Fast Monte Carlo Algorithms for Matrix Operations and Large Data Set Analysis

Michael W. Mahoney

Department of Mathematics
Yale University
michael.mahoney@yale.edu
http://www.cs.yale.edu/homes/mmahoney

Joint work with:
Petros Drineas and Ravi Kannan
and with:
S. Lafon, A. Lee, M. Maggioni, and R. Coifman

Overview and Summary

- Pass-Efficient Model and Random Sampling
- Matrix Multiplication
- Singular Value Decomposition
- CUR Decomposition
- Lower Bounds
- Kernel-based data sets and KernelCUR
- Tensor-based data sets and TensorCUR
- Large scientific (e.g., chemical and biological) data

Goal: *To develop and analyze fast Monte Carlo algorithms for performing useful computations on large matrices.*

- Matrix Multiplication
- Computation of the Singular Value Decomposition
- Computation of the CUR Decomposition
- Testing Feasibility of Linear Programs

Such matrix computations generally require time which is *superlinear* in the number of nonzero elements of the matrix, e.g., n^3 in practice.

These and related algorithms useful in applications where data sets are modeled by matrices and are extremely large.

Applications of these Algorithms

Matrices arise, e.g., since n *objects* (documents, genomes, images, web pages), each with m *features*, may be represented by a matrix $A \in \mathbb{R}^{m \times n}$.

- Covariance Matrices
- Latent Semantic Indexing
- DNA Microarray Data
- Eigenfaces and Image Recognition
- Similarity Query
- Matrix Reconstruction
- Numerous Linear Programming Applications
- Design of Approximation Algorithms for Classical CS NP-hard Optimization Problems

Linear Algebra Review

For $A \in \mathbb{R}^{m \times n}$ let $A^{(j)}$, $j = 1, \ldots, n$, denote the j-th column of A and $A_{(i)}$, $i = 1, \ldots, m$, denote the i-th row of A.

$\|A\|_2 = \sup_{x \in \mathbb{R}^n,\ x \neq 0} \frac{|Ax|}{|x|}$
$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2 \right)^{1/2} = \left(\mathbf{Tr}\left(A^T A \right) \right)^{1/2}$
$\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\, \|A\|_2$

Theorem. [SVD] *If $A \in \mathbb{R}^{m \times n}$, then there exist orthogonal matrices U and V and a matrix $\Sigma =$* **diag**$(\sigma_1, \ldots, \sigma_\rho)$, *$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_\rho \geq 0$, such that*

$$A = U \Sigma V^T = U_r \Sigma_r V_r^T = \sum_{t=1}^{r} \sigma_t u^t v^{tT}.$$

$U = [u^1 u^2 \ldots u^m]$, $V = [v^1 v^2 \ldots v^n]$, and Σ constitute the Singular Value Decomposition (SVD) of A.

- σ_i are the singular values of A
- u^i, v^i are the i-th left and the i-th right singular vectors

Linear Algebra Review, Cont.

Recall that:

- $\begin{cases} Av^i = \sigma_i u^i \\ A^T u^i = \sigma_i v^i \end{cases}$

- $\begin{cases} \|A\|_2 = \sigma_1 \\ \|A\|_F^2 = \sum_{i=1}^r \sigma_i^2 \end{cases}$

Theorem. *Let* $A_k = U_k \Sigma_k V_k^T = \sum_{t=1}^k \sigma_t u^t v^{tT}$*:*

- $A_k = U_k U_k^T A = \left(\sum_{t=1}^k u^t u^{tT} \right) A$

- $A_k = A V_k V_k^T = A \left(\sum_{t=1}^k v^t v^{tT} \right)$

- $\|A - A_k\|_2 = \min_{D \in \mathbb{R}^{m \times n} : \mathit{rank}(D) \leq k} \|A - D\|_2$

- $\|A - A_k\|_F^2 = \min_{D \in \mathbb{R}^{m \times n} : \mathit{rank}(D) \leq k} \|A - D\|_F^2$

- $\max_{t:1 \leq t \leq n} |\sigma_t(A+E) - \sigma_t(A)| \leq \|E\|_2$

- $\sum_{k=1}^n (\sigma_k(A+E) - \sigma_k(A))^2 \leq \|E\|_F^2$

The Pass-Efficient Model

Amount of *disk space* has increased enormously; *RAM* and *computing speeds* have increased less rapidly.

We can *store* large amounts of data but we **cannot** *process* these data with traditional algorithms.

In the *Pass-Efficient Model*:

- Data are assumed to be *stored on disk*.
- The only access the algorithm has to the data is with a pass, where a *pass* is a sequential read of the entire input from disk where only a constant amount of processing time is permitted per bit read.
- An algorithm is allowed *additional RAM space* and *additional computation time*.

An algorithm is *pass-efficient* if it requires a small constant number of passes and sublinear additional time and space to compute a *description* of the solution.

If data are $A \in \mathbb{R}^{m \times n}$, then algorithms which require additional time and space that is $O(m+n)$ or $O(1)$ are pass-efficient.

Approximating Matrix Multiplication

For $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, AB may be written as the sum of n rank-one matrices:

$$AB = \sum_{t=1}^{n} A^{(t)} B_{(t)}.$$

$$\left(\quad A \quad \right) \left(\quad B \quad \right) = \sum_{t=1}^{n} \left(A^{(t)} \right) \left(\quad B_{(t)} \quad \right).$$

$\textsc{BasicMatrixMultiplication}$ (BMMA) Algorithm Summary.

