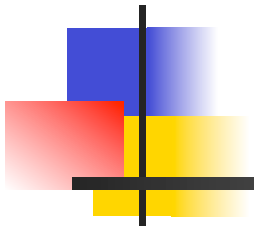


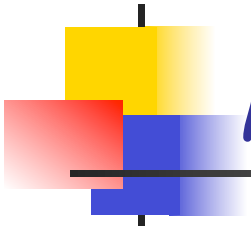
Theory (and some practice) of Randomized Algorithms for Matrices and Data

Michael W. Mahoney

Stanford University

October 2012

(For more info, see: http://cs.stanford.edu/people/mmahoney)



Matrix computations

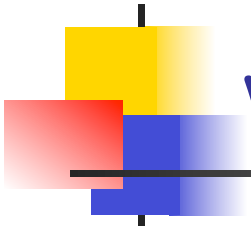
Eigendecompositions, QR, SVD, least-squares, etc.

Traditional algorithms:

- compute "exact" answers to, say, 10 digits as a black box
- assume the matrix is in RAM and minimize flops

But they are NOT well-suited for:

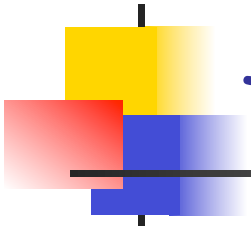
- with missing or noisy entries
- problems that are very large
- distributed or parallel computation
- when communication is a bottleneck
- when the data must be accessed via "passes"



Why randomized matrix algorithms?

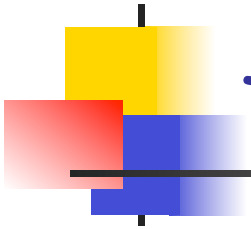
- Faster algorithms: worst-case theory and/or numerical code
- Simpler algorithms: easier to analyze and reason about
- More-interpretable output: useful if analyst time is expensive
- Implicit regularization properties: and more robust output
- Exploit modern computer architectures: by reorganizing steps of alg
- Massive data: matrices that they can be stored only in slow secondary memory devices or even not at all

Today, mostly (but not exclusively) focus on **low-rank matrix approximation** and **least-squares approximation**: ubiquitous, fundamental, and at the center of recent developments



The general idea …

- Randomly sample columns/rows/entries of the matrix, with carefully-constructed *importance sampling probabilities*, to form a randomized sketch
- Preprocess the matrix with random projections, to form a randomized sketch by sampling columns/rows *uniformly*
- Use the sketch to compute an approximate solution to the original problem w.h.p.
- Resulting sketches are "similar" to the original matrix in terms of singular value and singular vector structure, e.g., w.h.p. are bounded distance from the original matrix



The devil is in the details …

Decouple the randomization from the linear algebra:

- originally within the analysis, then made explicit
- permits much finer control in application of randomization

Importance of statistical leverage scores:

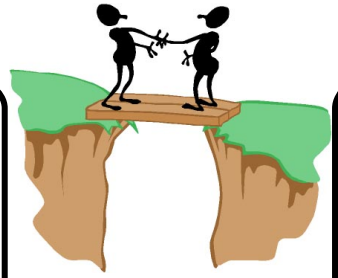
- historically used in regression diagnostics to identify outliers
- best random sampling algorithms use them as importance sampling distribution
- best random projection algorithms go to a random basis where they are roughly uniform

Couple with domain expertise—to get best results!

History of Randomized Matrix Algs

Theoretical origins

- theoretical computer science, convex analysis, etc.
- Johnson-Lindenstrauss
- Additive-error algs
- Good worst-case analysis
- No statistical analysis

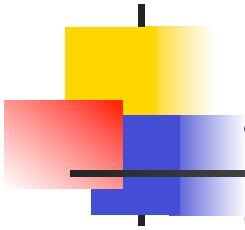


Practical applications

- NLA, ML, statistics, data analysis, genetics, etc
- Fast JL transform
- Relative-error algs
- Numerically-stable algs
- Good statistical properties

How to "bridge the gap"?

- decouple randomization from linear algebra
- importance of statistical leverage scores!



Statistical leverage, coherence, etc.

Mahoney and Drineas (2009, PNAS); Drineas, Magdon-Ismail, Mahoney, and Woodruff (2012, ICML)

Definition: Given a "tall" n x d matrix A, i.e., with n > d, let U be *any* n x d orthogonal basis for span(A), & let the d-vector $U_{(i)}$ be the i^{th} *row* of U. Then:

- the statistical leverage scores are $\lambda_i = ||U_{(i)}||_2^2$, for $i \in \{1,\ldots,n\}$
- the coherence is $\gamma = \max_{i \in \{1,\ldots,n\}} \lambda_i$
- the (i,j)-cross-leverage scores are $U_{(i)}^T U_{(j)} = \langle U_{(i)}, U_{(j)} \rangle$

Note: There are extension of this to:

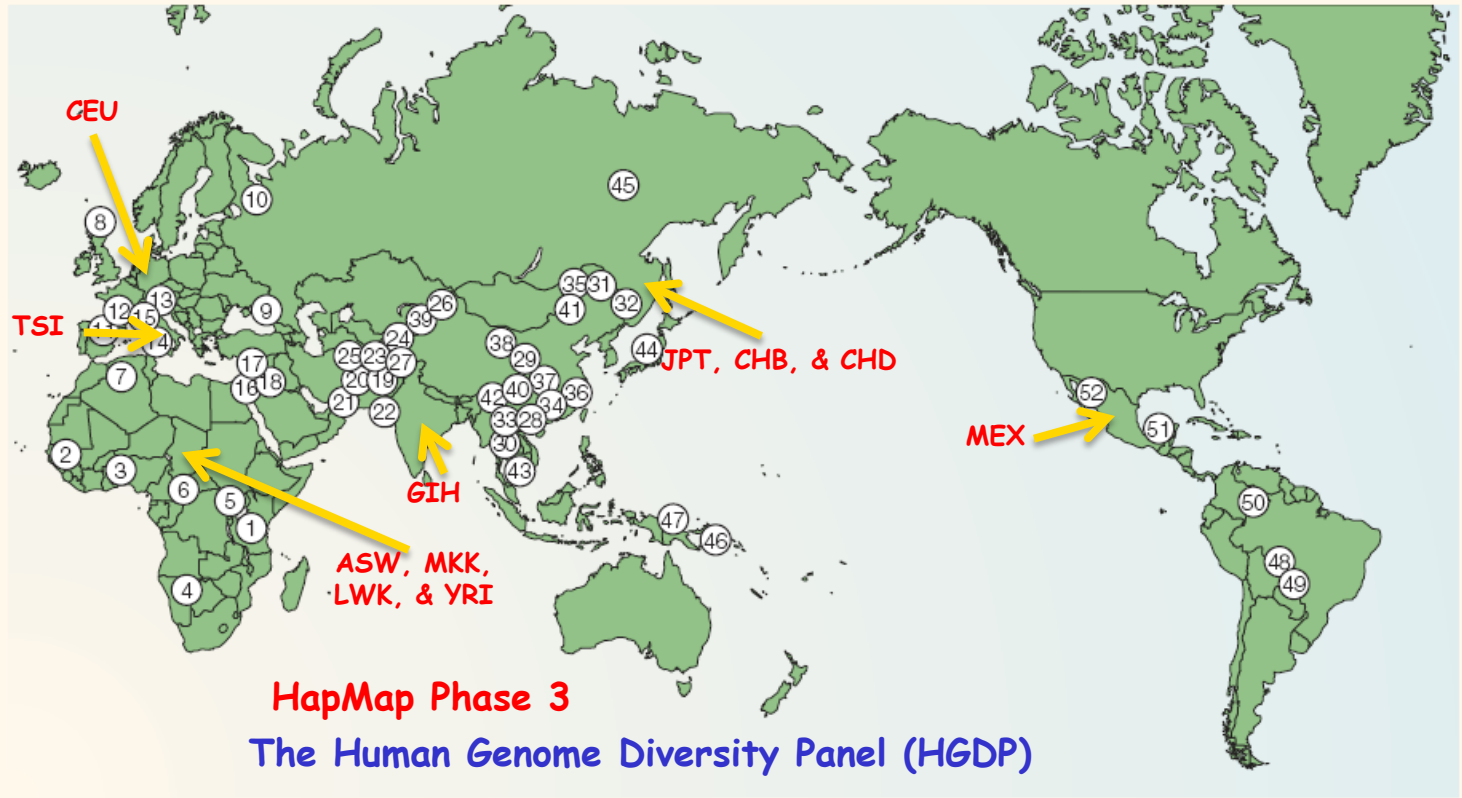
- "fat" matrices A, with n, d are large and low-rank parameter k
- L1 and other p-norms

Applications in: Human Genetics

Single Nucleotide Polymorphisms: the most common type of genetic variation in the genome across different individuals.

They are known locations at the human genome where two alternate nucleotide bases (alleles) are observed (out of A, C, G, T).

Matrices including thousands of individuals and hundreds of thousands if SNPs are available.



Africans
1 Bantu
2 Mandenka
3 Yoruba
4 San
5 Mbuti pygmy
6 Biaka
7 Mozabite

Europeans
8 Orcadian
9 Adygei
10 Russian
11 Basque
12 French
13 North Italian
14 Sardinian
15 Tuscan

Western Asians
16 Bedouin
17 Druze
18 Palestinian

Central and Southern Asians
19 Balochi
20 Brahui
21 Makrani
22 Sindhi
23 Pathan
24 Burusho
25 Hazara
26 Uygur
27 Kalash

Eastern Asians
28 Han (S. China)
29 Han (N. China)
30 Dai
31 Daur
32 Hezhen
33 Lahu
34 Miao
35 Oroqen
36 She
37 Tujia
38 Tu
39 Xibo
40 Yi
41 Mongola
42 Naxi
43 Cambodian
44 Japanese
45 Yakut

Oceanians
46 Melanesian
47 Papuan

Native Americans
48 Karitiana
49 Surui
50 Colombian
51 Maya
52 Pima

Cavalli-Sforza (2005) *Nat Genet Rev*

Rosenberg et al. (2002) *Science*

Li et al. (2008) *Science*

The International HapMap Consortium (2003, 2005, 2007) *Nature*

HGDP data

- 1,033 samples
- 7 geographic regions
- 52 populations

HapMap Phase 3 data

- 1,207 samples
- 11 populations

Apply SVD/PCA on the (joint) HGDP and HapMap Phase 3 data.

Matrix dimensions:

2,240 subjects (rows)

447,143 SNPs (columns)

Dense matrix:

over one billion entries

The Singular Value Decomposition (SVD)

The formal definition:

Given any m x n matrix A, one can decompose it as:

$$\underset{m \times n}{A} = \underset{m \times \rho}{U} \cdot \underset{\rho \times \rho}{\Sigma} \cdot \underset{\rho \times n}{V^T}$$

ρ: rank of A

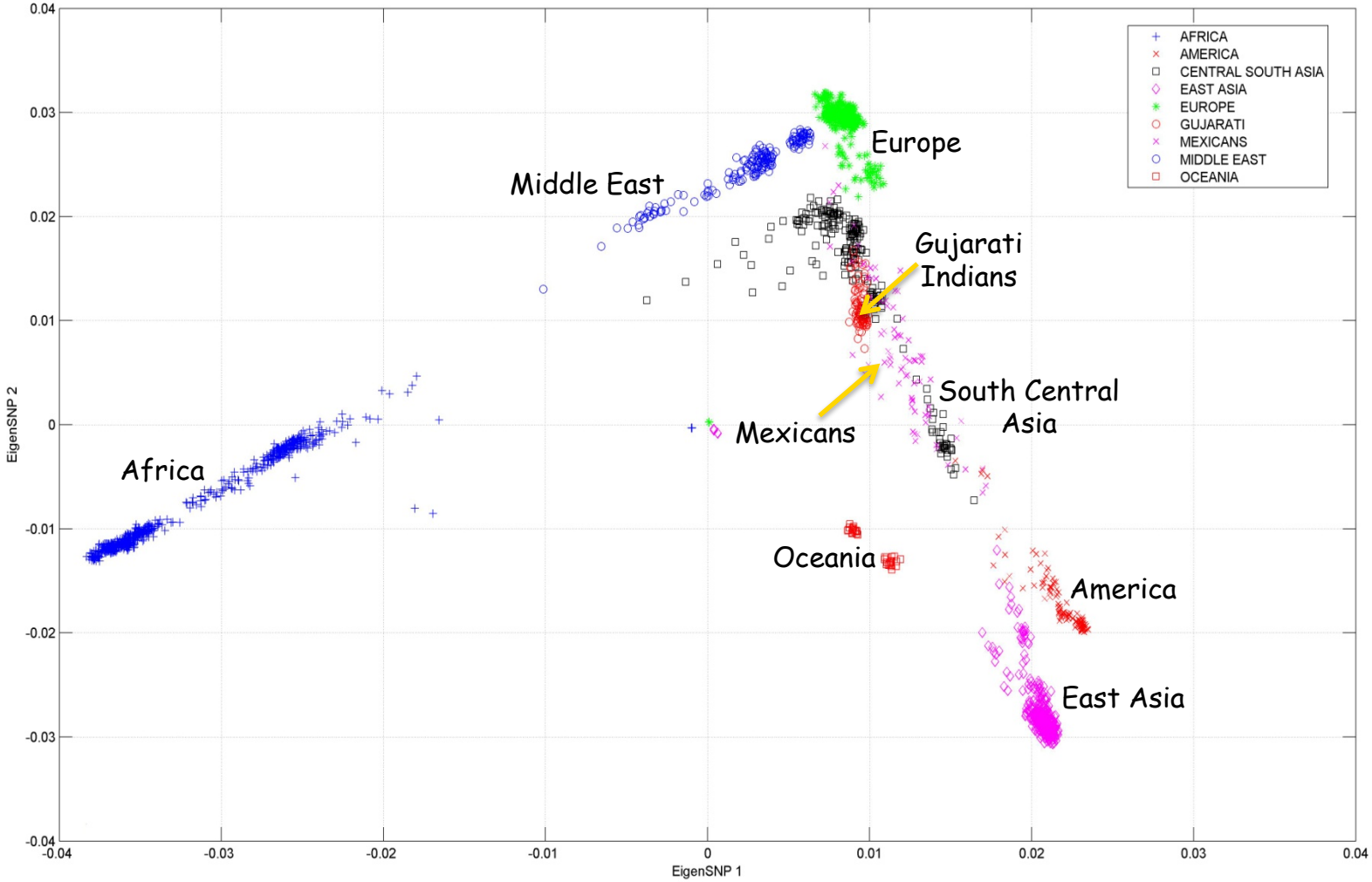
U (V): orthogonal matrix containing the left (right) singular vectors of A.

