APPM4720/5720: Fast algorithms for big data

Gunnar Martinsson

The University of Colorado at Boulder

Course objectives:

The purpose of this course is to teach efficient algorithms for processing very large datasets.

Specifically, we are interested in algorithms that are based on *low rank approximation*. Given an $m \times n$ matrix $\mathbf{A}$, it is sometimes possible to build an approximate factorization

$$\underset{m \times n}{\mathbf{A}} \approx \underset{m \times k}{\mathbf{E}} \underset{k \times n}{\mathbf{F}}.$$

In the applications we have in mind, m and n can be very large (think $m, n \sim 10^6$ for dense matrices, and much larger for sparse matrices), and $k \ll \min(m, n)$.

Factorizing matrices can be very helpful:

- Storing $\mathbf{A}$ requires mn words of storage.

  Storing $\mathbf{E}$ and $\mathbf{F}$ requires $km + kn$ words of storage.
- Given a vector $\mathbf{x}$, computing $\mathbf{Ax}$ requires mn flops.

  Given a vector $\mathbf{x}$, computing $\mathbf{Ax} = \mathbf{E}(\mathbf{Fx})$ requires $km + kn$ flops.
- The factors $\mathbf{E}$ and $\mathbf{F}$ are often useful for *data interpretation.*

Examples of data interpretation via matrix factorization:

- *Principal Component Analysis:* Form an empirical covariance matrix from some collection of statistical data. By computing the singular value decomposition of the matrix, you find the directions of maximal variance.

- *Finding spanning columns or rows:* Collect statistical data in a large matrix. By finding a set of spanning columns, you can identify some variables that "explain" the data. (Say a small collection of genes among a set of recorded genomes, or a small number of stocks in a portfolio.)

- *Relaxed solutions to k-means clustering:* Partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. Relaxed solutions can be found via the singular value decomposition.

- *PageRank:* Form a matrix representing the link structure of the internet. Determine its leading eigenvectors. Related algorithms for graph analysis.

- *Eigenfaces:* Form a matrix whose columns represent normalized grayscale images of faces. The "eigenfaces" are the left singular vectors.

Main components of the course:

- **Matrix factorization algorithms designed for a modern computing environment**
  We will discuss algorithms designed to scale correctly for very large matrices when executed on multicore processors, GPUs, parallel computers, distributed ("cloud") computers, etc. The key is to minimize *communication* and *energy consumption* (as opposed to flops). We will extensively explore new *randomized* algorithms.
- **Applications of matrix factorization**
  We will discuss a range of applications from computational statistics, machine learning, image processing, etc. Techniques include: Principal Componen Analysis, Latent Semantic Indexing, Linear Regression, PageRank, etc.
- **The idea of randomized projections for dimension reduction**
  So called *Johnson-Lindenstrauss* techniques are useful in linear algebra, algorithm design, and data analysis. We will explore mathematical properties of randomized projections, and their use in different contexts.
- **Other useful "scalable" algorihms**
  Fast Fourier Transform — an extremely versatile and powerful tool that parellelizes well. Applications to signal processing, solving PDEs, image compression, etc.
  Fast Multipole Method — another powerful and scalable algorithm with applications to solving PDEs, simulating gravitation and electrostatics, and much more.

Week:	*Material covered:*
1:	Low-rank approximation – the problem formulation and a brief survey of applications. The Singular Value Decomposition (SVD) and the Eckart-Young theorem on optimality.
2:	Power iterations, and Krylov methods. Gram-Schmidt and the QR factorization. How to cheaply get an approximate SVD from a QR factorization.
3:	Interpretation of data. The interpolative decomposition (ID) and the CUR decomposition.
4,5:	Randomized methods for computing low-rank approximations. Connection to QuickSort and Monte Carlo.
6,7:	Applications of low-rank approximation: Principal Component Analysis (PCA), Latent Semantic Indexing (LSI), eigenfaces, pagerank, potential evaluation.
8,9:	Linear regression problems. L^2 and L^1 minimization. Brief introduction to linear programming.
10:	Johnson-Lindenstrauss methods; random projections as a tool for dimensionality reduction.
11:	Nearest neighbor search.
12:	Clustering and the "k-means" problem.
13:	The Fast Fourier Transform.
14:	The Fast Multipole Method.
15:	Project presentations.