- Given $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, $c \in \mathbb{Z}^+$, and $\{p_i\}_{i=1}^n$.
- Randomly sample c columns of A according to $\{p_i\}_{i=1}^n$ and rescale each column by $1/\sqrt{cp_{i_t}}$ to form $C \in \mathbb{R}^{m \times c}$.
- Sample the corresponding c rows of B and rescale each row by $1/\sqrt{cp_{i_t}}$ to form $R \in \mathbb{R}^{c \times p}$.
- Return $P = CR$.

$$P = CR = \sum_{t=1}^{c} C^{(t)} R_{(t)} = \sum_{t=1}^{c} \frac{1}{cp_{i_t}} A^{(i_t)} B_{(i_t)}$$

Implementation of the BMMA

- Recall, $A : \mathbb{R}^n \to \mathbb{R}^m$ and $B : \mathbb{R}^p \to \mathbb{R}^n$.
- Uniform sampling: $O(1)$ space and time to sample and $O(m+p)$ space and time to construct C and R
- Nonuniform sampling: for nice probabilities one pass and $O(n)$ (or $O(1)$ if $B = A^T$) space and time to construct probabilities and a second pass and $O(m+p)$ space and time to construct C and R.

Def: A set of sampling probabilities $\{p_i\}_{i=1}^n$ are *nearly optimal probabilities* if $\exists$ a positive constant $\beta \leq 1$:

$$p_k \geq \frac{\beta \left|A^{(k)}\right| \; \left|B_{(k)}\right|}{\sum_{k'=1}^n \left|A^{(k')}\right| \; \left|B_{(k')}\right|}$$

Note: If $\beta = 1$ then $\mathbf{E}\left[\|AB - CR\|_F^2 \right]$ is minimized.

Lemma. [DKM]

$$\begin{aligned}
\mathbf{E}\left[(CR)_{ij}\right] &= (AB)_{ij} \\
\mathbf{Var}\left[(CR)_{ij}\right] &= \frac{1}{c}\sum_{k=1}^{n}\frac{A_{ik}^2 B_{kj}^2}{p_k} - \frac{1}{c}(AB)_{ij}^2 \\
\mathbf{E}\left[\,\|AB - CR\|_F^2\right] &= \sum_{i=1}^{m}\sum_{j=1}^{p}\mathbf{Var}\left[(CR)_{ij}\right].
\end{aligned}$$

Theorem. [DKM] *If* $\{p_i\}_{i=1}^n$ *are* nearly optimal probabilities *then*

$$\mathbf{E}\left[\,\|AB - CR\|_F\right] \leq \frac{1}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F.$$

Let $\delta \in (0,1)$ *and* $\eta = 1 + \sqrt{(8/\beta)\log(1/\delta)}$*; then with probability at least* $1-\delta$*:*

$$\|AB - CR\|_F \leq \frac{\eta}{\sqrt{\beta c}}\|A\|_F\,\|B\|_F.$$

Proof. Expectation straightforward; whp uses Doob martingales and Hoeffding-Azuma inequality. $\Box$

Corollary. [DKM] *If* $B = A^T$ *and* $\{p_i\}_{i=1}^n$ *are* nearly optimal probabilities, *i.e.*, $p_k \geq \frac{\beta \left| A^{(k)} \right|^2}{\|A\|_F^2}$, *then*

$$\mathbf{E}\left[\left\| AA^T - CC^T \right\|_F \right] \leq \frac{1}{\sqrt{\beta c}} \|A\|_F^2$$

and with probability at least $1 - \delta$*:*

$$\left\| AA^T - CC^T \right\|_F \leq \frac{\eta}{\sqrt{\beta c}} \|A\|_F^2 .$$

$$\left(\quad A \quad \right) \left(\quad A^T \quad \right) = \left(\quad C \quad \right) \left(\quad C^T \quad \right)$$

Approximating the SVD of a Matrix

Goal: Given a matrix $A \in \mathbb{R}^{m \times n}$ we wish to approximate its top k singular values and the corresponding singular vectors in a constant number of passes through the data and additional space and time that is either $O(m+n)$ or $O(1)$, independent of m and n.

$\textsc{LinearTimeSVD}$ Algorithm Summary. (DFKVV99)

- Given $A \in \mathbb{R}^{m \times n}$, $c, k \in \mathbb{Z}^+$, and $\{p_i\}_{i=1}^n$.
- Randomly sample c columns of A according to $\{p_i\}_{i=1}^n$ and rescale each column by $1/\sqrt{cp_{i_t}}$ to form $C \in \mathbb{R}^{m \times c}$.
- Compute $C^TC \in \mathbb{R}^{c \times c}$ (recall $CC^T \approx AA^T$) and its SVD; the singular vectors of C^TC are right singular vectors of C.
- Compute $H_k (= U_C)$, the top k left singular vectors of C and approximations to the left singular vectors of A.

Note: Sampling probabilities p_k must be chosen carefully; assume they are nearly optimal.

The LinearTimeSVD Algorithm, Cont.

Theorem. [DFKVV99,DKM] *Construct* H_k *with the* $\textsc{LinearTimeSVD}$ *algorithm by sampling* c *columns of* A *with nearly optimal probabilities and let* $\eta = 1+\sqrt{(8/\beta)\log(1/\delta)}$. *Let* $\epsilon > 0$. *If* $c = \Omega(k\eta^2/\epsilon^4)$, *then*

$$\left\|A - H_k H_k^T A\right\|_F \leq \left\|A - A_k\right\|_F + \epsilon \left\|A\right\|_F$$

in expectation and with high probability. In addition, if $c = \Omega(\eta^2/\epsilon^4)$, *then*

$$\left\|A - H_k H_k^T A\right\|_2 \leq \left\|A - A_k\right\|_2 + \epsilon \left\|A\right\|_F$$

in expectation and with high probability.