Σ: diagonal matrix containing $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_\rho$, the singular values of A.

Important: Keeping top k singular vectors provides "best" rank-k approximation to A (w.r.t. Frobenius norm, spectral norm, etc.):

$$A_k = \text{argmin}\{ ||A-X||_{2,F} : \text{rank}(X) \leq k \}.$$

Paschou, Lewis, Javed, & Drineas (2010) J Med Genet

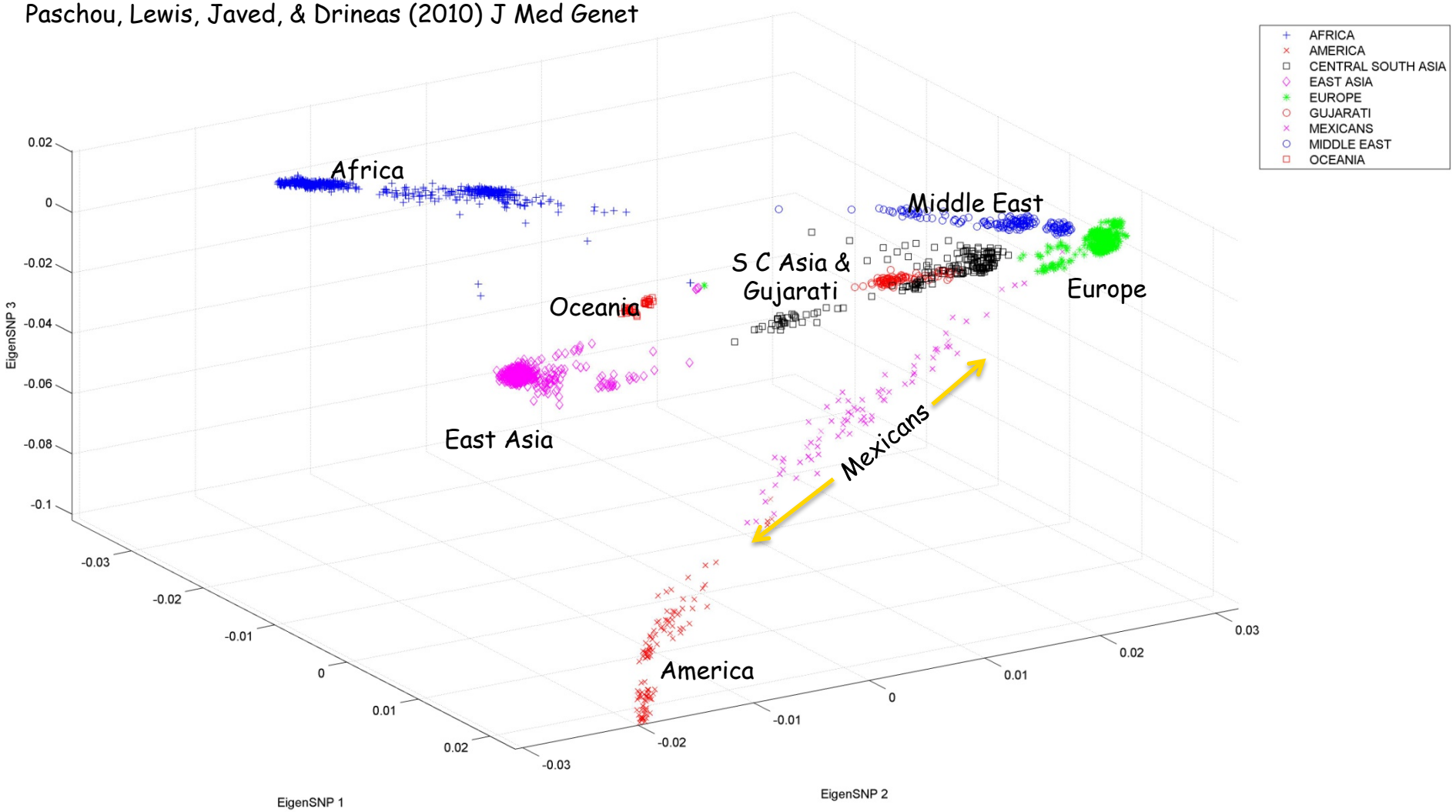


- Top two Principal Components (PCs or eigenSNPs)

(Lin and Altman (2005) *Am J Hum Genet*)

- The figure renders visual support to the "out-of-Africa" hypothesis.
- Mexican population seems out of place: we move to the top three PCs.

Paschou, Lewis, Javed, & Drineas (2010) J Med Genet

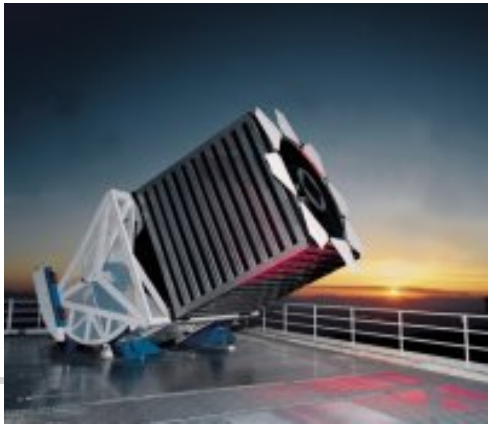


Not altogether satisfactory: the principal components are linear combinations of all SNPs, and – of course – can not be assayed!

Can we find **actual SNPs** that capture the information in the singular vectors?

Formally: spanning the same subspace.

Applications in: Astronomy



Szalay (2012, MMDS)

CMB Surveys (pixels)

1990	COBE	1000
2000	Boomerang	10,000
2002	CBI	50,000
2003	WMAP	1 Million
2008	Planck	10 Million

Angular Galaxy Surveys (obj)

1970	Lick	1M
1990	APM	2M
2005	SDSS	200M
2011	PS1	1000M
2020	LSST	30000M

Time Domain

- QUEST
- SDSS Extension survey
- Dark Energy Camera
- Pan-STARRS
- LSST…

Galaxy Redshift Surveys (obj)

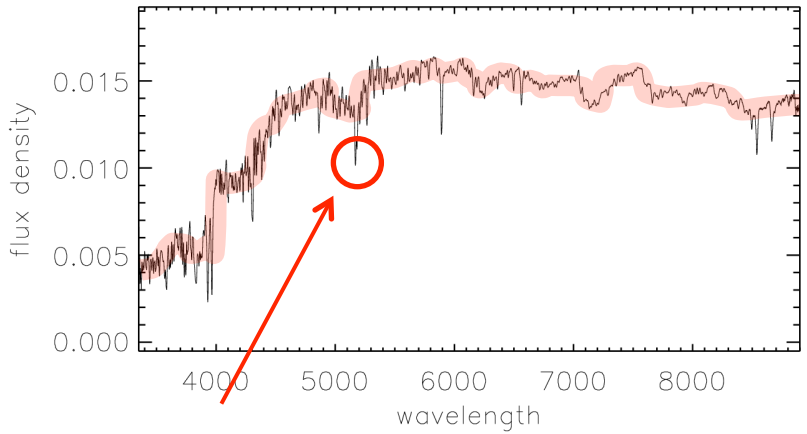
1986	CfA	3500
1996	LCRS	23000
2003	2dF	250000
2008	SDSS	1000000
2012	BOSS	2000000
2012	LAMOST	2500000

"The Age of Surveys" – generate petabytes/year …

Galaxy properties from galaxy spectra

Szalay (2012, MMDS)

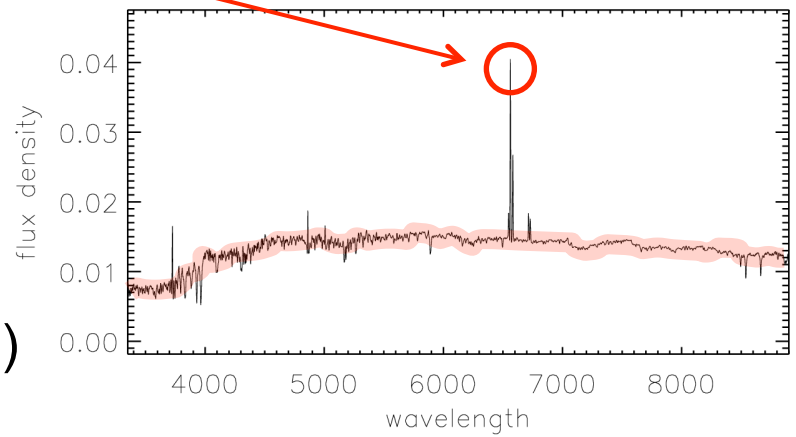
4K x 1M SVD Problem: ideal for randomized matrix algorithms



Spectral Lines

Continuum Emissions

Can we select "informative" frequencies (columns) or images (rows) "objectively"?



Galaxy diversity from PCA

PC

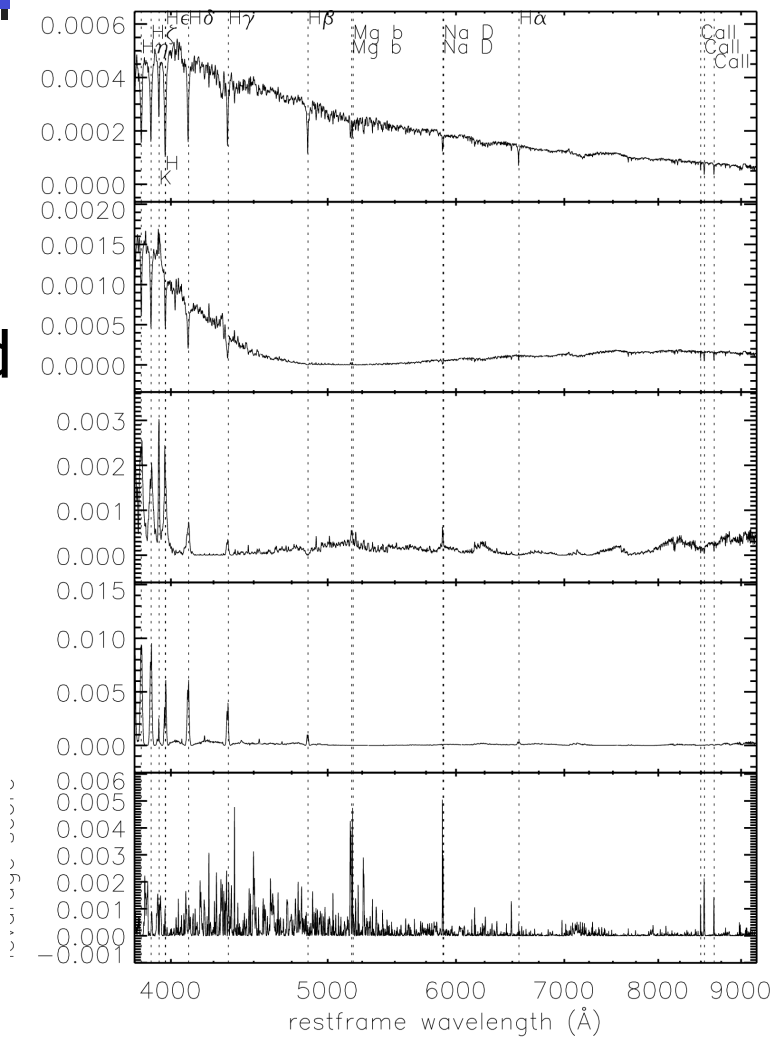
1st

2nd

3rd

4th

5th



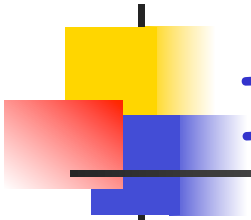
[Average Spectrum]

[Stellar Continuum]

[Finer Continuum Features + Age]

[Age]
Balmer series hydrogen lines

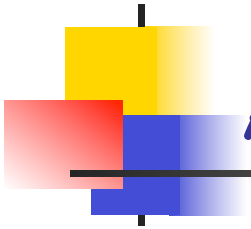
[Metallicity]
Mg b, Na D, Ca II Triplet



Issues with eigen-analysis

- **Computing large SVDs: computational time**
  - In commodity hardware (e.g., a 4GB RAM, dual-core laptop), using MatLab 7.0 (R14), the computation of the SVD of the dense 2,240-by-447,143 matrix A takes about 20 minutes.
  - Computing this SVD is not a one-liner, since we can not load the whole matrix in RAM (runs out-of-memory in MatLab).
  - Instead, compute the SVD of AA^T.
  - In a similar experiment, compute **1,200 SVDs** on matrices of dimensions (approx.) 1,200-by-450,000 (roughly, a full leave-one-out cross-validation experiment).