Note: This is a new course! The actual timeline may deviate from the target listed above — some topics may need more or less time.

Top Ten Algorithms of the 20th Century

1. ***Monte Carlo Method (1946):***
2. ***Simplex Method for linear programming (1947):***
3. ***Krylov methods (1950):***
4. ***Decompositional approach to matrix computations (1951):***
5. ***Fortran optimizing compiler (1957):***
6. ***QR algorithm for eigenvalues (1959):***
7. ***Quicksort (1962):***
8. ***Fast Fourier Transform (1965):***
9. ***Integer relation detection algorithm (1977):*** ???
10. ***The Fast Multipole Method (1987):***

Algorithms set in red will be discussed directly in class.

Algorithms set in blue will not be covered in any detail, but will be touched upon briefly.

From Jan./Feb. 2000 issue of Computing in Science & Engineering, as compiled by Jack Dongarra and Francis Sullivan — posted on course webpage.

Many of the algorithms we will describe are based on 20th century algorithms, but have been re-engineered to better solve 21st century problems.

Specifically, the course will extensively discuss *randomized* methods for matrix factorizations. These were developed in response to the need to process matrices for which classical algorithms were not designed.

Difficulties include:

- The matrices involved can be very large (size $1\,000\,000 \times 1\,000\,000$, say).
- Constraints on communication — few "passes" over the data.
  - Data stored in slow memory.
  - Parallel processors.
  - Streaming data.
- Lots of noise — hard to discern the "signal" from the "noise."

  High-quality denoising algorithms tend to require global operations.

  Can you trade accuracy for speed?

Question: Computers get more powerful all the time. Can't we just use the algorithms we have, and wait for computers to get powerful enough?

Question: Computers get more powerful all the time. Can't we just use the algorithms we have, and wait for computers to get powerful enough?

Observation 1: The size of the problems increase too! Tools for acquiring and storing data are improving at an even faster pace than processors.

Question: Computers get more powerful all the time. Can't we just use the algorithms we have, and wait for computers to get powerful enough?

Observation 1: The size of the problems increase too! Tools for acquiring and storing data are improving at an even faster pace than processors.

The famous "deluge" of data: documents, web searching, customer databases, hyper-spectral imagery, social networks, gene arrays, proteomics data, sensor networks, financial transactions, traffic statistics (cars, computer networks), ...

Question: Computers get more powerful all the time. Can't we just use the algorithms we have, and wait for computers to get powerful enough?

Observation 1: The size of the problems increase too! Tools for acquiring and storing data are improving at an even faster pace than processors.

The famous "deluge" of data: documents, web searching, customer databases, hyper-spectral imagery, social networks, gene arrays, proteomics data, sensor networks, financial transactions, traffic statistics (cars, computer networks), ...

Observation 2: Good algorithms are necessary. The flop count must scale close to linearly with problem size.

Question: Computers get more powerful all the time. Can't we just use the algorithms we have, and wait for computers to get powerful enough?

Observation 1: The size of the problems increase too! Tools for acquiring and storing data are improving at an even faster pace than processors.

The famous "deluge" of data: documents, web searching, customer databases, hyper-spectral imagery, social networks, gene arrays, proteomics data, sensor networks, financial transactions, traffic statistics (cars, computer networks), ...

Observation 2: Good algorithms are necessary. The flop count must scale close to linearly with problem size.

Observation 3: Communication is becoming the real bottleneck. Robust algorithms with good flop counts exist, but many were designed for an environment where you have random access to the data.

Question: Computers get more powerful all the time. Can't we just use the algorithms we have, and wait for computers to get powerful enough?

Observation 1: The size of the problems increase too! Tools for acquiring and storing data are improving at an even faster pace than processors.