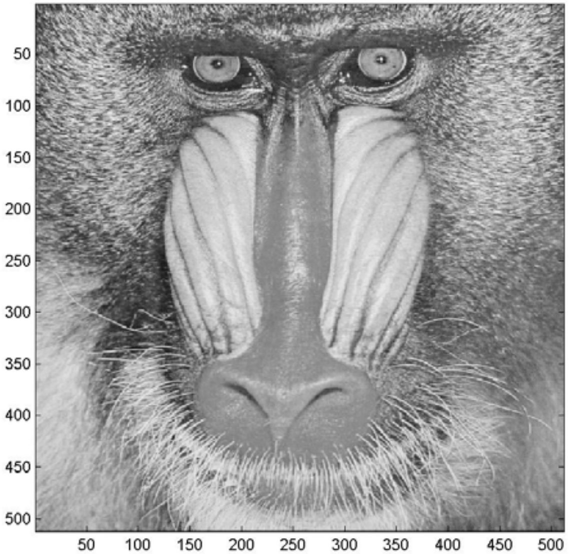
Proof. Combine $\|\cdot\|_F^2$ and $\|\cdot\|_2^2$ results with bound on $\left\|AA^T - CC^T\right\|_F$ from approximate matrix multiplication algorithm. $\Box$

Lower Bounds

- How many queries does a sampling algorithm need to approximate a given function accurately with high probability?
- ZBY03 proves lower bounds for the low rank matrix approximation problem and the matrix reconstruction problem.
  - Any sampling algorithm that with high probability finds a good low rank approximation requires $\Omega(m+n)$ queries.
  - Even if the algorithm is given the exact weight distribution over the columns of a matrix it will still require $\Omega(k/\epsilon^4)$ queries.
  - Finding a matrix D such that $\|A-D\|_F \leq \epsilon \|A\|_F$ requires $\Omega(mn)$ queries and that finding a D such that $\|A-D\|_2 \leq \epsilon \|A\|_F$ requires $\Omega(m+n)$ queries.
- Applied to our results:
  - The LinearTimeSVD algorithm is optimal with respect to $\|\cdot\|_F$ bounds; see also DFKVV99.
  - The ConstantTimeSVD algorithm is optimal with respect to $\|\cdot\|_F$ bounds up to polynomial factors; see also FKV98.
  - The CUR algorithm is optimal for constant ϵ.

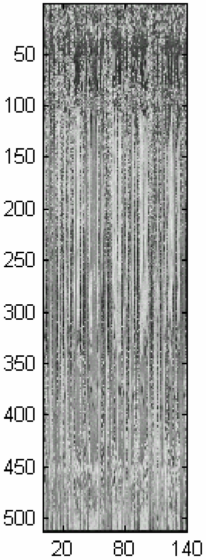
Example of randomized SVD

A



Original matrix

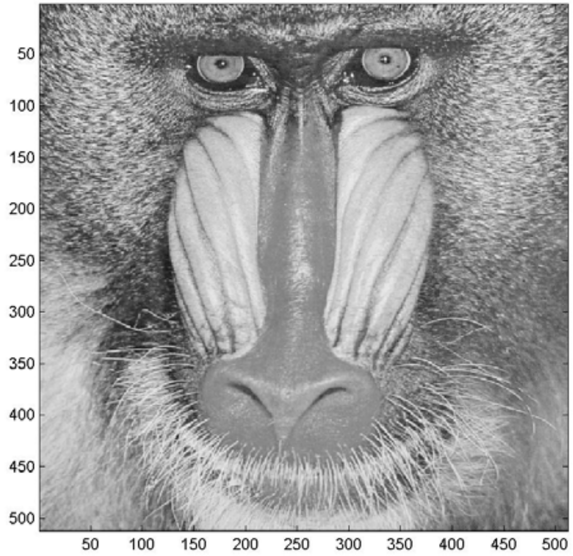
C



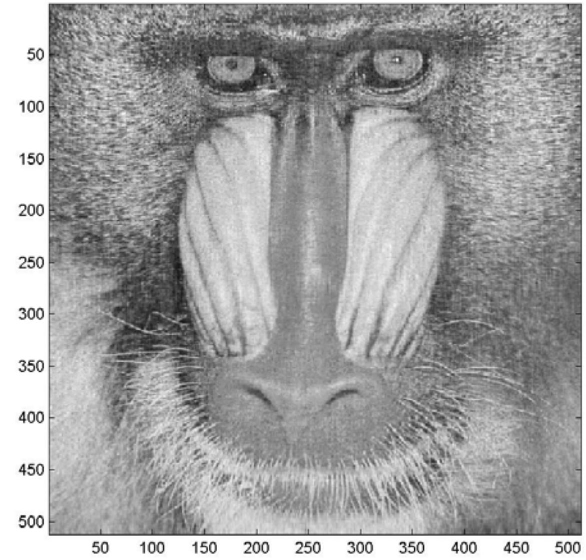
After sampling columns

Compute the top k left singular vectors of the matrix C and store them in the 512-by-k matrix H_k.

Example of randomized SVD (cont'd)



A



$H_kH_k^TA$

A and $H_kH_k^TA$ are close.

A novel CUR matrix decomposition

1. A "sketch" consisting of a few rows/columns of the matrix is adequate for efficient approximations.
2. Create an approximation to the original matrix of the following form:

Carefully chosen U

$$\begin{pmatrix} \\ & A & \\ \\ \end{pmatrix} \approx \begin{pmatrix} \\ C \\ \\ \end{pmatrix} \cdot \begin{pmatrix} U \end{pmatrix} \cdot \begin{pmatrix} & R & \end{pmatrix}$$

O(1) columns

O(1) rows

3. Given a query vector x, instead of computing A · x, compute CUR · x to identify its nearest neighbors.

$$\max_{x:|x|=1} \|Ax - CURx\| = \|A - CUR\|_2 \leq \epsilon \|A\|_F$$

The CUR decomposition

Given a **large** m-by-n matrix A (stored on disk), compute a decomposition CUR of A such that:

$$\underbrace{\begin{pmatrix} \\ \\ & A & \\ \\ \\ \end{pmatrix}}_{m \times n} \approx \underbrace{\begin{pmatrix} \\ \\ C \\ \\ \\ \end{pmatrix}}_{m \times c} \cdot \underbrace{\begin{pmatrix} U \end{pmatrix}}_{c \times r} \cdot \underbrace{\begin{pmatrix} & R & \end{pmatrix}}_{r \times n}$$

1. C consists of c = $O(k/\varepsilon^2)$ columns of A.
2. R consists of r = $O(k/\varepsilon^2)$ rows of A.
3. C (R) is created using importance sampling, e.g. columns (rows) are picked in i.i.d. trials with respect to probabilities

$$p_i = |A^{(i)}|^2 / \sum_i |A^{(i)}|^2$$

The CUR decomposition (cont'd)

Given a **large** m-by-n matrix A (stored on disk), compute a decomposition CUR of A such that:

4. C, U, R can be stored in O(m+n) space, after making **two** passes through the entire matrix A, using O(m+n) additional space and time.