    (e.g., Drineas, Lewis, & Paschou (2010) PLoS ONE)
- **Selecting *actual columns* that "capture the structure" of the top PCs**
  - Combinatorial optimization problem; hard even for small matrices.
  - Often called the Column Subset Selection Problem (CSSP).
  - Not clear that such "good" columns even exist.
  - Avoid "reification" problem of "interpreting" singular vectors!



Applications in: Real Implementations

In **RAM**:

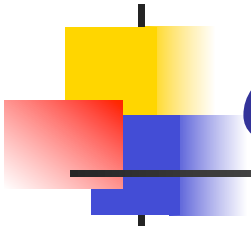
- Compute answers to 1 digit or 10 digits faster than LAPACK
- (TCS theory NOT *directly* relevant here – high-precision issues.)

For **very large vector space analytics**

- *Small-scale data*: model by graphs and matrices, and compute eigenvectors/ correlations
- *Large-scale data*: model with flat table and relational model, and compute with join/select and other "counting" procedures
- (TCS theory NOT *directly* relevant here – communication issues)

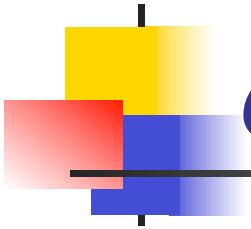
Can we "bridge the gap"

- between TCS theory and numerical implementations
- between "vector-space computations" and "very large-scale analytics"?



Outline

- Background, motivation, and applications
- Algorithms (in RAM) for least-squares approximation
- Algorithms (in RAM) for low-rank approximation
- Implementation of L1 and L2 regression in parallel and distributed environments



Outline

- Background, motivation, and applications
- **Algorithms (in RAM) for least-squares approximation**
- Algorithms (in RAM) for low-rank approximation
- Implementation of L1 and L2 regression in parallel and distributed environments

Least Squares (LS) Approximation

$$\underset{n \times d}{\begin{pmatrix} & \\ & A \\ & \end{pmatrix}} \begin{pmatrix} \\ \hat{x} \\ \\ \end{pmatrix} \approx \begin{pmatrix} \\ b \\ \\ \end{pmatrix} \quad , \quad n \gg d$$

$$\begin{aligned} \mathcal{Z}_2 &= \min_{x \in \mathbb{R}^d} ||b - Ax||_2 \\ &= ||b - A\hat{x}||_2 \end{aligned}$$

We are interested in over-constrained Lp regression problems, $n \gg d$.

Typically, there is no x such that $Ax = b$.

Want to find the "best" x such that $Ax \approx b$.

Ubiquitous in applications & central to theory:

Statistical interpretation: best linear unbiased estimator.

Geometric interpretation: orthogonally project b onto span(A).

Exact solution to LS Approximation

Cholesky Decomposition:

If A is full rank and well-conditioned,

decompose $A^TA = R^TR$, where R is upper triangular, and

solve the normal equations: $R^TRx=A^Tb$.

QR Decomposition:

Slower but numerically stable, esp. if A is rank-deficient.

Write $A=QR$, and solve $Rx = Q^Tb$.

Singular Value Decomposition:

Most expensive, but best if A is very ill-conditioned.

Write $A=U\Sigma V^T$, in which case: $\mathbf{x_{OPT}} = \mathbf{A^+b} = \mathbf{V\Sigma^{-1}_kU^Tb}$.

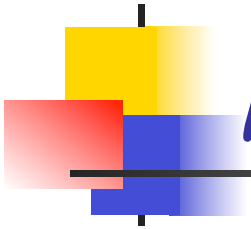
Complexity is $O(nd^2)$ for all of these, but constant factors differ.

$$\begin{aligned} \mathcal{Z}_2 &= \min_{x \in R^d} ||b - Ax||_2 \\ &= ||b - A\hat{x}||_2 \end{aligned}$$

Projection of b on the subspace spanned by the columns of A

$$\begin{aligned} \mathcal{Z}_2^2 &= ||b||_2^2 - ||AA^+b||_2^2 \\ \hat{x} &= A^+b \end{aligned}$$

Pseudoinverse of A

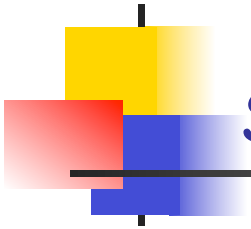


Modeling with Least Squares

Assumptions underlying its use:

- Relationship between "outcomes" and "predictors is (roughly) linear.
- The error term ε has mean zero.
- The error term ε has constant variance.
- The errors are uncorrelated.
- The errors are normally distributed (or we have adequate sample size to rely on large sample theory).

Should always check to make sure these assumptions have not been (too) violated!



Statistical Issues and Regression Diagnostics

Model: $b = Ax+\varepsilon$ — b = response; $A^{(i)}$ = carriers;

ε = error process s.t.: mean zero, const. varnce, (i.e., E(e)=0 and Var(e)=$\sigma^2 I$), uncorrelated, normally distributed

$x_{opt} = (A^T A)^{-1} A^T b$ — (what we computed before)

$b' = Hb$ — $H = A(A^T A)^{-1} A^T$ = "hat" matrix

H_{ij} - measures the leverage or influence exerted on b'_i by b_j, *regardless* of the value of b_j (since H depends only on A)

$e' = b - b' = (I-H)b$ — vector of residuals - note: E(e')=0, Var(e')=$\sigma^2(I-H)$

Trace(H)=d — Diagnostic Rule of Thumb: Investigate if $H_{ii} > 2d/n$

$H=UU^T$ — U is from SVD ($A=U\Sigma V^T$), or *any* orthogonal matrix for span(A)

$H_{ii} = |U^{(i)}|_2^2$ — leverage scores = row "lengths" of spanning orthogonal matrix

A "classic" randomized algorithm (1of3)

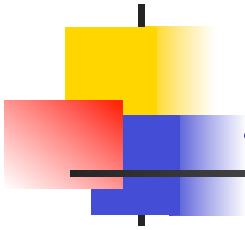
Drineas, Mahoney, and Muthukrishnan (2006, SODA & 2008, SIMAX)

Over-constrained least squares (n x d matrix A,n >>d)

- Solve: $\mathcal{Z} = \min_{x \in R^d} ||Ax - b||_2$
- Solution: $x_{opt} = A^{\dagger} b$

Randomized Algorithm:

- For all i ε {1,…,n}, compute $p_i = \frac{1}{d}||U_{(i)}||_2^2$
- Randomly sample O(d log(d)/ ε) rows/elements fro A/b, using *{p_i} as importance sampling probabilities.*
- Solve the induced subproblem: $\tilde{x}_{opt} = (SA)^{\dagger} Sb$



A "classic" randomized algorithm (2of3)

Drineas, Mahoney, and Muthukrishnan (2006, SODA & 2008, SIMAX)

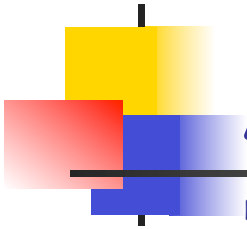
Theorem: Let $\gamma = ||U_A U_A^T b||_2 / ||b||_2$. Then:

- $||A\tilde{x}_{opt} - b||_2 \leq (1+\epsilon)\mathcal{Z}$
- $||x_{opt} - \tilde{x}_{opt}||_2 \leq \sqrt{\epsilon}\left(\kappa(A)\sqrt{\gamma^{-2}-1}\right)||x_{opt}||_2$

This naïve algorithm runs in $O(nd^2)$ time

- But it can be improved !!!

This algorithm is bottleneck for Low Rank Matrix Approximation and many other matrix problems.



A "classic" randomized algorithm (3of3)

Drineas, Mahoney, and Muthukrishnan (2006, SODA & 2008, SIMAX)

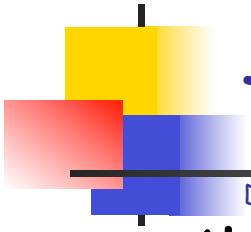
Sufficient condition for relative-error approximation.

For the "preprocessing" matrix X:

$$\sigma^2_{min}\left(XU_A\right) \geq 1/\sqrt{2}; \text{ and}$$
$$||U_A^T X^T X b^{\perp}||_2^2 \leq \epsilon \mathcal{Z}^2/2,$$

• *Important: this condition decouples the randomness from the linear algebra.*

• Random sampling algorithms with leverage score probabilities and random projections satisfy it!



Theoretically "fast" algorithms

Drineas, Mahoney, Muthukrishnan, and Sarlos (2007); Drineas, Magdon-Ismail, Mahoney, and Woodruff (2011)

Algorithm 1: Fast Random Projection Algorithm for LS Problem

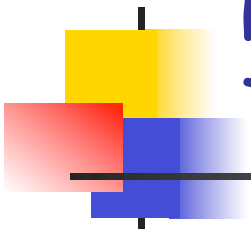
- Preprocess input (in $o(nd^2)$time) with Fast-JL transform, uniformizes leverage scores, and sample uniformly in the randomly-rotated space
- Solve the induced subproblem

Algorithm 2: Fast Random Sampling Algorithm for LS Problem

- Compute $1\pm\varepsilon$ approximation to statistical leverage scores (in $o(nd^2)$ time), and use them as importance sampling probabilities
- Solve the induced subproblem

Main theorem: For both of these randomized algorithms, we get:

- *$(1\pm\varepsilon)$-approximation*
- *in roughly* $O\left(nd\log\left(d\log(n)/\epsilon\right)+d^3\log(n)\log(d\log n)/\epsilon\right)$ *time!!*



Random Projections (1 of 4): The Johnson-Lindenstrauss lemma

For every set S of m points in $\mathbb{R}^n$ and every $\epsilon > 0$, there exists a mapping $f : \mathbb{R}^n \to \mathbb{R}^s$, where $s = O\left(\log m/\epsilon^2\right)$, such that for all points $u \in S$,

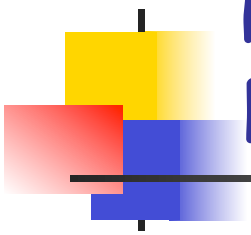
$$(1-\epsilon)\ \|u\|_2 \leq\ \|f(u)\|_2 \leq (1+\epsilon)\ \|u\|_2$$

holds with probability at least $1 - 1/m^2$.

Johnson & Lindenstrauss (1984)

- We can represent S by an m-by-n matrix A, whose rows correspond to points.
- We can represent all f(u) by an m-by-s $\tilde{A}$.
- The "mapping" corresponds to the construction of an n-by-s matrix Ω and computing

$$\tilde{A} = A\ \Omega$$



Random Projections (2 of 4): Different constructions for Ω matrix

"Slow" Random Projections ($\geq O(nd^2)$ time to implement in RAM model):

- JL (1984): random k-dimensional space
- Frankl & Maehara (1988): random orthogonal matrix
- DasGupta & Gupta (1999): random matrix with entries from N(0,1), normalized
- Indyk & Motwani (1998): random matrix with entries from N(0,1), normalized
- Achlioptas (2003): random matrix with entries in {-1,0,+1}, normalized
- Alon (2003): optimal dependency on n, and almost optimal dependency on ε

"Fast" Random Projections ($o(nd^2)$ time to implement in RAM model):

- Ailon and Chazelle (2006,2009); Matousek (2008); and many variants more recently.

Random Projections (3 of 4): Fast Johnson-Lindenstrauss Transform

Facts implicit or explicit in: Ailon & Chazelle (2006), Ailon and Liberty (2008), and Matousek(2008).

$$P \in \mathbb{R}^{s \times n}$$

$$s = O\left(\log m/\epsilon^2\right)$$

$$P_{ij} = \sqrt{q} \times \begin{cases} +1 & \text{,w.p. } q/2 \\ 0 & \text{,w.p. } 1\text{-q} \\ -1 & \text{,w.p. } q/2 \end{cases}$$

$$q = O\left(\log^2 m\right)$$

$$H \in \mathbb{R}^{n \times n}$$

Normalized Hadamard-Walsh transform matrix

(if n is not a power of 2, add all-zero columns to A; or use other related Hadamard-based methods)

$$D \in \mathbb{R}^{n \times n}$$

Diagonal matrix with D_{ii} set to +1 or -1 w.p. 1/2.

$$R = (PHD)^T \in \mathbb{R}^{n \times s} \longrightarrow \tilde{A} = \frac{1}{\sqrt{s}} AR$$

- P can also be a matrix representing the "uniform sampling" operation.
- In both cases, the O(n log (n)) running time is computational bottleneck.

