Stat260/CS294: Randomized Algorithms for Matrices and Data

Lecture 14 - 10/21/2013

Lecture 14: Additive-error Low-rank Matrix Approximation with Sampling and Projections

Lecturer: Michael Mahoney *Scribe: Michael Mahoney*

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

14 Additive-error Low-rank Matrix Approximation

Today, we will shift gears and begin to discuss RandNLA algorithms for low-rank matrix approximation. We will start with additive-error low-rank matrix approximation with sampling and projections. These are of interest historically and since they illustrate several techniques (norm-squared sampling, simple linear algebraic manipulations, the use of matrix perturbation theory, etc.), but they are much coarser than more recent finer bounds that can be obtained. Importantly, these additive-error bounds can be improved (and we will get to this soon). Depending on whether one is interesting in random sampling or random projection algorithms, the improvement comes either in the algorithm or in the analysis. Understanding this improvement will lead to a structural condition that extends the structural conditions for rectangular least squares problems to one for general "fat" matrices. As we will see, this condition underlies many of the theoretically and/or practically most interesting RandNLA algorithms for low-rank approximation.

Here is the reading for today and the next class.

- Drineas, Kannan, and Mahoney, "Fast Monte Carlo Algorithms for Matrices II: Computing Low-Rank Approximations to a Matrix"
- Deshpande and Vempala, "Adaptive Sampling and Fast Low-rank Matrix Approximation"

Today, in particular, we will cover the following topics.

- Basics of low-rank matrix approximation.
- Two simple matrix perturbation theory results.
- An overview of RandNLA methods for low-rank approximation.
- A basic random sampling algorithm and a quality-of-approximation result for it.

14.1 Basics of Low-rank Matrix Approximation

Since we are going to shift gears now and talk about how to use randomized algorithms to compute low-rank approximation of matrices, we will start with a brief overview of low-rank matrix approximation. Hopefully, this should just be a review to set notation, and in fact we have already covered some of these topics in our discussion of regression for very rectangular matrices, but we describe it here in detail since some of the details are different for matrices where both the number of rows and the number of columns are very large and of comparable size.

(BTW, this has mattered less in TCS and ML, where one is typically interested in quality-of-approximation metrics that depend only on how well one reproduces the top part of the spectrum, but it has mattered more in areas such as NLA and scientific computing, where one is also interested in controlling how the top and bottom parts of the singular subspace of the matrix versus approximated matrix interact.)

If $A \in \mathbb{R}^{m\times n}$, then there exist orthogonal matrices $U = [u^1 u^2 \dots u^m] \in \mathbb{R}^{m\times m}$ and $V = [v^1 v^2 \dots v^n] \in \mathbb{R}^{n\times n}$ where $\left\{u^t\right\}_{t=1}^m \in \mathbb{R}^m$ and $\left\{v^t\right\}_{t=1}^n \in \mathbb{R}^n$ are such that

$$U^T A V = \Sigma = \mathrm{Diag}(\sigma_1, \dots, \sigma_\rho), \tag{1}$$

where $\Sigma \in \mathbb{R}^{m\times n}$, $\rho = \min\{m, n\}$ and $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_\rho \geq 0$. Equivalently,

$$A = U\Sigma V^T.$$

The three matrices U, V, and Σ constitute the Singular Value Decomposition (SVD) of A. The σ_i are the singular values of A and the vectors u^i, v^i are the i-th left and the i-th right singular vectors, respectively. The columns of U and V satisfy the relations $Av^i = \sigma_i u^i$ and $A^T u^i = \sigma_i v^i$. For symmetric positive definite (or semi-definite) matrices the left and right singular vectors are the same. The singular values of A are the non-negative square roots of the eigenvalues of $A^T A$ and of AA^T. Furthermore, the columns of U, i.e., the left singular vectors, are eigenvectors of AA^T and the columns of V, i.e., the right singular vectors, are eigenvectors of $A^T A$.