5. The product CUR satisfies (with high probability)

$$\|A - CUR\|_F \leq \|A - A_k\|_F + \epsilon \|A\|_F$$

$$\|A - CUR\|_2 \leq \epsilon \|A\|_F$$

Computing U

Intuition:

The CUR algorithm essentially expresses every row of the matrix A as a linear combination of a small subset of the rows of A.

- This small subset consists of the rows in R.
- Given a row of A – say $A_{(i)}$ – the algorithm computes a good fit for the row $A_{(i)}$ using the rows in R as the basis, by approximately solving

$$\min_u \left\| \underset{1 \times n}{\left(\quad A_{(i)} \quad \right)} - \underset{1 \times r}{\left(\quad u \quad \right)} \cdot \underset{r \times n}{\left(\quad R \quad \right)} \right\|_2$$

Notice that only c = O(1) element of the i-th row are given as input.

However, a vector of coefficients *u* can still be computed.

Computing U (cont'd)

Given *c* elements of $A_{(i)}$ the algorithm computes a good fit for the row $A_{(i)}$ using the rows in R as the basis, by approximately solving:

$$\min_u \left\| \underbrace{\left(\quad \tilde{A}_{(i)} \quad \right)}_{1 \times c} - \underbrace{\left(\quad u \quad \right)}_{1 \times r} \cdot \underbrace{\left(\quad \tilde{R} \quad \right)}_{r \times c} \right\|_2$$

However, our CUR decomposition *approximates* the vectors *u* instead of exactly computing them.

Open problem: Is it possible to improve our error bounds using the optimal coefficients?

Error bounds for CUR

Assume A_k is the "best" rank k approximation to A (through SVD). Then, if we pick $O(k/\varepsilon^2)$ rows and $O(k/\varepsilon^2)$ columns,

$$\|A - CUR\|_F^2 \leq \|A - A_k\|_F^2 + \varepsilon \|A\|_F^2$$

If we pick $O(1/\varepsilon^2)$ rows and $O(1/\varepsilon^2)$ columns,

$$\begin{aligned} \|A - CUR\|_2^2 &\leq \|A - A_k\|_2^2 + \varepsilon \|A\|_F^2 \\ &\leq \left(\frac{1}{k+1} + \varepsilon\right) \|A\|_F^2 \\ &\leq 2\varepsilon \|A\|_F^2 \end{aligned}$$

Other CUR decompositions (1)

Computing U in constant time (instead of O(m+n))

Our CUR decomposition computes a provably efficient U in linear time.

In recent work (DM '04), we demonstrate how to compute a provably efficient U in constant time – the ConstantTimeCUR decomposition.

Our ConstantTimeCUR decomposition:

- samples $O(\text{poly}(k,\varepsilon))$ rows and columns of A,
- needs an extra pass through the matrix A,
- significantly improves the error bounds of Frieze, Kannan, and Vempala, FOCS '98, JACM '04,
- is useful for designing approximation algorithms,
- but has a more complicated analysis.

Other CUR decompositions (2)

Solving for the optimal U

Given c elements of the i-th row of A, the algorithm computes the "best fit" for the i-th row using the rows in R as the basis, by solving:

$$\min_u \left\| \underset{1 \times c}{\left(\tilde{A}_{(i)} \right)} - \underset{1 \times r}{\left(u \right)} \cdot \underset{r \times c}{\left(\tilde{R} \right)} \right\|_2 \longrightarrow u = \tilde{R}^+ \tilde{A}_{(i)}$$

Using the above strategy, we can also compute a CUR decomposition, with a different U in the middle.

(This decomposition has been experimentally proposed in the context of fast kernel computation.)

Open problem: What is the improvement?

Other CUR decompositions (3)

<u>An alternative perspective:</u>

$$\begin{pmatrix} & & \\ & A & \\ & & \end{pmatrix} \approx \begin{pmatrix} \\ C \\ \\ \end{pmatrix} \cdot \begin{pmatrix} U \end{pmatrix} \cdot \begin{pmatrix} & R & \end{pmatrix}$$

Optimal U

Q. Can we find the "best" set of columns and rows to include in C and R?

Randomized or quasi-randomized or deterministic strategies are acceptable.

Results by S. A. Goreinov, E. E. Tyrtyrshnikov, and N.L. Zamarashkin imply (rather weak) error bounds if we choose the columns and rows of A that define a parallelpiped of maximal volume.

Fast Computation of Kernels

Q. SVD has been used to identify/extract linear structure from data. What about non-linear structures, like multi-linear structures or non-linear manifold structure?

A. Kernel-based learning algorithms.

Data $\Psi = \{\Psi_{(1)}, \ldots, \Psi_{(m)}\} \in R^{m \times n}$

Mapping $\phi : \Psi \to \Phi$ (feature space)

Gram Matrix $G_{ij} = \underbrace{G\left(\Psi_{(i)}, \Psi_{(j)}\right)}_{\text{PSD matrix}} = \underbrace{\left\langle \phi\left(\Psi_{(i)}\right), \phi\left(\Psi_{(j)}\right)\right\rangle}_{\text{inner product}}$

Algorithms extracting linear structure can be applied to G without knowing ϕ !

Isomap, LLE, Laplacian Eigenmaps, SDE, are all Kernel PCA for special Gram matrices.

However, running, e.g., SVD to extract linear structure from the Gram matrix still requires $O(m^3)$ time.

We can apply CUR-type decompositions to speed up such calculations.

Fast Computation of Kernels (cont'd)

A potential issue is that the CUR decomposition of the Gram matrix is not a positive semidefinite matrix.

However, if we compute the "optimal" U matrix, then the CUR approximation to the optimal matrix is PSD.