Random Projections (4 of 4): Randomized Hadamard preprocessing

Facts implicit or explicit in: Ailon & Chazelle (2006), Ailon and Liberty (2008), and Matousek(2008).

Let H_n be an *n-by-n* deterministic Hadamard matrix, and
Let D_n be an *n-by-n* random diagonal matrix with +1/-1 chosen u.a.r. on the diagonal.

Fact 1: Multiplication by H_nD_n doesn't change the solution:

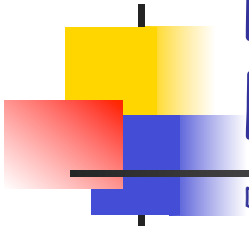
$$||Ax - b||_2 = ||H_nD_nAx - H_nD_nb||_2 = ||\mathcal{H}Ax - \mathcal{H}b||_2$$

(since H_n and D_n are orthogonal matrices).

Fact 2: Multiplication by H_nD_n is fast - only *O(n log(r)) time*, where r is the number of elements of the output vector we need to "touch".

Fact 3: Multiplication by H_nD_n approximately uniformizes *all leverage scores*:

$$||U_{(i)_{\mathcal{H}A}}||_2 = ||(\mathcal{H}U_A)_{(i)}||_2 \leq O\left(\sqrt{\frac{d\log n}{n}}\right)$$



Fast approximation of statistical leverage and matrix coherence (1 of 4)

Drineas, Magdon-Ismail, Mahoney, and Woodruff (2011, arXiv)

Simple (deterministic) algorithm:

- Compute a basis Q for the left singular subspace, with QR or SVD.
- Compute the Euclidean norms of the *rows* of Q.

Running time is $O(nd^2)$, if n >> d, O(on-basis) time otherwise.

We want *faster*!

- *$o(nd^2)$ or o(on-basis), with no assumptions on input matrix A.*
- Faster in terms of flops of clock time for not-obscenely-large input.
- OK to live with ε-error or to fail with overwhelmingly-small δ probability

Fast approximation of statistical leverage and matrix coherence (2 of 4)

Drineas, Magdon-Ismail, Mahoney, and Woodruff (2011, arXiv)

View the computation of leverage scores i.t.o an under-constrained LS problem

Recall (A is n x d, n » d):

- $\min_{x \in R^n} ||x^T A - e_i A||_2^2 \quad \rightarrow \quad x^T = e_i A A^\dagger$

But:

- $p_i = ||e_i U_A||_2^2 = ||e_i U_A U_A^T||_2^2 = ||e_i A A^\dagger||_2^2$

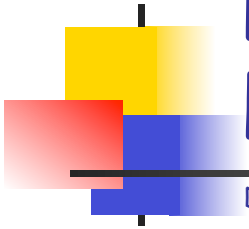
Leverage scores are the norm of a min-length solution of an under-constrained LS problem!

Fast approximation of statistical leverage and matrix coherence (3 of 4)

Drineas, Magdon-Ismail, Mahoney, and Woodruff (2011, arXiv)

$$
\begin{aligned}
p_i &= ||(AA^\dagger)_{(i)}||_2^2 \\
&\approx ||(A(\Omega_1 A)^\dagger)_{(i)}||_2^2 \quad \text{where } \Omega_1 \text{ is a fast SRHT} \\
&\approx ||(A(\Omega_1 A)^\dagger \Omega_2)_{(i)}||_2^2 \quad \text{where } \Omega_2 \text{ is Rand Proj}
\end{aligned}
$$

- This is simpler than for the full under-constrained LS solution since only need the norm of the solution.
- This is essentially using R^{-1} from QR of subproblem as preconditioner for original problem.
- I.e., $\Omega_1 A$ is a randomized "sketch" of A; QR = $\Omega_1 A$ is QR decomposition of this sketch; and evaluate row norms of $X \approx A R^{-1}$., but need Ω_2, a second projection, to make it "fast."



Fast approximation of statistical leverage and matrix coherence (4 of 4)

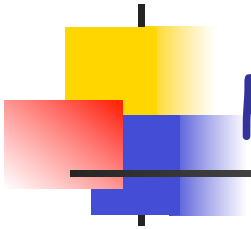
Drineas, Magdon-Ismail, Mahoney, and Woodruff (2011, arXiv)

Theorem: Given an n x d matrix A, with n >> d, let P_A be the projection matrix onto the column space of A. Then , there is a randomized algorithm that w.p. ≥ 0.999:

- computes *all* of the n diagonal elements of P_A (i.e., leverage scores) to within relative (1±ε) error;
- computes *all the large* off-diagonal elements of P_A to within additive error;
- runs in *$o(nd^2)$* time.*

*Running time is basically O(n d log(n)/ε), i.e., same as DMMS fast randomized algorithm for over-constrained least squares.

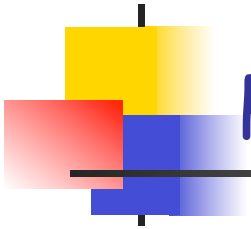
Note: Clarkson-Woodruff (2012) can compute these in "input sparsity" time!



Practically "fast" implementations (1of2)

Use "randomized sketch" to construct preconditioner for traditional iterative methods:

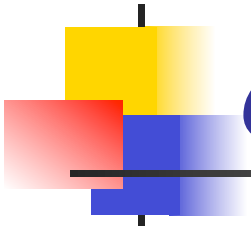
- RT08: preconditioned iterative method improves $1/\varepsilon$ dependence to $\log(1/\varepsilon)$, important for high precision
- AMT10: much more detailed evaluation, different Hadamard-type preconditioners, etc.
- CRT11: use Gaussian projections to compute orthogonal projections with normal equations
- MSM11: use Gaussian projections and LSQR or Chebyshev semi-iterative method to minimize communication, e.g., for parallel computation in Amazon EC2 clusters!



Practically "fast" implementations (2of2)

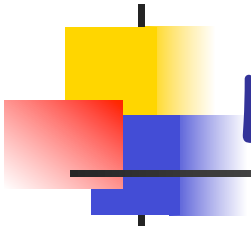
Avron, Maymounkov, and Toledo 2010:

- *Blendenpik "beats Lapack's direct dense least-squares solver by a large margin on essentially any dense tall matrix"*
- *Empirical results "show the potential of random sampling algorithms and suggest that random projection algorithms should be incorporated into future versions of Lapack."*



Outline

- Background, motivation, and applications
- Algorithms (in RAM) for least-squares approximation
- **Algorithms (in RAM) for low-rank approximation**
- Implementation of L1 and L2 regression in parallel and distributed environments

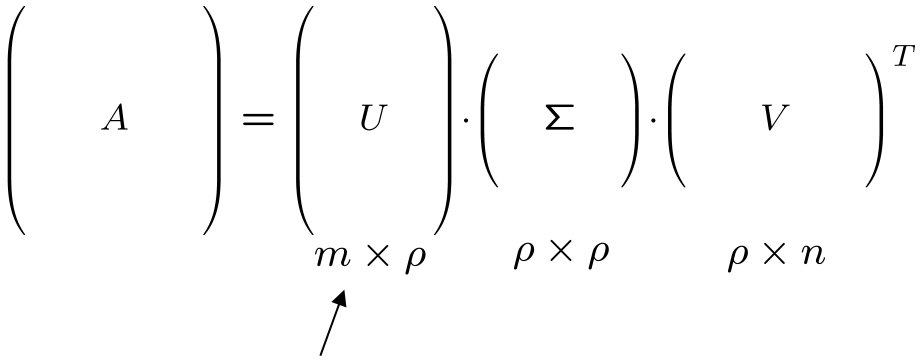


Low-rank approximation algorithms

Many randomized algorithms for low-rank matrix approximation use extensions of these basic least-squares ideas:

- Relative-error random sampling CX/CUR algorithms (DMM07)
- Relative-error random projection algorithms (S08)
- Column subset selection problem (exactly k columns) (BMD09)
- Numerical implementations, with connections to interpolative decomposition (LWMRT07,WLRT08,MRT11)
- Numerical implementations for slower spectral decay (RST09)

Recall, SVD decomposes a matrix as …

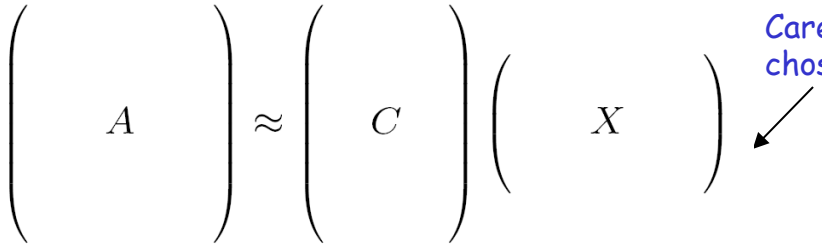


The SVD has very strong optimality properties., e.g. the matrix U_k is the "best" in many ways.

- Note that, given U_k, the best $X = U_k^T A = \Sigma V^T$.
- SVD can be computed fairly quickly.
- The columns of U_k are linear combinations of up to all columns of A.

CX (and CUR) matrix decompositions

Mahoney and Drineas (2009, PNAS); Drineas, Mahoney, and Muthukrishnan (2008, SIMAX)



Goal: choose actual columns C to make (some norm) of A-CX small.

Why?

If A is a subject-SNP matrix, then selecting representative columns is equivalent to selecting representative SNPs to capture the same structure as the top eigenSNPs.

If A is a frequency-image astronomical matrix, then selecting representative columns is equivalent to selecting representative frequencies/wavelengths.

Note: To make C small, we want c as small as possible!

CX (and CUR) matrix decompositions

Mahoney and Drineas (2009, PNAS); Drineas, Mahoney, and Muthukrishnan (2008, SIMAX)

$$\begin{pmatrix} \\ A \\ \\ \end{pmatrix} \approx \begin{pmatrix} \\ C \\ \\ \end{pmatrix} \begin{pmatrix} \\ X \\ \\ \end{pmatrix}$$

c columns of A

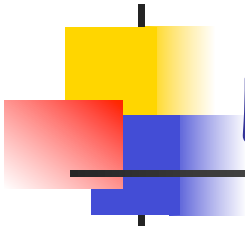
Easy to see optimal $X = C^{+}A$.

Hard to find good columns (e.g., SNPs) of A to include in C.

This Column Subset Selection Problem (CSSP), heavily studied in N LA, is a hard combinatorial problem.

Two issues are connected

- There exist (1+ε) "good" columns in any matrix that contain information about the top principal components: $||A-CX||_F \leq (1+\varepsilon)\ ||A-A_k||_F$
- We can identify such columns via a simple statistic: **the leverage scores.**
- This does not immediately imply faster algorithms for the SVD, but, combined with random projections, it does!
- Analysis (almost!) boils down to understanding least-squares approximation.



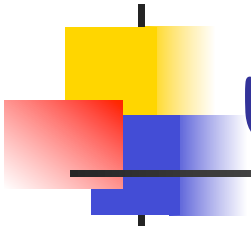
Low-rank structural condition

Boutsidis, Mahoney and Drineas (2009, SODA)

- **Structural condition underlying the randomized low-rank algorithm.** If $V_k^T Z$ has full rank, then for $\nu \in \{2, F\}$, *i.e.*, for both the Frobenius and spectral norms,

$$||A - P_{AZ}A||_\nu^2 \leq ||A - A_k||_\nu^2 + \left|\left|\Sigma_{k,\perp} \left(V_{k,\perp}^T Z\right) \left(V_k^T Z\right)^\dagger\right|\right|_\nu^2$$

holds, where P_{AZ} is a projection onto the span of AZ, and where the dagger symbol represents the Moore-Penrose pseudoinverse.