The SVD can reveal important information about the structure of a matrix. If we define r by $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_\rho = 0$, then $\mathrm{rank}(A) = r$, $\mathrm{null}(A) = \mathrm{span}\{v^{r+1}, \dots, v^\rho\}$, and $\mathrm{range}(A) = \mathrm{span}\{u^1, \dots, u^r\}$. In particular, if we let $U_r \in \mathbb{R}^{m\times r}$ denote the matrix consisting of the first r columns of U, $V_r \in \mathbb{R}^{r\times n}$ denote the matrix consisting of the first r columns of V, and $\Sigma_r \in \mathbb{R}^{r\times r}$ denote the principal $r \times r$ sub-matrix of Σ, then

$$A = U_r \Sigma_r V_r^T = \sum_{t=1}^{r} \sigma_t u^t {v^t}^T. \tag{2}$$

(BTW, this is the usual linear algebraic notion of rank. Note, however, that one can also define other notions of "soft rank," sometimes called the "effective rank" and typically defined as the ratio of the Frobenius to spectral norm, both defined below, and this is sometimes of greater interest in machine learning and data analysis applications. We won't explicitly cover that much, but we note that many of the analysis tools we do discuss are also useful more or less directly for dealing with this softer notion of rank.)

Note that this dyadic decomposition property given in Eqn. (2) provides a canonical description of a matrix as a sum of r rank one matrices of decreasing importance. If $k \leq r$ and we define

$$A_k = U_k \Sigma_k V_k^T = \sum_{t=1}^{k} \sigma_t u^t {v^t}^T \tag{3}$$

then $A_k = U_k U_k^T A = \left(\sum_{t=1}^k u^t u^{tT}\right) A$ and $A_k = A V_k V_k^T = A \left(\sum_{t=1}^k v^t v^{tT}\right)$, i.e., A_k is the projection of A onto the space spanned by the top k singular vectors of A. Furthermore, the distance (as measured by both $\|\cdot\|_2$ and $\|\cdot\|_F$) between A and any rank k approximation to A is minimized by A_k, i.e.,

$$\min_{D \in \mathbb{R}^{m \times n}: \text{rank}(D) \leq k} \|A - D\|_2 = \|A - A_k\|_2 = \sigma_{k+1}(A) \tag{4}$$

and

$$\min_{D \in \mathbb{R}^{m \times n}: \text{rank}(D) \leq k} \|A - D\|_F^2 = \|A - A_k\|_F^2 = \sum_{t=k+1}^{r} \sigma_t^2(A). \tag{5}$$

Thus, A_k constructed from the k largest singular triplets of A is the optimal rank k approximation to A with respect to both $\|\cdot\|_F$ and $\|\cdot\|_2$. (It is actually optimal with respect to the more general class of unitarily-invariant norms.) More generally, one can also show that $\|A\|_2 = \sigma_1$ and that $\|A\|_F^2 = \sum_{i=1}^r \sigma_i^2$.

We reviewed the above since our main m.o. will be that we will project onto a set of random columns (or random linear combinations of columns) that is not the optimal set of columns, and we will show that the error incurred is not much worse than the optimal set of columns.

Finally, let's conclude this linear algebra review of general fat matrices with a brief word about the running time of computing the SVD and of low-rank approximations to the SVD. It might seem like this should be a simple question with a simple answer, but it is actually a rather complicated topic, and the answer depends on your background and your perspective as to what's important to count and what is acceptable to be ignored. Here are the key points for us to keep in mind.

- One can compute the full SVD of a general matrix A, in infinite precision arithmetic, with various direct methods, in $\Theta\left(n^3\right)$ time (or in $\min\{mn^2, m^2n\}$ time, if A is an $m \times n$ matrix) operations in the RAM model (which is actually *not* such a good model here). One can compute a rank-k low-rank approximation to A in that same $\Theta\left(n^3\right)$ time by computing the full SVD and keeping only those top k components of interest.

- Of course, that naive approach throws away a lot, but as an inequality the running time is $O\left(n^3\right)$ time. Alternatively, one must read the full input, and so the running time is $\Omega\left(n^2\right)$ for general matrices and $\Omega\left(\text{nnz}(A)\right)$, where $\text{nnz}(A)$ is the number of non-zero entries in the matrix A, for sparse matrices.