For the special case of PSD matrix $G = XX^T$ for some matrix X, we can prove that using the "optimal" U guarantees:

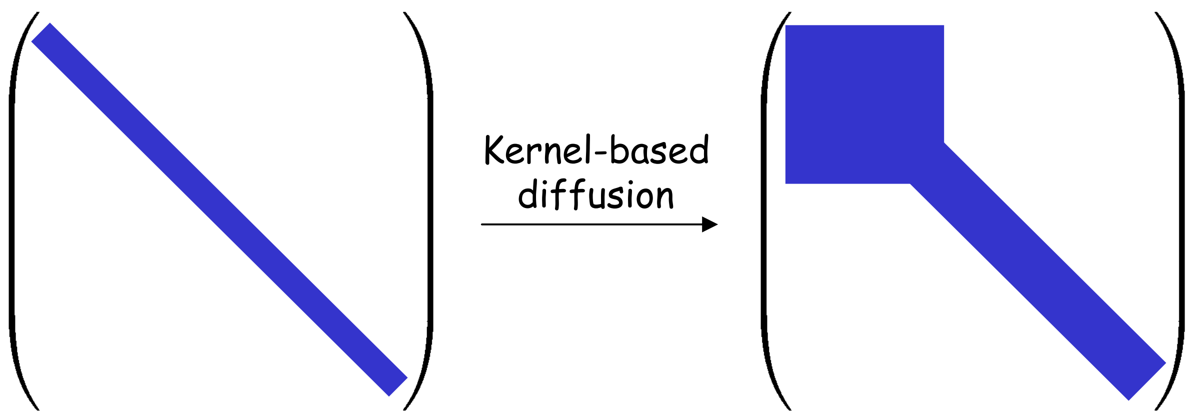
$$\|G - CUC^T\|_F^2 \leq \|G - G_k\|_F^2 + \epsilon \|X\|_F^4$$

$\|X\|_F^4$ vs. $\|G\|_F^2 = \|XX^T\|_F^2$

What we can (almost) do with kernels

Adjacency matrix, t=0

Adjacency matrix, t=t*



To construct a coarse-grained version of the data graph:

- Construct landmarks,
- Partition/Quantization,
- Diffusion wavelets.

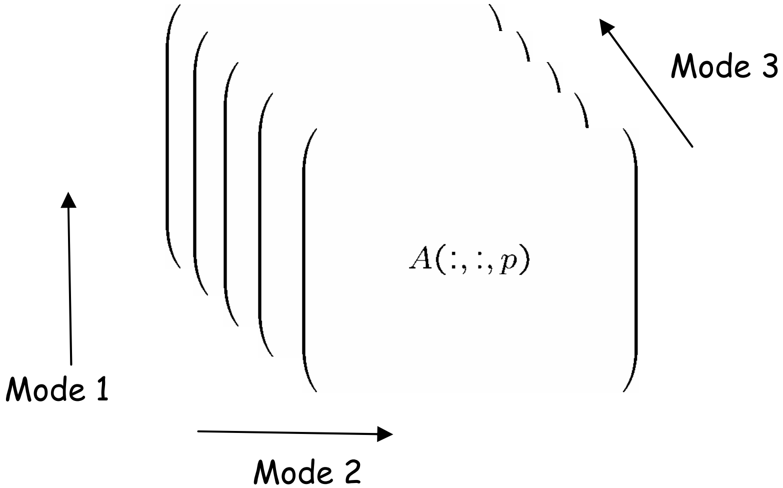
To construct landmarks, randomly sample with the "right" probabilities:

- $p_i = |A^{(i)}|^2 / \|A\|_F^2$
- $p_i \sim 1/|A^{(i)}|$ for outliers,
- uniform sampling.

Datasets modeled as tensors

Our goal:

Extract structure from a tensor dataset A using a small number of samples.



m x n x p tensor A

Q. What do we know about tensor decompositions?

A. Not much, although tensors arise in numerous applications.

Tensors

Tensors appear both in Math and CS.

- Represent high dimensional functions
- Connections to complexity theory (i.e., matrix multiplication complexity)
- Data Set applications (i.e., Independent Component Analysis, higher order statistics, etc.)

Also, many practical applications, e.g., Medical Imaging, Hyperspectral Imaging, video, Psychology, Chemometrics, etc.

However, there does not exist a definition of tensor rank (and associated tensor SVD) with the – nice – properties found in the matrix case.

Tensor rank

A definition of tensor rank

Given a tensor

$$A \in \mathcal{R}^{n_1 \times n_2 \times \ldots n_d}$$

find the minimum number of rank one tensors into it can be decomposed.

$$A = \sum_{i=1}^{r} u_1^i \otimes u_2^i \otimes \ldots \otimes u_d^i$$

outer product

- agrees with matrices for d=2
- related to computing bilinear forms and algebraic complexity theory.

BUT

- computing it is NP-hard
- only weak bounds are known
- tensor rank depends on the underlying ring of scalars
- successive rank one approximations are no good

Tensor α-rank

The α-rank of a tensor

Given

$$A \in \mathcal{R}^{n_1 \times n_2 \times \dots n_d}$$

create the "unfolded" matrix

$$A_{[\alpha]} \in R^{n_\alpha \times N_\alpha}$$

$$N_\alpha = \prod_{i \neq \alpha} n_i$$

and compute its rank, called the α-rank.