Using this matrix structural condition

Randomly sampling exactly k columns

- Boutsidis, Mahoney, & Drineas 2008

Fast (1+ε)-low-rank projection algorithm

- Sarlos 2006

Project to l=k+p dimensions, for small constant p

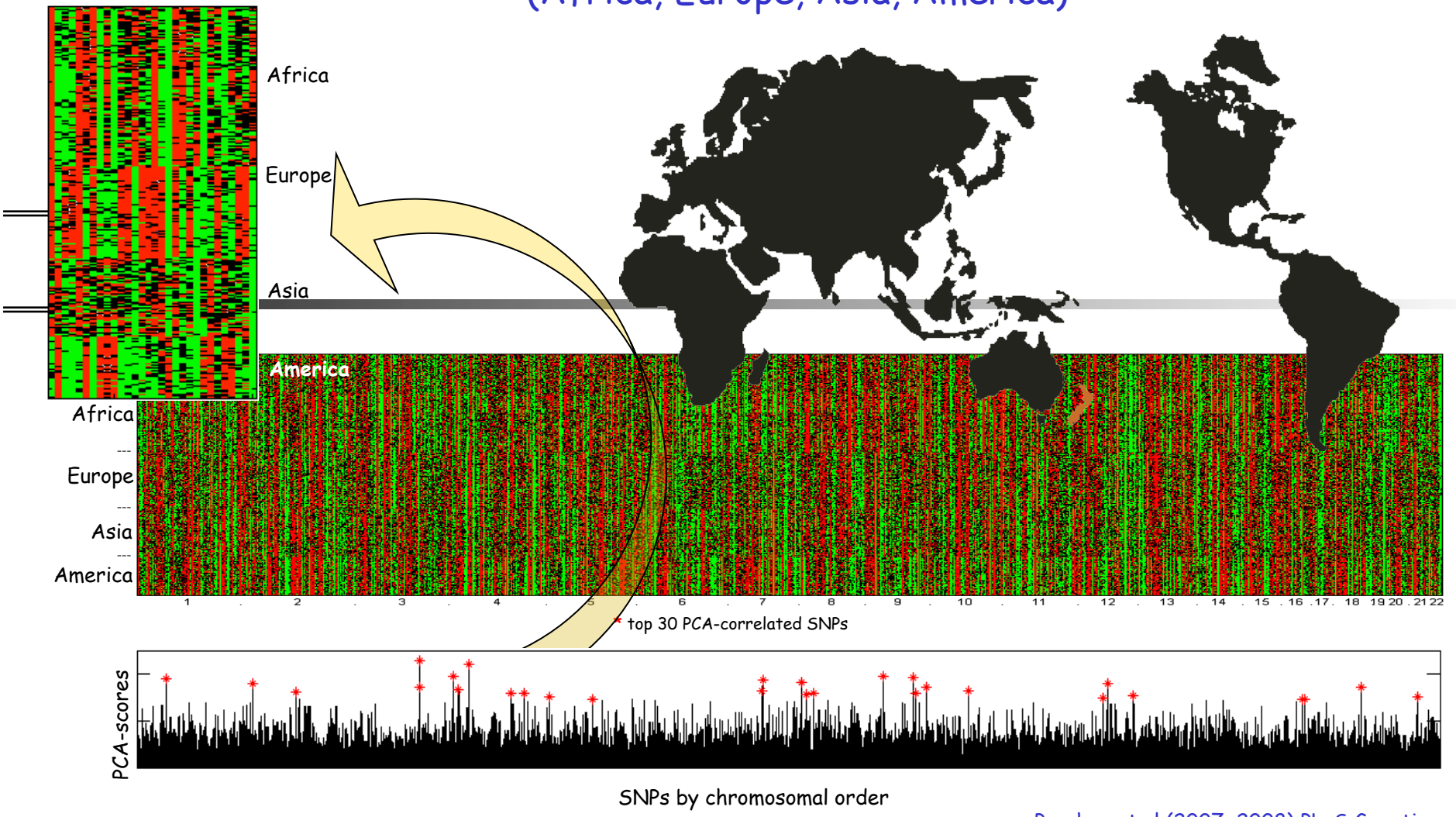
- Wolfe, Liberty, Rokhlin, Tygert 2008

Couple with q steps of power iteration

- Rokhlin,Szlam, Tygert 2009

Decoupling the randomization from the linear algebra---much easier to couple TCS theory with existing NLA & scientific computing methods!

Selecting PCA SNPs for individual assignment to four continents (Africa, Europe, Asia, America)

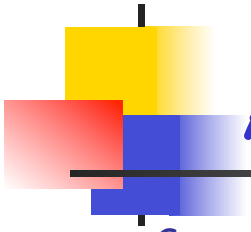


Paschou et al (2007; 2008) PLoS Genetics

Paschou et al (2010) J Med Genet

Drineas et al (2010) PLoS One

Javed et al (2011) Annals Hum Genet



An interesting observation

Sampling w.r.t. to leverage scores results in redundant columns being selected.

(Almost) identical columns have (almost) the same leverage scores and thus might be all selected, even though they do not really add new "information."

First Solution:

Apply a "redundancy removal" step, e.g., a deterministic CSSP algorithm on the sampled columns.

Very good empirically, even with "naïve" CSSP algorithms (such as the pivoted QR factorization).

Conjecture:

The "leverage scores" filter out relevant columns, so deterministic methods do a better job later.

Paschou et al. (2007,2008) for population genetics applications; and Boutsidis et al. (2009, 2010) for theory.

Second Solution:

Apply clustering to the sampled columns and then return a representative column from each cluster.

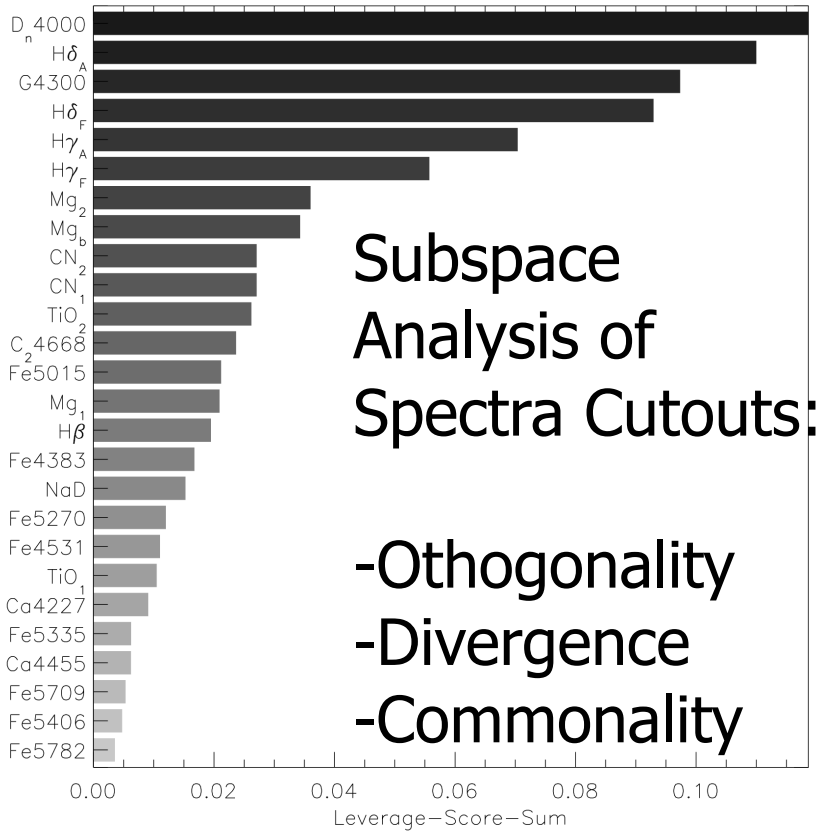
Very good empirically, since it permits clustering of SNPs that have similar functionalities and thus allows better understanding of the proposed ancestry-informative panels.

Ranking Astronomical Line Indices

INDEX DEFINITIONS

	Name	Index Bandpass	Pseudocontinua	Units	Measures	Error[1]	Notes
01	CN_1	4143.375-4178.375	4081.375-4118.875 4245.375-4285.375	mag	CN, Fe I	0.021	
02	CN_2	4143.375-4178.375	4085.125-4097.625 4245.375-4285.375	mag	CN, Fe I	0.023	2
03	Ca4227	4223.500-4236.000	4212.250-4221.000 4242.250-4252.250	Å	Ca I, Fe I, Fe II	0.27	2
04	G4300	4282.625-4317.625	4267.625-4283.875 4320.125-4336.375	Å	CH, Fe I	0.39	
05	Fe4383	4370.375-4421.625	4360.375-4371.625 4444.125-4456.625	Å	Fe I, Ti II	0.53	2
06	Ca4455	4453.375-4475.875	4447.125-4455.875 4478.375-4493.375	Å	Ca I, Fe I, Ni I, Ti II, Mn I, V I	0.25	2
07	Fe4531	4515.500-4560.500	4505.500-4515.500 4561.750-4580.500	Å	Fe I, Ti I, Fe II, Ti II	0.42	2
08	Fe4668	4635.250-4721.500	4612.750-4631.500 4744.000-4757.750	Å	Fe I, Ti I, Cr I, Mg I, Ni I, C_2	0.64	2
09	$H\beta$	4847.875-4876.625	4827.875-4847.875 4876.625-4891.625	Å	$H\beta$, Fe I	0.22	3
10	Fe5015	4977.750-5054.000	4946.500-4977.750 5054.000-5065.250	Å	Fe I, Ni I, Ti I	0.46	2,3
11	Mg_1	5069.125-5134.125	4895.125-4957.625 5301.125-5366.125	mag	MgH, Fe I, Ni I	0.007	3
12	Mg_2	5154.125-5196.625	4895.125-4957.625 5301.125-5366.125	mag	MgH, Mg *b*, Fe I	0.008	3
13	Mg *b*	5160.125-5192.625	5142.625-5161.375 5191.375-5206.375	Å	Mg *b*	0.23	3
14	Fe5270	5245.650-5285.650	5233.150-5248.150 5285.650-5318.150	Å	Fe I, Ca I	0.28	3
15	Fe5335	5312.125-5352.125	5304.625-5315.875 5353.375-5363.375	Å	Fe I	0.26	3
16	Fe5406	5387.500-5415.000	5376.250-5387.500 5415.000-5425.000	Å	Fe I, Cr I	0.20	2,3
17	Fe5709	5698.375-5722.125	5674.625-5698.375 5724.625-5738.375	Å	Fe I, Ni I, Mg I Cr I, V I	0.18	2
18	Fe5782	5778.375-5798.375	5767.125-5777.125 5799.625-5813.375	Å	Fe I, Cr I Cu I, Mg I	0.20	2
19	Na D	5878.625-5911.125	5862.375-5877.375 5923.875-5949.875	Å	Na I	0.24	
20	TiO_1	5938.375-5995.875	5818.375-5850.875 6040.375-6105.375	mag	TiO	0.007	
21	TiO_2	6191.375-6273.875	6068.375-6143.375 6374.375-6416.875	mag	TiO	0.006	

(Worthey et al. 94;
Trager et al. 98)



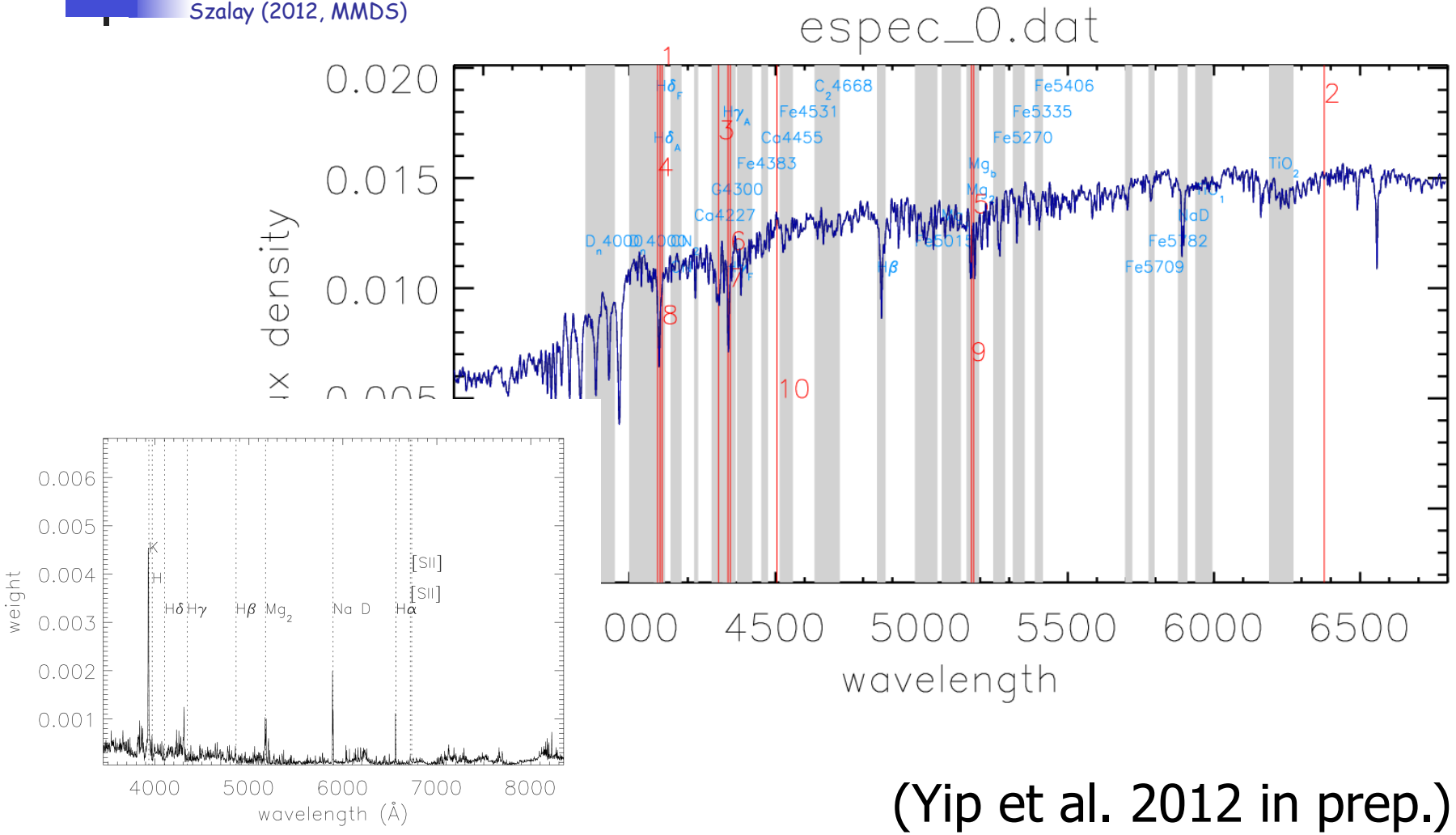
Subspace
Analysis of
Spectra Cutouts:

-Othogonality
-Divergence
-Commonality

(Yip et al. 2012 in prep.)