- Even aside from roundoff issues, this $O\left(n^3\right)$ running time provides a straw-man comparison, in the sense that one almost never needs the full SVD. Indeed, one almost never needs the full rank-k approximation to the SVD. Instead one typically only computes what one needs, e.g., a basis for the left or right singular subspace, the top k singular values, all or some of the diagonal elements for a projection matrix onto the span of A, an orthogonal matrix spanning a subspace close to the top k left or right singular subspace, etc.

- In these cases, some sort of iterative algorithm is typically used, and the running time of these iterative algorithms depends on lots of details about the matrix, the implementations, whether the matrix is represented explicitly or implicitly, whether communication is expensive, etc. (You got a feeling for some of these things when we discussed Blendenpik, but the situation is much more complex for general matrices.)

- We won't dwell on these issues too much, since we aren't focused primarily on implementations and since different disciplines say very different things about this. For the development of RandNLA algorithms (and matrix algorithms in machine learning and data analysis more generally) it is at least as important to understand some of the details/subtleties of what different research areas say is important for running time as it is to know any simple statement of running time.
- All that being said and as a rule of thumb (and ignoring things like condition number and other related issues), think of the running time as being roughly $\Theta(mnk)$, where the matrix A is of size $m \times n$ and where one is interested in a rank k approximation. Informally, you have to touch each entry once, and you have to touch each dimension once. Also, roughly, if the matrix is sparse, then the mn can be replaced with $\mathrm{nnz}(A)$. (We will see that, roughly, randomized algorithms can be used to improve the k to $\log(k)$, but they also have many other benefits—robustness, parallelizability, simplicity, etc.—that are at least as important as the running time improvements.)

14.2 Two Results from Matrix Perturbation Theory

Matrix perturbation theory has to do with how properties of a matrix such as its spectrum change (or are perturbed) as the elements of the matrix are varied. It is a large area, with many varied applications, and we will need only a very small part of it. In particular, from the perturbation theory of matrices it is known that the size of the difference between two matrices can be used to bound the difference between the singular value spectrum of the two matrices. More precisely, if $A, E \in \mathbb{R}^{m\times n}, m \geq n$, then:

$$\max_{t:1\leq t\leq n} |\sigma_t(A+E) - \sigma_t(A)| \leq \|E\|_2 \tag{6}$$

and

$$\sum_{k=1}^{n} (\sigma_k(A+E) - \sigma_k(A))^2 \leq \|E\|_F^2 . \tag{7}$$

The latter inequality is known as the Hoffman-Wielandt inequality.

Note that both of these results are of the form of the ℓ_p norm of the difference of the singular values of a matrix and a perturbed version of the matrix is bounded above by a matrix norm that equals the ℓ_p norm of the singular values of the perturbation. (These two results are for $p = \infty$ and $p = 2$, respectively.)

Note also that neither of these bounds depends on the structure of the perturbation. Given the large fraction of RandNLA algorithms that boil down to matrix perturbation results such as these two results, and given that the perturbations from random sampling or random projection that RandNLA algorithms perform are quite structured, it is of interest to see if one can get finer bounds by taking advantage of the structured form of the perturbation.

14.3 Randomization for Low-rank Matrix Approximation

In the randomized algorithms for low-rank approximation that we will discuss, there will be several "knobs," and the details of how these knobs are handled are important. Not only will those details make a big difference for how successful various algorithms are in theory and/or in practice, but—if not given appropriate attention—those same details can be the source of a great deal of confusion about how different algorithms related to each other.

The reason for this latter comment is that different research communities find it more or less natural/convenient to fiddle with different knobs in different ways, and (relatedly) different research communities find it more or less natural/convenient to ask for different types of quality-of-approximation guarantees. This has led to a confusing array of algorithms, which are often superficially quite different, but which in reality have very strong (algorithmic or statistical or structural) similarities and connections. In the next few weeks, we will try to focus on these commonalities, trying to highlight structural properties responsible for seemingly-different results.

