithms do one of two things.

- **Solve the subproblem with a black box.** Here, the smaller subproblem is solved with a traditional black box solver. This is most appropriate for theoretical analysis and when one is not concerned with the complexity of the algorithm with respect to the ϵ error parameter.
- **Solve a preconditioned version of the original problem.** Here, one uses the sample to construct a preconditioner for the original problem and solves the original problem with a traditional preconditioned iterative method. This is most appropriate when one is interested in high precision solutions where the dependence on the ϵ error parameter is important.

The quality of the bounds will boil down to certain structural results (i.e., depending on the linear algebraic structure) and then an additional error from the random sampling process, and we will use matrix perturbation bounds to establish that those latter errors are small. The most obvious example, where we will discuss this in greatest detail, is for the least-squares problem, but the same idea holds for the low-rank approximation problem, etc. (In addition, although we will not have the time to go into it in quite as much detail, we should note that the same ideas hold not only in RAM, but also in streaming and parallel and distributed environments; we won't spend as much time on those extensions, but the basic idea will be to refrain from optimizing FLOPs and instead worry about other things like communication.)

1.6 Course announcement

For completeness, here is the description of the course from the initial coarse announcement.

Matrices are a popular way to model data (e.g., term-document data, people-SNP data, social network data, machine learning kernels, and so on), but the size-scale, noise properties, and diversity of modern data presents serious challenges for many traditional deterministic matrix algorithms. The course will cover the theory and practice of randomized algorithms for large-scale matrix problems arising in modern massive data set analysis. Topics to be covered include: underlying theory, including the Johnson-Lindenstrauss lemma, random sampling and projection algorithms, and connections between representative problems such as matrix multiplication, least-squares regression, least-absolute deviations regression, low-rank matrix approximation, etc.; numerical and computational issues that arise in practice in implementing algorithms in different computational environments; machine learning and statistical issues, as they arise in modern large-scale data applications; and extensions/connections to related problems as well as recent work that builds on the basic methods. Appropriate for graduate students in computer science, statistics, and mathematics, as well as computationally-inclined students from application domains. Here are several representative topics:

- Introduction and Overview

- Approximate Matrix Multiplication: Building Blocks and Establishing Concentration
- Random Projections: Slow, Fast, and Subspace
- Least-squares Regression: Sampling versus Projections, Low versus High Precision
- Low-rank Matrix Approximation: Additive-error, Relative-error, and Fewer Samples
- Element-wise Sampling and Applications
- Solving Square and Non-square Linear Equations
- Preserving Sparsity in Theory and in Practice
- Extensions and Applications: Kernel-based Learning; Matrix Completion; Graph Sparsification; Lp Regression and Convex Optimization; Parallel and Distributed Environments; Etc.

1.7 Update: Dec 2013: Things not covered this semester

There were several topics that I hoped to cover that we didn't have time to cover. Here is a summary of the most important.

- Sparsity-preserving Random Projections. These are very sparse but structured random projections that run in "input sparsity" time, which means proportional to the number of nonzeros plus "lower order terms," which means polynomial in the low dimension (of a rectangular problem) or the rank parameter (if both dimensions of the input are large). Here are the basic references.
  - The basic result: Clarkson and Woodruff, "Low rank approximation and regression in input sparsity time"
  - A simpler linear algebraic proof is in Section 3 of: Meng and Mahoney, "Low-distortion Subspace Embeddings in Input-sparsity Time and Applications to Robust Linear Regression"
  - A generalization of the previous result: Nelson and Nguyen, "OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings"
- Low-rank Approximation and Kernel-based Learning. The basic issue here is two-fold: whether one is interested in making claims about a given kernel (i.e., SPSD) matrix of using that kernel matrix for some sort of inferential or statistical task; and how to establish that a low-rank approximation preserves the SPSD property, which is in general hard to do unless one uses uniform sampling or has diagonally dominant input. Here are the basic references.
  - Gittens and Mahoney, "Revisiting the Nystrom Method for Improved Large-Scale Machine Learning" (the arXiv version—the arXiv version is *much* more detailed than the short ICML version)
  - Bach, "Sharp analysis of low-rank kernel matrix approximations"
- Least absolute deviations, i.e., ℓ_1, regression. This extends the basic ideas beyond ℓ_2 objectives. The following is the most comprehensive reference, but it is not easy reading.
  - Clarkson, Drineas, Magdon-Ismail, Mahoney, Meng, and Woodruff, "The Fast Cauchy Transform and Faster Robust Linear Regression"