Identifying new line indices objectively

Szalay (2012, MMDS)



(Yip et al. 2012 in prep.)

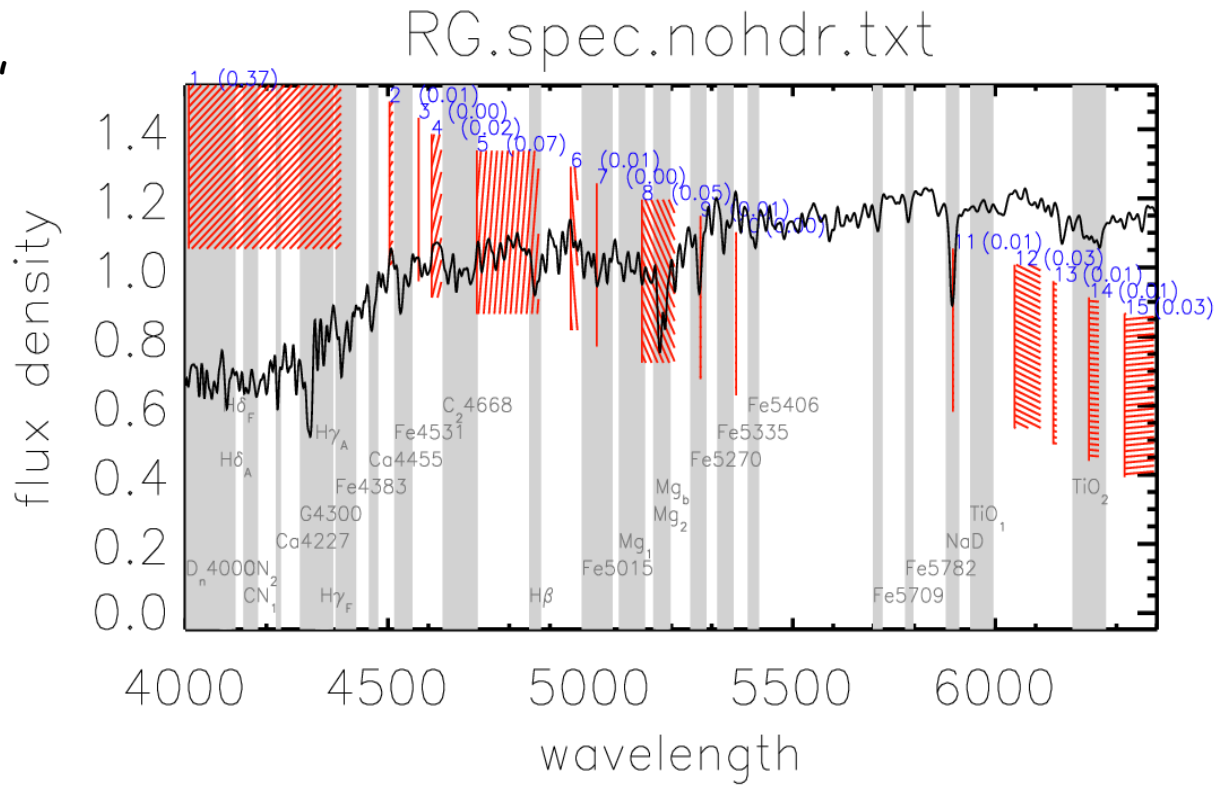
New Spectral Regions (M2;k=5; overselecting 10X; combine if <30A)

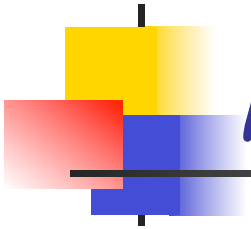
Szalay (2012, MMDS)

Old Lick indices are "ad hoc"

New indices are "objective"

- Recover atomic lines
- Recover molecular bands
- Recover Lick indices
- Informative regions are orthogonal to each other, in contrast to Lick





More recent CSSP-based improvements

Can get (1+ε)-approximation bound with $s=3k/\varepsilon$ columns

- Boutsidis, Drineas, & Magdon-Ismail (FOCS 2011)
- Uses ideas from Batson, Spielman, & Srivastva (STOC 2009)

A (1+ε)-approximation needs at least k/ε columns

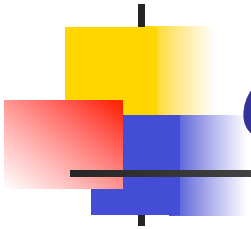
- Deshpande & Vempala (RANDOM 2006)

Almost asymptotically optimal bound

- Guruswami & Sinop (SODA 2012)
- Both deterministic and randomized algorithms

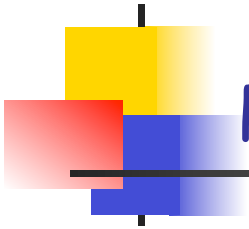
Application to column-based reconstruction in QIP

- Guruswami & Sinop (SODA 2011)



Outline

- Background, motivation, and applications
- Algorithms (in RAM) for least-squares approximation
- Algorithms (in RAM) for low-rank approximation
- Implementation of L1 and L2 regression in parallel and distributed environments



Parallel environments and how they scale

Shared memory

- cores: [10, 10^3]*
- memory: [100GB, 100TB]

Message passing

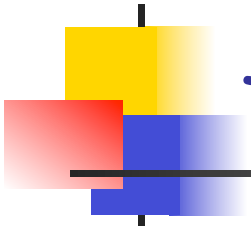
- cores: [200, 10^5]**
- memory: [1TB, 1000TB]
- CUDA cores: [5 x 10^4, 3 x 10^6]***
- GPU memory: [500GB, 20TB]

MapReduce

- cores: [40, 10^5]****
- memory: [240GB, 100TB]
- storage: [100TB, 100PB]*****

Distributed computing

- cores: [-, 3 x 10^5]******



Traditional algorithms

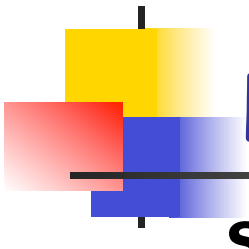
For L2 regression:

- *direct methods*: QR, SVD, and normal equation ($O(mn^2 + n^2)$ time)
  - Pros: high precision & implemented in LAPACK
  - Cons: hard to take advantage of sparsity & hard to implement in parallel environments

- *iterative methods*: CGLS, LSQR, etc.
  - Pros: low cost per iteration, easy to implement in some parallel environments, & capable of computing approximate solutions
  - Cons: hard to predict the number of iterations needed

For L1 regression:

- linear programming
- interior-point methods (or simplex, ellipsoid? methods)
- re-weighted least squares
- first-order methods



Two important notions: leverage and condition

Statistical leverage. (Think: eigenvectors & low-precision solutions.)

- The *statistical leverage scores* of A (assume m>>n) are the diagonal elements of the projection matrix onto the column span of A.
- They equal the L2-norm-squared of any orthogonal basis spanning A.
- They measure:
  - how well-correlated the singular vectors are with the canonical basis
  - which constraints have largest "influence" on the LS fit
  - a notion of "coherence" or "outlierness"
- *Computing them exactly is as hard as solving the LS problem*.

Condition number. (Think: eigenvalues & high-precision solutions.)

- The *L2-norm condition number* of A is $(A) = \sigma_{max}(A)/\sigma_{min}(A)$.
- $\kappa(A)$ bounds the number of iterations
  - for ill-conditioned problems (e.g., $\kappa(A) \cong 10^6 >> 1$), convergence speed is slow.
- *Computing $\kappa(A)$ is generally as hard as solving the LS problem*.

These are for the L2-norm. Generalizations exist for the L1-norm.

Condition number, well-conditioned bases and leverage scores for L1 norm

(Dasgupta, Drineas, Harb, Kumar, Mahoney (2008); Clarkson, Drineas, Magdon-Ismail, Mahoney, Meng, Woodruff (2012))

Convenient to formulate L1 regression in what follows as:

$$\min_{x \in R^n} ||Ax||_1 \text{ s.t. } c^T x = 1$$

• **Def:** A matrix $U \in R^{m \times n}$ is *$(\alpha, \beta, p = 1)$-conditioned* if $||U||_1 \leq \alpha$ and $||x||_\infty \leq \beta ||Ux||_1$, forall x; and *L1-well-conditioned* if α, β = poly(n).

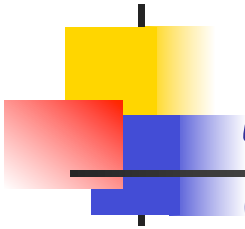
• **Def:** The *L1 leverage scores* of an m x n matrix A, with m > n, are the L1-norms-squared of the rows of any L1-well-conditioned basis of A. (Only well-defined up to poly(n) factors.)

• **Def:** The *L1-norm condition number* of A, denoted by $\kappa_1(A)$, is:

$$\kappa_1(A) = \sigma_{1,max}(A) / \sigma_{1,min}(A)$$
$$= (\text{Max}_{||x||2=1} ||Ax||_1) / (\text{Min}_{||x||2=1} ||Ax||_1)$$

Note that this implies:

$$\sigma_{1,min}(A)||x||_2 \leq ||Ax||_1 \leq \sigma_{1,max}(A)||x||_2 \text{ , forall } x \in R^n.$$



Meta-algorithm for L2 regression

(Drineas, Mahoney, etc., 2006, 2008, etc., starting with SODA 2006; Mahoney FnTML, 2011.)

1: Using the L2 statistical leverage scores of A, construct an importance sampling distribution $\{p_i\}_{i=1,\ldots,m}$

2: Randomly sample a small number of constraints according to $\{p_i\}_{i,\ldots,m}$ to construct a subproblem.

3: Solve the L2-regression problem on the subproblem.

Naïve implementation: $1 + \varepsilon$ approximation in $O(mn^2/\varepsilon)$ time. (Ugh.)

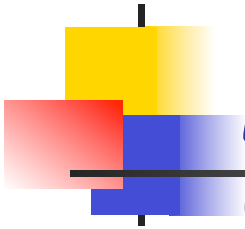
"Fast" $O(mn \log(n)/\varepsilon)$ in RAM if

- Hadamard-based projection and sample uniformly
- Quickly compute approximate leverage scores

"High precision" $O(mn \log(n)\log(1/\varepsilon))$ in RAM if:

- use the random projection/sampling basis to construct a preconditioner

Question: can we extend these ideas to parallel-distributed environments?



Meta-algorithm for L1 (& Lp) regression

(Clakson 2005, DDHKM 2008, Sohler and Woodruff 2011, CDMMMW 2012, Meng and Mahoney 2012.)

1: Using the L1 statistical leverage scores of A, construct an importance sampling distribution $\{p_i\}_{i=1,\ldots,m}$

2: Randomly sample a small number of constraints according to $\{p_i\}_{i,\ldots,m}$ to construct a subproblem.

3: Solve the L1-regression problem on the subproblem.

Naïve implementation: $1 + \varepsilon$ approximation in $O(mn^5/\varepsilon)$ time. (Ugh.)

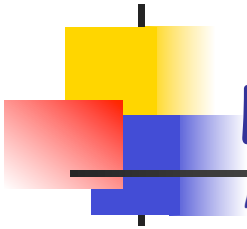
"Fast" in RAM if

- we perform a fast "L1 projection" to uniformize them approximately
- we approximate the L1 leverage scores quickly

"High precision" in RAM if:

- we use the random projection/sampling basis to construct an L1 preconditioner

Question: can we extend these ideas to parallel-distributed environments?