As with our description of algorithms for least-square approximation, we will discuss algorithms for low-rank matrix approximation first in terms of basic random sampling algorithms, and then in terms of extensions to random projection algorithms.

Here are examples of different perspectives that people adopt on these algorithms.

- **Additive-error bounds versus relative-error bounds.** Let's say that we would like to quantify how well a matrix C, or perhaps the best rank k approximation to C if C has more than k columns, captures the top part of the spectrum of a matrix A, and (for now) let's say that we are interested in the error with respect to the Frobenius norm. One type of bound one could hope for is to show that

  $$\|A - P_{C_k}A\|_F \leq \|A - P_{U_k}A\|_F + \epsilon \|A\|_F . \quad (8)$$

  In the theory of algorithms, bounds of the form (8) are known as *additive-error bounds*, the reason being that the "additional" error (above and beyond that incurred by the SVD) is bounded above by an additive factor of the form ϵ times the scale $\|A\|_F$.

  Bounds of this form are very different and in general weaker than when the additional error enters as a multiplicative factor, such as when the error bounds are of the form

  $$||A - P_{C_k}A|| \leq f(m,n,k,\eta)||A - P_{U_k}A||,$$

  where $f(\cdot)$ is some function and η represents other parameters of the problem. Bounds of this type are of greatest interest when $f(\cdot)$ does not depend on m or n, in which case they are known as a *constant-factor bounds*, or when they depend on m and n only weakly. The strongest bounds are when $f = 1 + \epsilon$, for an error parameter ϵ, *i.e.*, when the bounds are of the form

  $$\|A - P_{C_k}A\|_F \leq (1+\epsilon) \|A - P_{U_k}A\|_F . \quad (9)$$

  These *relative-error bounds* are the gold standard (in TCS, but not necessarily in other areas), since the scale of the additive error becomes the residual error itself, and they provide a *much* stronger notion of approximation than additive-error or weaker multiplicative-error bounds.

- **Other notions of reconstruction quality.** Eqn. (8) and Eqn. (9) measure in two different ways how much of A is captured by the sample C, and they do so by measuring a particular norm (the Frobenius norm) of the difference between two matrices. Of course, one might be interested in other norms, e.g., the spectral norm, which is the largest singular value and thus the ℓ_∞ norm of the vector of singular values, or the trace/nuclear norm, which is the sum of the singular values and thus the ℓ_1 norm on the vector of singular values. (The Frobenius norm is the ℓ_2 or Euclidean norm on the singular value vector.) There are still other norms of interest. Alternatively, one might want to measure the quality in some other way, e.g., with respect to a divergence or whatever.

- **Reconstructing the matrix versus other notions of approximation quality.** Eqn. (8) and Eqn. (9) make statements about how well C captures the information in the top part of this spectrum of A. This is reasonable, but sometimes one is also interested in other things. (For example, we saw this before in ℓ_2 regression, where we wanted not just relative error on the objective function value, but we also wanted to say that the actual solution vectors were close.) In the case of low-rank approximation via U_k, the matrix consisting of the top k left singular vectors of A, in addition to capturing the maximum amount of A with respect to any unitarily-invariant matrix norm, U_k is also "good" for other reasons: the columns of U_k are orthogonal to each other and thus maximally "spread out," the matrix U_k is exactly orthogonal to the matrix $U_{k,\perp}$, where the latter is the matrix consisting of the bottom $m-k$ singular vectors, and thus the approximation is maximally far from the optimal residual subspace, etc.

  In more general low-rank matrix approximation methods, these considerations manifest themselves by, e.g., asking for an interpolative decomposition, where the condition number of the sample matrix C is relatively good, asking for a rank-revealing decomposition, where one shows that the singular values of the part of the matrix that is not captured are not too large, that there is not much overlap between the sample and the bottom $m-k$ singular directions of A, etc. Importantly, while these notions are all vaguely related and in many cases coincide when considering the exactly optimal SVD-based low-rank approximation, approximately optimizing one of them often says very little about exactly or approximately optimizing another one of them. Depending on the downstream application, one or the other of these objectives might be of greatest interest.