Pros:

- Easily computable,

Cons:

- different α-ranks are different,
- information is lost.

$$A \xrightarrow{\text{"unfold"}} n \underbrace{\left(\qquad A_{[\alpha]} \qquad \right)}_{n^{d-1}}$$

Tensors in real applications

3 classes of tensors in data applications

1. All modes are comparable (e.g., tensor faces, chemometrics)
2. A priori dominant mode (e.g., coarse scales, microarrays vs. time, images vs. frequency)
3. All other combinations

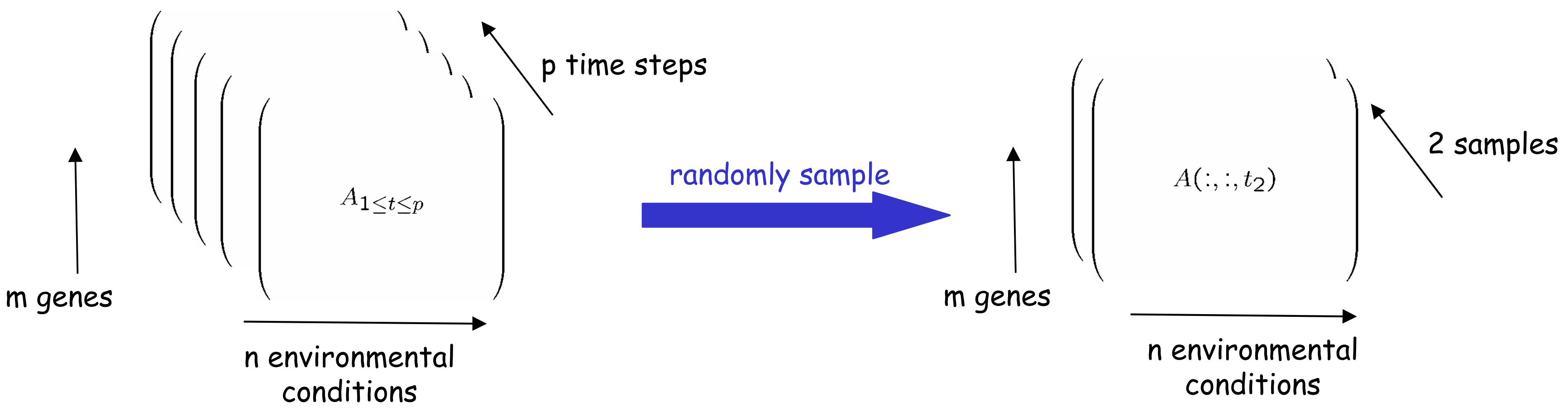
Drineas and Mahoney '04, TensorSVD paper deals with (1).

Drineas and Mahoney '04, TensorCUR paper deals with (2).

We will focus on (2), where there is a preferred mode.

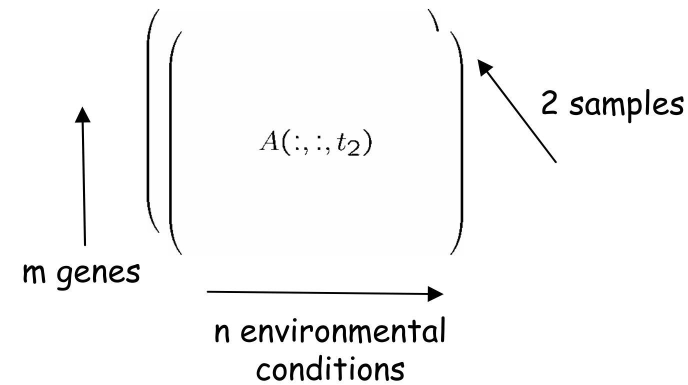
The TensorCUR algorithm (3-modes)

- Choose the preferred mode α (time)
- Pick a few representative snapshots: $p_t = \frac{\|A(:,:,t)\|_F^2}{\sum_{t=1}^{m} \|A(:,:,t)\|_F^2}$
- Express all the other snapshots in terms of the representative snapshots.



The TensorCUR algorithm (cont'd)

- Let R denote the tensor of the sampled snapshots.
- Express the remaining images as linear combinations of the sampled snapshots.

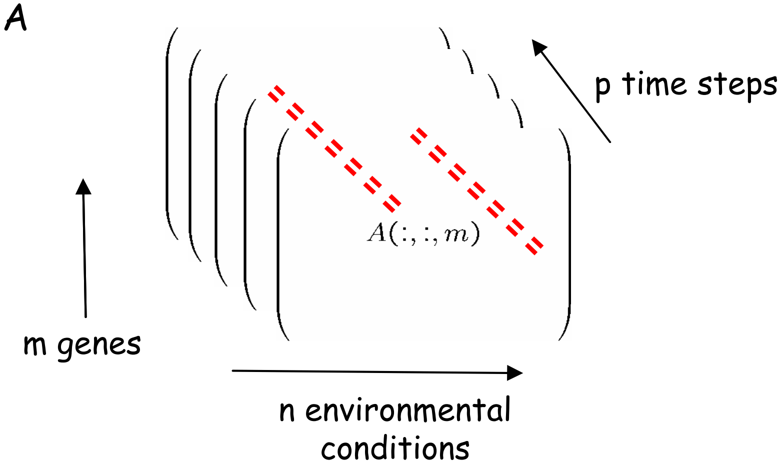


- First, pick a constant number of "fibers" of the tensor A (the red dotted lines).
- Express the remaining snapshots as linear combination of the sampled snapshots.

$$\min_u \sum_{i,j} \left(A(i,j,s) - \sum_{s \in R} u_s A(i,j,s)\right)^2$$

sampled fibers

sampled snapshots



The TensorCUR algorithm (cont'd)

Theorem:

$$\|A - CU \times_\alpha R\|_F^2 \leq \left\| A_{[\alpha]} - \left(A_{[\alpha]}\right)_{k_\alpha} \right\|_F^2 + \epsilon \|A\|_F^2$$

Unfold R along the α dimension and pre-multiply by CU

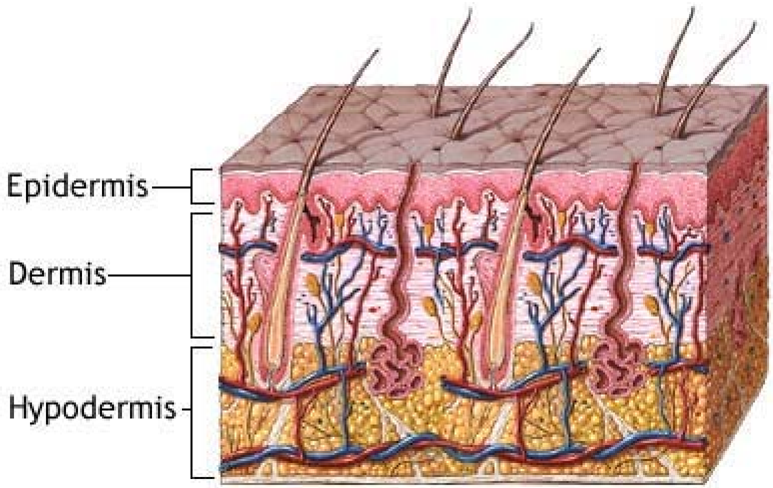
Best rank k_a approximation to $A_{[a]}$