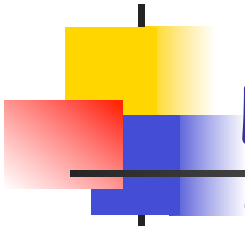


LSRN: a fast parallel implementation

Meng, Saunders, and Mahoney (2011, arXiv)

A parallel iterative solver based on normal random projections

- computes unique min-length solution to $\min_x ||Ax-b||_2$
- very over-constrained or very under-constrained A
- full-rank or rank-deficient A
- A can be dense, sparse, or a linear operator
- easy to implement using threads or with MPI, and scales well in parallel environments



LSRN: a fast parallel implementation

Meng, Saunders, and Mahoney (2011, arXiv)

Algorithm:

- Generate a γn x m matrix with i.i.d. Gaussian entries G
- Let N be R^{-1} or $V \Sigma^{-1}$ from QR or SVD of GA
- Use LSQR or Chebyshev Semi-Iterative (CSI) method to solve the preconditioned problem $\min_y ||ANy-b||_2$

Things to note:

- Normal random projection: embarassingly parallel
- Bound $\kappa(A)$: strong control on number of iterations
- CSI particularly good for parallel environments: doesn't have vector inner products that need synchronization b/w nodes

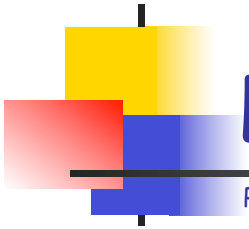
LSRN: Solving real-world problems

Meng, Saunders, and Mahoney (2011, arXiv)

TABLE 6.2

Real-world problems and corresponding running times in seconds. DGELSD doesn't take advantage of sparsity. Though MATLAB's backslash (SuiteSparseQR) may not give the min-length solutions to rank-deficient or under-determined problems, we still report its running times. Blendenpik either doesn't apply to rank-deficient problems or runs out of memory (OOM). LSRN's running time is basically determined by the problem size and the sparsity.

matrix	m	n	nnz	rank	cond	DGELSD	$A\backslash b$	Blendenpik	LSRN
landmark	71952	2704	1.15e6	2671	1.0e8	29.54	0.6498*	-	17.55
rail4284	4284	1.1e6	1.1e7	full	400.0	> 3600	1.203*	OOM	136.0
tnimg_1	951	1e6	2.1e7	925	-	630.6	1067*	-	36.02
tnimg_2	1000	2e6	4.2e7	981	-	1291	> 3600*	-	72.05
tnimg_3	1018	3e6	6.3e7	1016	-	2084	> 3600*	-	111.1
tnimg_4	1019	4e6	8.4e7	1018	-	2945	> 3600*	-	147.1
tnimg_5	1023	5e6	1.05e8	full	-	> 3600	> 3600*	OOM	188.5



LSQR

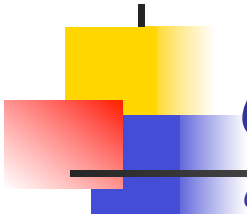
Paige and Saunders (1982)

Code snippet (Python):

```
u     = A.matvec(v) - alpha*u
beta  = sqrt(comm.allreduce(np.dot(u,u)))
...
v     = comm.allreduce(A.rmatvec(u)) - beta*v
```

Cost per iteration:

- two matrix-vector multiplications
- two cluster-wide synchronizations



Chebyshev semi-iterative (CSI)

Golub and Varga (1961)

The strong concentration results on $\sigma^{\max}(AN)$ and $\sigma^{\min}(AN)$ enable use of the CS method, which requires an accurate bound on the extreme singular values to work efficiently.

Code snippet (Python):

```
v  = comm.allreduce(A.rmatvec(r)) - beta*v
x += alpha*v
r -= alpha*A.matvec(v)
```

Cost per iteration:

- two matrix-vector multiplications
- one cluster-wide synchronization

LSRN: on Amazon EC2 cluster

Meng, Saunders, and Mahoney (2011, arXiv)

TABLE 6.3

Test problems on the Amazon EC2 cluster and corresponding running times in seconds. When we enlarge the problem scale by a factor of 10 and increase the number of cores accordingly, the running time only increases by a factor of 50%. It shows LSRN's good scalability. Though the CS method takes more iterations, it is faster than LSQR by saving communication cost.

solver	N_{nodes}	np	matrix	m	n	nnz	N_{iter}	T_{iter}	T_{total}
LSRN w/ CS	2	4	tnimg_4	1024	4e6	8.4e7	106	34.03	170.4
LSRN w/ LSQR							84	41.14	178.6
LSRN w/ CS	5	10	tnimg_10	1024	1e7	2.1e8	106	50.37	193.3
LSRN w/ LSQR							84	68.72	211.6
LSRN w/ CS	10	20	tnimg_20	1024	2e7	4.2e8	106	73.73	220.9
LSRN w/ LSQR							84	102.3	249.0
LSRN w/ CS	20	40	tnimg_40	1024	4e7	8.4e8	106	102.5	255.6
LSRN w/ LSQR							84	137.2	290.2

ℓ_1-norm preconditioning via oblivious projections

Find an oblivious (i.e., independent of A) projection matrix $\Pi \in \mathbb{R}^{\mathcal{O}(n \log n) \times m}$, such that

$$\|Ax\|_1 \leq \|\Pi Ax\|_1 \leq \kappa_\Pi \|Ax\|_1, \quad \forall x.$$

Compute $R = \mathrm{qr}(\Pi A)$.
Then,

$$\frac{1}{\kappa_\Pi}\|y\|_2 \leq \|AR^{-1}y\|_1 \leq \mathcal{O}(n^{1/2}\log^{1/2} n)\|y\|_2, \quad \forall y.$$

Therefore, AR^{-1} is ℓ_1-well-conditioned: $\kappa_1(AR^{-1}) = \mathcal{O}(n^{1/2}\log^{1/2} n \cdot \kappa_\Pi)$.

Constructions for Π	time	κ_Π
Cauchy (Sohler and Woodruff 2011)	$\mathcal{O}(mn^2 \log n)$	$\mathcal{O}(n \log n)$
Fast Cauchy (Clarkson, Drineas, Magdon-Ismail, Mahoney, Meng, and Woodruff 2012)	$\mathcal{O}(mn \log n)$	$\mathcal{O}(n^2 \log^2 n)$

Evaluation on large-scale ℓ_1 regression problem (1 of 2).

	$\|x - x^*\|_1 / \|x^*\|_1$	$\|x - x^*\|_2 / \|x^*\|_2$	$\|x - x^*\|_\infty / \|x^*\|_\infty$
CT (Cauchy)	[0.008, 0.0115]	[0.00895, 0.0146]	[0.0113, 0.0211]
GT (Gaussian)	[0.0126, 0.0168]	[0.0152, 0.0232]	[0.0184, 0.0366]
NOCD	[0.0823, 22.1]	[0.126, 70.8]	[0.193, 134]
UNIF	[0.0572, 0.0951]	[0.089, 0.166]	[0.129, 0.254]

Table: The first and the third quartiles of relative errors in 1-, 2-, and ∞-norms on a data set of size $10^{10} \times 15$. CT clearly performs the best. (FCT performs similarly.) GT follows closely. NOCD generates large errors, while UNIF works but it is about a magnitude worse than CT.

Evaluation on large-scale ℓ_1 regression problem (2 of 2).

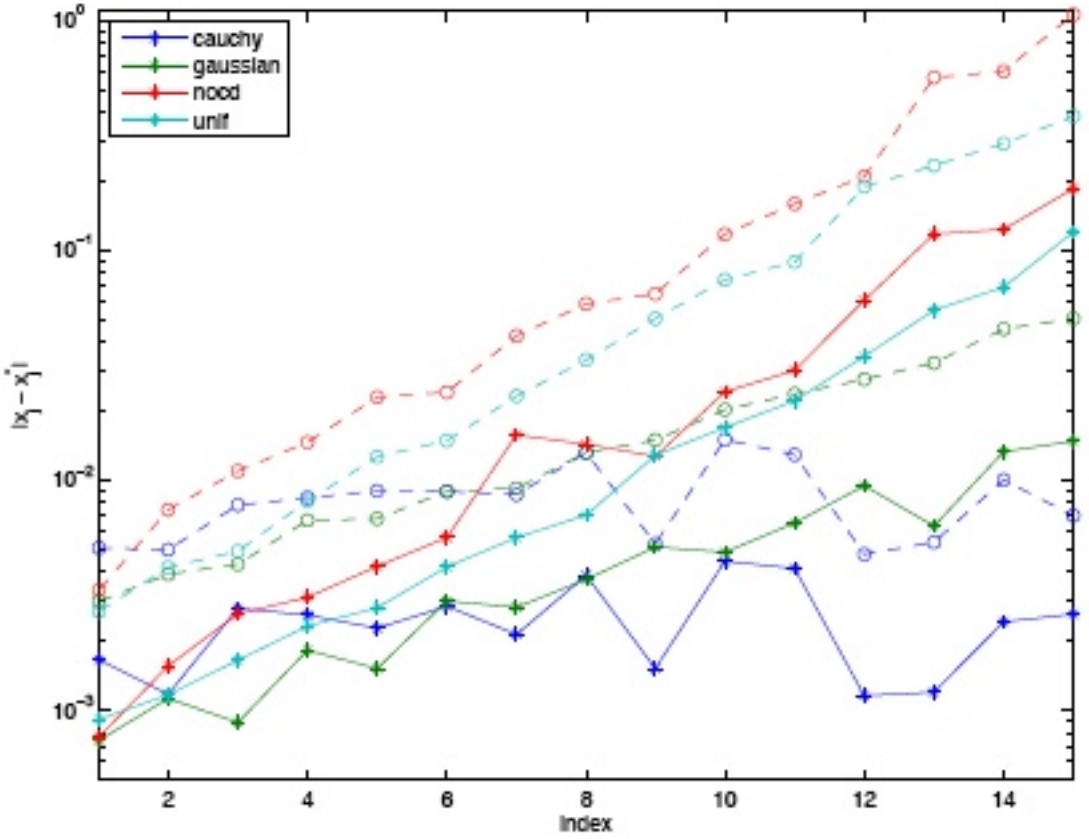


Figure: The first (solid) and the third (dashed) quartiles of entry-wise absolute errors on a data set of size $10^{10} \times 15$. CT clearly performs the best. (FCT performs similarly.) GT follows closely. NOCD and UNIF are much worse.

ℓ_1-norm preconditioning via ellipsoidal rounding

Find an ellipsoid $\mathcal{E} = \{x \,|\, x^T E^{-1} x \leq 1\}$ such that

$$\frac{1}{\kappa_1}\mathcal{E} \subseteq \mathcal{C} = \{x \,|\, \|Ax\|_1 \leq 1\} \subseteq \mathcal{E}.$$

Then we have

$$\|y\|_2 \leq \|AE^{1/2}y\|_1 \leq \kappa_1 \|y\|_2, \quad \forall y.$$

	time	κ_1	passes
Löwner-John ellipsoid	(exists)	$n^{1/2}$	
Clarkson 2005 (Lovász 1986)	$\mathcal{O}(mn^5 \log m)$	n	multiple
Meng and Mahoney 2012	$\mathcal{O}(mn^3 \log m)$	$2n$	multiple
	$\mathcal{O}(mn^2 \log \frac{m}{n})$	$2n^2$	single
	$\mathcal{O}(mn \log \frac{m}{n^2})$	$\mathcal{O}(n^{5/2} \log^{1/2} n)$	single

Fast ellipsoidal rounding

1. Partition A into sub-matrices $A_1, A_2, \ldots, A_M$ of size $\mathcal{O}(n^3 \log n) \times n$.
2. Compute $\tilde{A}_i \in \mathbb{R}^{\mathcal{O}(n \log n) \times n} = \mathsf{FJLT}(A_i)$, for $i = 1, \ldots, M$.
3. Compute an ellipsoid $\mathcal{E}$, which gives a $2n$-rounding of $\tilde{\mathcal{C}} = \{x \mid \sum_{i=1}^{M} \|\tilde{A}_i x\|_2 \leq 1\}$.

$\rightarrow$ By a proper scaling, $\mathcal{E}$ gives an $\mathcal{O}(n^{5/2} \log^{1/2} n)$-rounding of $\mathcal{C}$.

Can use this to get a "one-pass conditioning" algorithm!

A **MapReduce** implementation

- Inputs: $A \in \mathbb{R}^{m \times n}$ and κ_1 such that

$$\|x\|_2 \leq \|Ax\|_1 \leq \kappa_1 \|x\|_2, \quad \forall x,$$

  $c \in \mathbb{R}^n$, sample size s, and number of subsampled solutions n_x.
- **Mapper:**
  1. For each row a_i of A, let $p_i = \min\{s\|a_i\|_1/(\kappa_1 n^{1/2}), 1\}$.
  2. For $k = 1, \ldots, n_x$, emit $(k, a_i/p_i)$ with probability p_i.
- **Reducer:**
  1. Collect row vectors associated with key k and assemble A_k.
  2. Compute $\hat{x}_k = \arg\min_{c^T x = 1} \|A_k x\|_1$ using interior-point methods.
  3. Return $\hat{x}_k$.

Note that multiple subsampled solutions can be computed in a single pass.

Iteratively solving

If we want to have a few more accurate digits from the subsampled solutions, we may consider iterative methods.

	passes	extra work per pass
subgradient (Clarkson 2005)	$\mathcal{O}(n^4/\epsilon^2)$	
gradient (Nesterov 2009)	$\mathcal{O}(m^{1/2}/\epsilon)$	
ellipsoid (Nemirovski and Yudin 1972)	$\mathcal{O}(n^2 \log(\kappa_1/\epsilon))$	
inscribed ellipsoids (Tarasov, Khachiyan, and Erlikh 1988)	$\mathcal{O}(n \log(\kappa_1/\epsilon))$	$\mathcal{O}(n^{7/2} \log n)$

The Method of Inscribed Ellipsoids (MIE)

MIE works similarly to the bisection method, but in a higher dimension.