  These observations hold in general, e.g., with deterministic algorithms like rank-revealing QR decompositions, but we will be mostly interested in how they hold for randomized algorithms via using random sampling or random projections. We will discuss how to deal with some of these issues with RandNLA algorithms. As we will see, in some cases this difference manifests itself in the algorithm, while in other cases it manifests itself in the analysis.

- **Sampling versus projection.** As we saw before, roughly, random projections correspond to uniform sampling in randomly rotated spaces. The same holds true, again at one level of granularity, in the case of randomized algorithms for low-rank approximation. Indeed, that perspective is often a helpful way to think about similarities between seemingly-different problems and algorithms, and so we will emphasize that perspective. But, if we get greedy in optimizing various factors (e.g., as needs to be done with the oversampling parameter when providing high-quality implementations), then sometimes it is better to do it directly and not view it as this two step process. Alternatively, it is sometimes convenient (e.g., in scientific computing) to view a random projection as providing an estimator for the range space of a matrix; and it is sometimes convenient (e.g., in TCS) to view a random projection as providing a data-oblivious or data-agnostic subsapce embedding. Whether projections directly "boil down" to sampling uniformly in a randomly-rotated space or do so only indirectly and approximatelly, it is helpful to think of sampling and projections on a similar footing and providing two different types of randomized "sketching" matrices.

- **More aggressive downsampling.** Sometimes, we are interested in sampling fewer than roughly $O(k \log k/\epsilon^2)$ or even $O(k/\epsilon)$ columns. For example, we might want to sample *exactly* k columns, or we might want to project onto *exactly* $k+p$ columns, where p is a small integer like 5 or 10. In these cases, the simplest analysis typically fails for worst-case matrices, and this is basically for coupon collector reasons, but this might be ok if there is a quick way to check whether some property has been satisfied. This is sometimes of interest by themselves and sometimes for their numerical properties (as we saw with LS), and we need to control different structures—consider CSSP and "slow" extensions to projections.

- **Reproducing the data versus reproducing hypothesized data.** Eqn. (8) and Eqn. (9) are statements about how well an algorithm does with respect to the data sitting in front of us. This is a very natural thing to ask for in NLA and TCS. Alternatively, one might be interested in how well the algorithm does with respect to hypothesized but unseen data. This latter approach is more natural in statistics and machine learning. The MSE, the usefulness of the low-rank approximation in a prediction task such as kernel ridge regression, etc., are all examples of such a metrics, and there are many others.

14.4 The LinearTimeSVD Algorithm

We will start with a very simple random sampling algorithm. Given an $m \times n$ matrix A, we wish to choose columns of A such that the projection of the matrix onto those columns "captures" as much of the matrix as possible, i.e., that is a basis for a space close to the space spanned by the top singular vectors of the matrix. Thus, in particular, if A is well approximated by a low-rank matrix, then we would like $A \approx P_{span(C)}A$, where C is a matrix consisting of the chosen columns and where $P_{span(C)}$ is a projection onto the column space of C. To this end, the LinearTimeSVD algorithm randomly samples a small number of columns from an input matrix, and it returns an approximation to the singular values and left singular vectors of that matrix. This algorithm is somewhat too simple to have found widespread use in practice (for reasons we will discuss), but it is important historically, and it is pedagogically convenient since it's analysis will illustrate several important concepts.

Algorithm 1 The LinearTimeSVD Algorithm.

```
Input: An $m \times n$ matrix $A$, integers $c, k$ s.t. $1 \le k \le c \le n$, and a probability distribution $\{p_i\}_{i=1}^n$.
Output: An $m \times k$ orthogonal matrix $H_k$ and numbers $\sigma_t(C), t = 1, \dots, k$.
1: for $t = 1$ to $c$ do
2:     Pick $i_t \in 1, \dots, n$ with $\mathbf{Pr}[i_t = \alpha] = p_\alpha$, $\alpha = 1, \dots, n$.
3:     Set $C^{(t)} = A^{(i_t)}/\sqrt{cp_{i_t}}$.
4: end for
5: Compute $C^TC$ and its singular value decomposition; say $C^TC = \sum_{t=1}^{c} \sigma_t^2(C) y^t {y^t}^T$.
6: Compute $h^t = Cy^t/\sigma_t(C)$ for $t = 1, \dots, k$.
7: Return $H_k$, where $H_k^{(t)} = h^t$, and $\sigma_t(C), t = 1, \dots, k$.
```