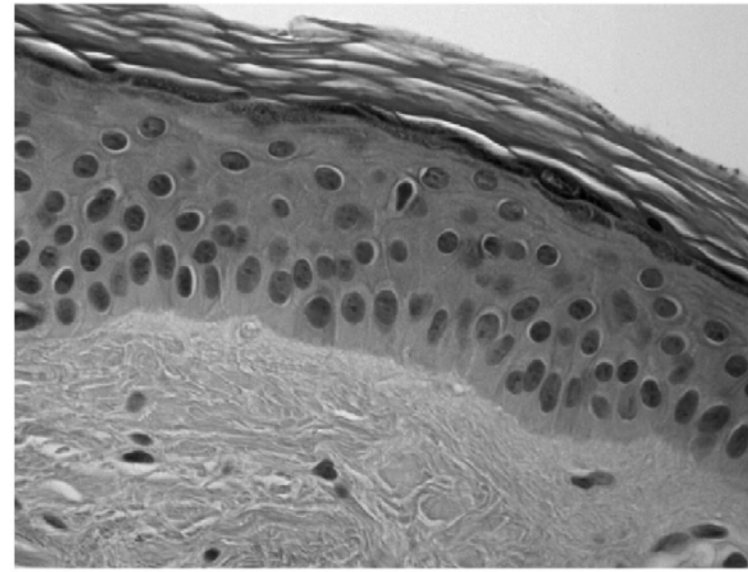
How to proceed:

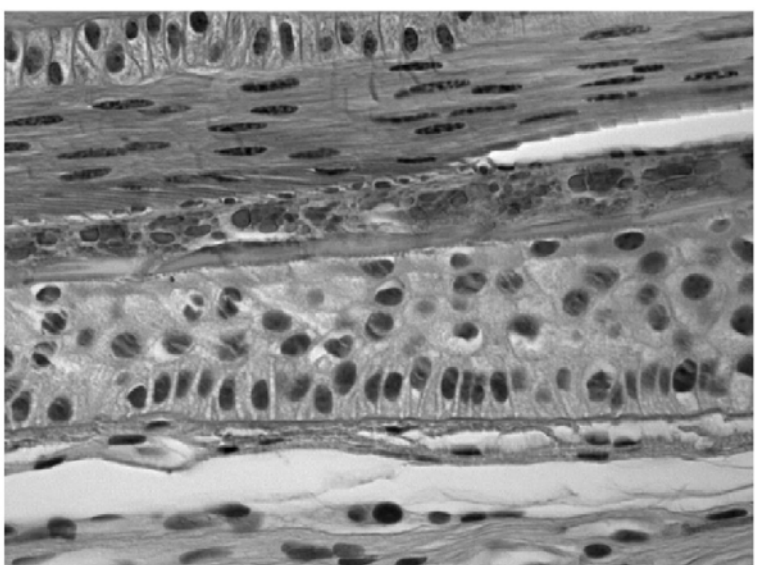
- Can recurse on each sub-tensor in R,
- or do SVD, exact or approximate,
- or do kernel-based diffusion analysis,
- or do wavelet-based methods.

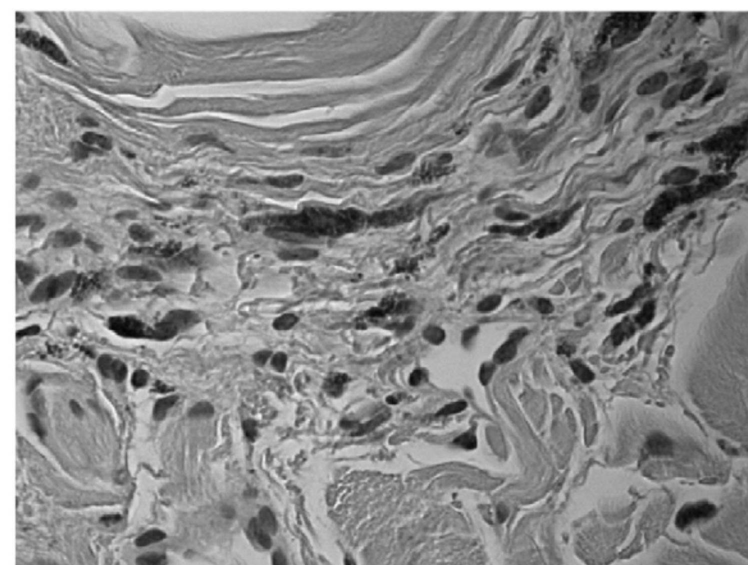
TensorCUR:

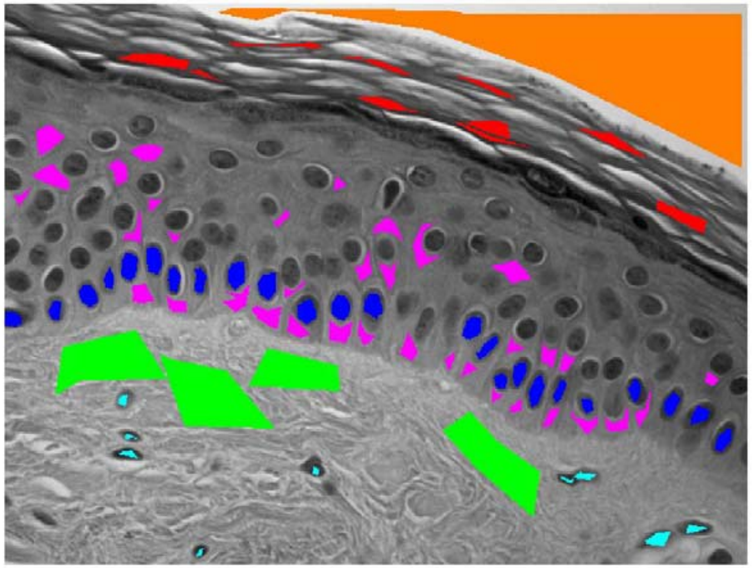
- Framework for dealing with very large tensor-based data sets,
- to extract a "sketch" in a principled and optimal manner,
- which can be coupled with more traditional methods of data analysis.