It starts with a search region $\mathcal{S}_0 = \{x \mid Sx \leq t\}$ which contains a ball of desired solutions described by a separation oracle. At step k, we first compute the maximum-volume ellipsoid $\mathcal{E}_k$ inscribing $\mathcal{S}_k$. Let y_k be the center of $\mathcal{E}_k$. Send y_k to the oracle, if y_k is not a desired solution, the oracle returns a linear cut that refines the search region $\mathcal{S}_k \to \mathcal{S}_{k+1}$.

Why do we choose MIE?

- Least number of iterations
- Initialization using all the subsampled solutions
- Multiple queries per iteration

Constructing the initial search region

Given any feasible $\hat{x}$, let $\hat{f} = \|A\hat{x}\|_1$ and $\hat{g} = A^T \text{sign}(A\hat{x})$. we have

$$\|x^* - \hat{x}\|_2 \leq \|A(x^* - \hat{x})\|_1 \leq \|Ax^*\|_1 + \|A\hat{x}\|_1 \leq 2\hat{f},$$

and, by convexity,

$$\|Ax^*\|_1 \geq \|A\hat{x}\|_1 + \hat{g}^T(x^* - \hat{x}),$$

which implies $\hat{g}^T x^* \leq \hat{g}^T \hat{x}$.

Hence, for each subsampled solution, we have a hemisphere that contains the optimal solution.

We use all these hemispheres to construct the initial search region $\mathcal{S}_0$.

Computing multiple f and g in a single pass

On MapReduce, the cost of input/output may dominate the cost of the actual computation, which requires us to design algorithms that could do more computations in a single pass.

- A single query:

$$f(x) = \|Ax\|_1, \quad g(x) = A^T \text{sign}(Ax).$$

- Multiple queries:

$$F(X) = \text{sum}(|AX|, 0), \quad G(X) = A^T \text{sign}(AX).$$

An example on a 10-node Hadoop cluster:

- $A : 10^8 \times 50$, 118.7GB.
- A single query: 282 seconds.
- 100 queries in a single pass: 328 seconds.

MIE with sampling initialization and multiple queries

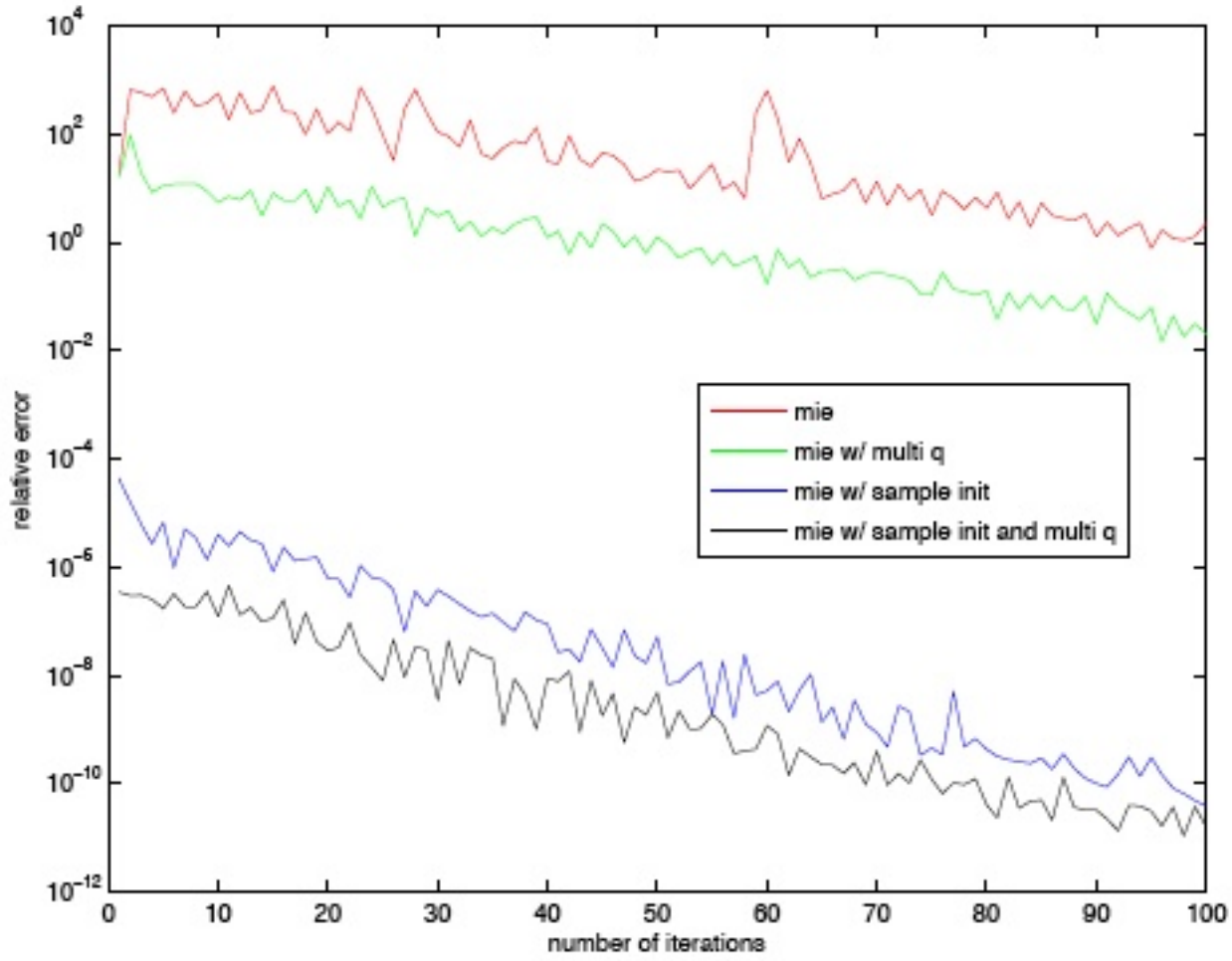


Figure: Comparing different MIEs on an ℓ_1 regression problem of size $10^6 \times 20$.

MIE with sampling initialization and multiple queries

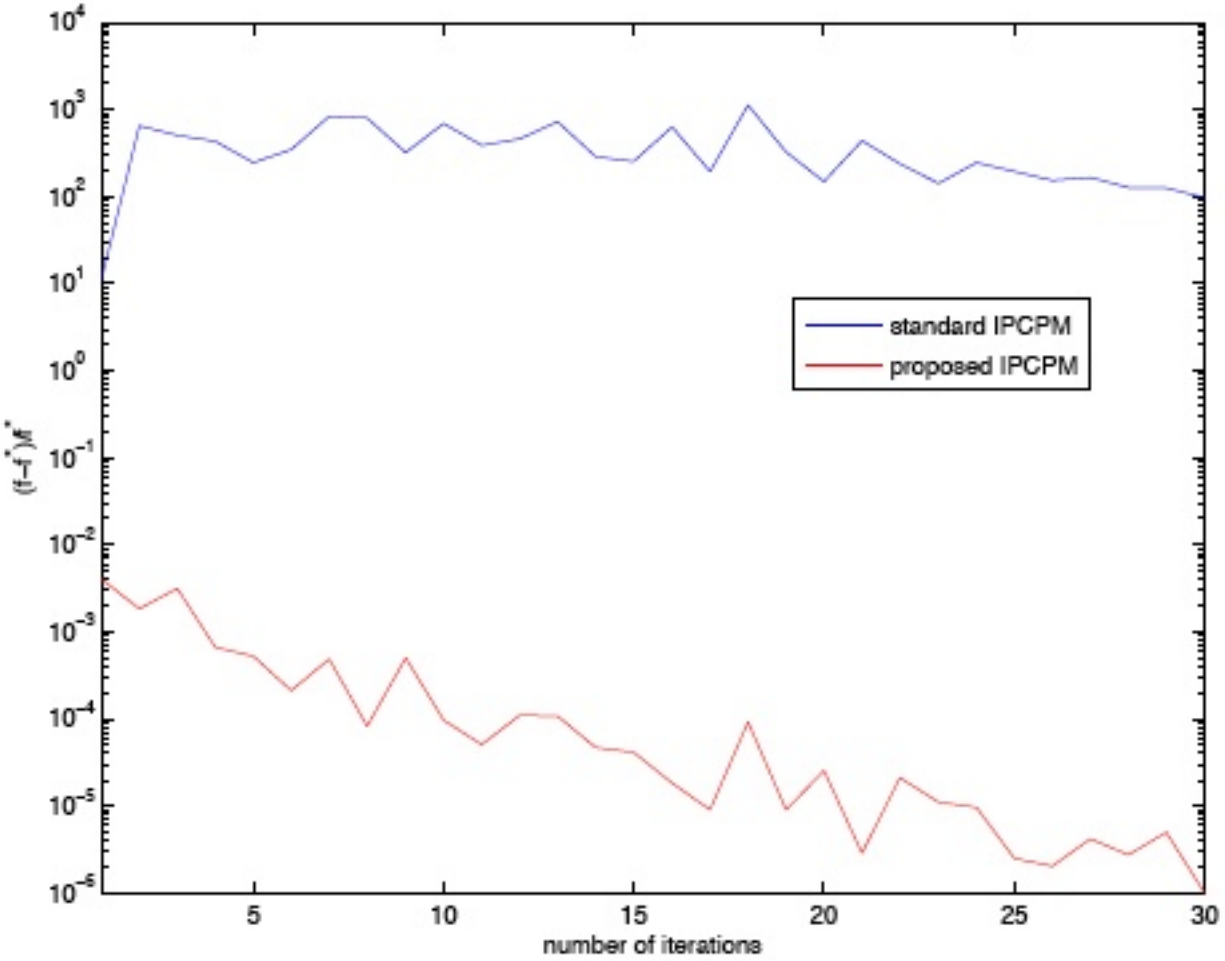
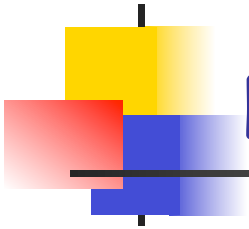


Figure: Comparing different MIEs on an ℓ_1 regression problem of size $5.24e9 \times 15$.



Dealing with sparse matrices

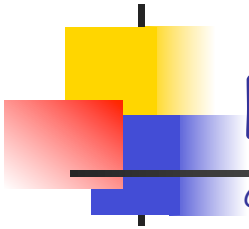
Many matrices are "sparse," i.e., have very few nonzeros:

- Scientific computing – nonzeros often structured, e.g., can apply fast to arbitrary vector
- Informatics graphs – nonzeros very unstructured, e.g., can have no good large partitions

Problem: Projections, etc. typically densify matrices, so can perform very poorly on sparse matrices.

Solution 1: If the sparse matrix is "structured," can often quickly apply a dense Gaussian projection.

Solution 2: Try to get embeddings and/or solution to regression, etc. problems in $O(nnz(A)) + poly(n/\varepsilon)$ time, if $m \gg n$, i.e., "input sparsity" time.



Dealing with sparse matrices, cont

Clarkson and Woodruff (arXiv 2012)

Thm: (Clarkson&Woodruff-12) Can find L2 embedding matrix and compute solution to L2 regression, low-rank approximation, leverage score computation, etc., in input sparsity time.

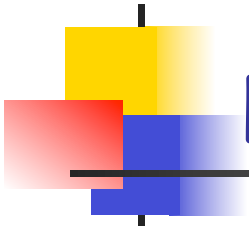
Pf: Decompose space into high-leverage and low-leverage parts. High-leverage rows are heavy hitters.

Extension 1: (Meng-Mahoney12) Don't need leverage decomposition for L2; improved direct proof that uses Gershgorin discs.

Extension 2a: (MM12) Use CW12 decomposition ideas extend to Lp input-sparsity time embeddings and apply to Lp regression.

Extension 2b: (CW12) Can get Lp regression with direct proof with L2 ideas and L1-FCT.

Fruitful interplay between TCS data streaming and NLA structural ideas; see David's talk for more details!

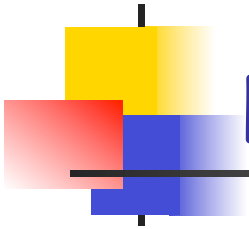


Future directions?

Lots of them:

- Other traditional NLA and large-scale optimization problems
- Parallel and distributed computational environments
- Sparse graphs, sparse matrices, and sparse projections
- Laplacian matrices and large informatics graphs
- Randomized algorithms and implicit regularization
- ...

"New data and new problems are forcing us to reconsider the algorithmic and statistical basis of large-scale data analysis."



For more info ...

Two very good recent reviews:

- "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," by N. Halko, P. G. Martinsson, J. Tropp, SIAM Review, 53(2), 2011. (Also available at arXiv:0909.4061).
- "Randomized Algorithms for Matrices and Data," M. W. Mahoney, NOW Publishers' Foundations and Trends in Machine Learning series, 2011. (Also available at arXiv: 1104.5557).

And no doubt more to come ...