The famous "deluge" of data: documents, web searching, customer databases, hyper-spectral imagery, social networks, gene arrays, proteomics data, sensor networks, financial transactions, traffic statistics (cars, computer networks), ...

Observation 2: Good algorithms are necessary. The flop count must scale close to linearly with problem size.

Observation 3: Communication is becoming the real bottleneck. Robust algorithms with good flop counts exist, but many were designed for an environment where you have random access to the data.

- *Communication speeds improve far more slowly than CPU speed and storage capacity.*
- The capacity of fast memory close to the processor is growing very slowly.
- Much of the gain in processor and storage capability is attained via parallelization. This poses particular challenges for algorithmic design.

Logistics

Prerequisites: To take this course, it is essential that you know linear algebra well. A course such as APPM3310 (Matrix Methods) or the equivalent is absolutely necessary, as we will frequently use matrix factorizations such as the SVD and the QR. Familiarity with basic probability is also important (Gaussian and Bernoulli random numbers; the concepts of expectations, standard deviations, etc). Basic numerical analysis (accuracy, stability, floating point arithmetic, etc) is assumed.

Knowledge of basic programming in Matlab is required, as is basic knowledge of analysis of algorithms such as estimating asymptotic complexity, etc.

Finally, some familiarity with Fourier methods is very helpful. For one part of the course, knowledge of basic electrostatics and the properties of the Laplace and Helmholtz equations will be assumed, but this material covers only one or two weeks, so this is not an essential pre-requisite.

Grading: There will be no regular exams. Instead, your grade will be based on projects, homeworks, etc, as follows:

- 20% for scribing.
- 30% for regular homeworks.
- 10% for the reference homework.
- 40% for a final project.

We will use a Google Spreadsheet to coordinate the scribing and the reference homeworks. You will receive an email inviting you to edit this spreadsheet in the first week. Please choose two lectures and one homework among the empty slots and enter your name.

Important: If you have not received the email invitation to edit the Google spreadsheet by Thursday, January 14, then please contact the instructor via email.

You can keep track of your scores for each component of the course that has been graded via the course D2L page. Please allow for at least 7 days after the deadline for scores to show up.

Text: The course is defined by the material covered in class, and there is no "official" text book. Course notes and links to papers discussed in class will be posted on the course website.

Lecture notes / scribing: As a participant in the course, you will be required to sign up for two lectures for which you will serve as a *scribe.* During the lecture when you are a scribe, you will take careful notes, type them up after the class (using latex if at all possible), and email them to the instructor within 48h. They will then be posted to the course webpage.

A template for the scribe notes can be downloaded from the course webpage.

Homeworks: There will be 6 homeworks, due at the end of weeks 3, 5, 7, 9, 11, and 13. Working in groups is allowed and encouraged, with the maximal group size being 3 for 4720 and 2 for 5720.

Each individual in the course (not each group!) will be required to sign up for one homework problem and be responsible for producing a "reference solution." This should be a typed solution, and should include Matlab codes where appropriate. The instructor will review the submitted reference homework, and suggest edits/corrections where appropriate. Once the reference homework is complete, it will be posted to the course webpage as a solution. You are allowed to work in your homework group to prepare the reference homework, but only one student will get credit for the problem.

Each regular homework set will be worth 5% of the grade. In addition, your reference homework problem will be worth 10%.

Project: Your grade in this course will to 40% be based on a final project. You are allowed (and encouraged!) to work in pairs on the project. Groups of three students could be allowed if the project chosen is particularly labor intensive, but this requires instructor permission.

In the last week of the course, each group is expected to deliver a brief (10 – 15 minutes) presentation of the project, and to hand in a final project report.

A number of suggested projects will be listed on the course webpage. You are also very welcome to think of projects on your own; if you want to go with this option, you need to discuss the chosen project with the instructor to get it approved. Please initiate this discussion no later than March 15, if possible.

The expectation is that the project is based on material covered in the first two thirds of the course, and will be completed during the last third. You must pick a project and notify the instructor of what your project is by March 19.