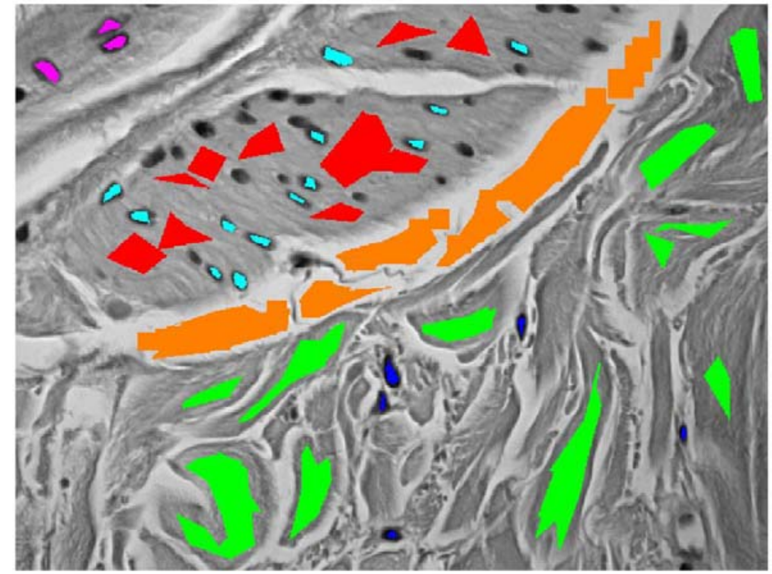

Epidermis
Dermis
Hypodermis

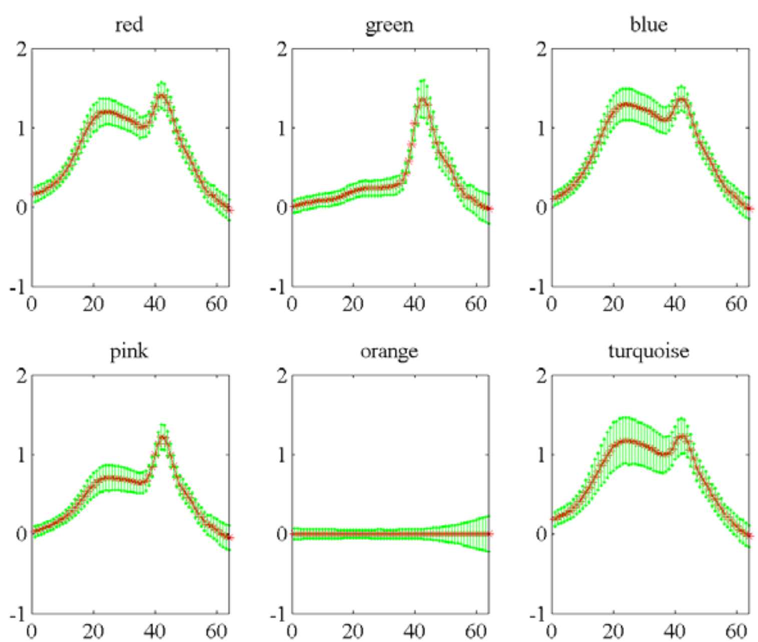


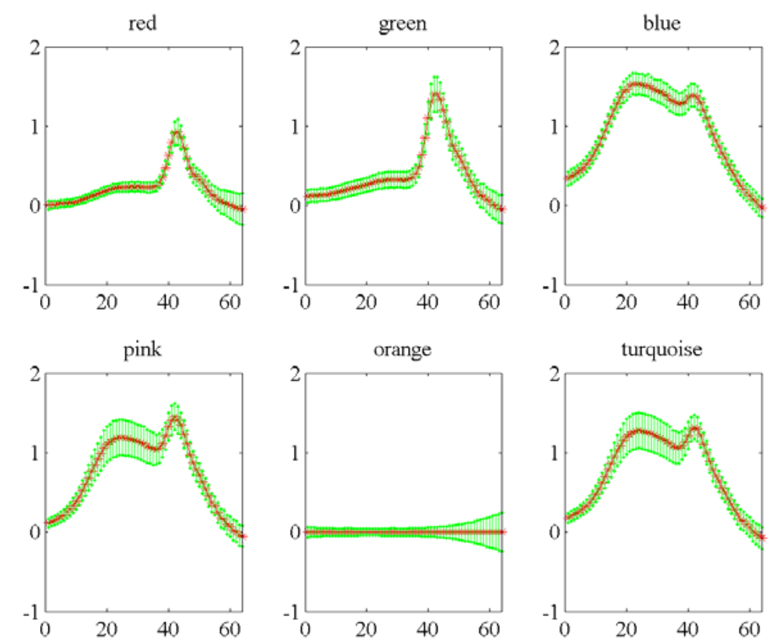









red
green
blue
pink
orange
turquoise


red
green
blue
pink
orange
turquoise

Apply *random sampling* methodology and *kernel-based diffusion maps* techniques to large physical and chemical and biological data sets.

Common application areas of large data set analysis:

- telecommunications,
- finance,
- web-based modeling, and
- astronomy.

Scientific data sets are quite different:

- with respect to their size,
- with respect to their noise properties, and
- with respect to the available field-specific intuition.

Data sets being considered:

- sequence and mutational data from G-protein coupled receptors
  - to identify mutants with enhanced stability properties,
- genomic microarray data
  - to understand large-scale genomic and cellular behavior,
- hyperspectral colon cancer data
  - to better represent large, complex visual data for improved detection of anomalous behavior, and
- simulational data
  - to more efficiently conduct large scale computations.