We have formulated this algorithm to say that it returns the top k left singular vectors of C, but we could have just returned the matrix C. (By that, we mean that the quality-of-approximation and running time claims that we discuss today and next time work for both C and C_k. If we were interested in different objectives, like we just discussed above, then the difference between C and C_k can become important. We will revisit this issue later with other low-rank approximation algorithms.)

The point here is that we want to say that the matrix C is in some sense close to the matrix A. It is not immediately obvious how to make that comparison, however, given that the two matrices have different dimensions. Note, though, that the ambient dimensionality of the range space of both matrices is the same, i.e., $\mathbb{R}^m$, and so we will say that they are similar if their left singular subspaces are similar, or relatedly if $CC^T \approx AA^T$. (Other notions of similarity are certainly possible, but this notion says that the two matrices have a similar correlational structure on the non-sampled dimension, and one important aspect of this notion is that we can relate it back to approximation algorithms for the matrix multiplication primitive.)

Before presenting our quality-of-approximation results, here is a summary of the running time of this algorithm.

- If we work with probabilities that are approximately proportional to the squared Euclidean norms of the columns of A, as in Eqn. (14) below, then one pass and $O(c)$, where c is the number of random samples to be drawn, i.e., the number of independent counters that are run in parallel in the pass efficient model, additional space and time are needed to choose the indices of the columns to choose. (Of course, if we work with uniform sampling probabilities, then we can choose the columns to be sampled without even looking at the data and store the indices of those columns in $O(c)$ additional space.)
- Given the indices of the columns to be sampled, then one additional pass and $O(mc)$ additional space and time is needed to select the columns from A and construct the matrix C.
- Given the matrix C, then computing C^TC requires $O(mc^2)$ additional space and time, and computing the SVD of C^TC requires $O(c^3)$ additional space and time.
- Given the SVD of C^TC, then computing H_k requires k matrix-vector multiplications, for a total of $O(mck)$ additional space and time.
- So, on the whole, if $c, k = O(1)$, then $O(m)$ additional space and time are needed. That is, the LinearTimeSVD algorithm has additional running space and time that is linear in the dimensionality of the data/features and not in the size or number of non-zeros of the matrix.

Next, we will be interested in establishing quality-of-approximation results for this algorithm. To separate clearly the effect of linear algebraic structure from the effect of randomization on the quality-of-approximation claims, we will first do this for general sampling probabilities, and we will then specialize the result to the case that the probabilities depend on the Euclidean norms squared of the columns of A. In the former case, the additional error, above and beyond that incurred by the best rank-k approximation, will depend on $||AA^T - CC^T||_\xi$, for $\xi = \{2, F\}$. Then, we will call our previous matrix multiplication results to bound that additional error. So, the choice of columns will enter only in the form of an approximate matrix multiplication bound.

Let's start with what we can establish for the Frobenius norm error.

Theorem 1 *Suppose $A \in \mathbb{R}^{m\times n}$ and let H_k be constructed from the* LinearTimeSVD *algorithm. Then,*

$$\left\|A - H_kH_k^TA\right\|_F^2 \leq \|A - A_k\|_F^2 + 2\sqrt{k}\left\|AA^T - CC^T\right\|_F .$$

Proof: Recall that for matrices X and Y, $\|X\|_F^2 = \mathbf{Tr}\left(X^T X\right)$, $\mathbf{Tr}\left(X+Y\right) = \mathbf{Tr}\left(X\right) + \mathbf{Tr}\left(Y\right)$, and also that $H_k^T H_k = I_k$. Thus, we may express $\left\|A - H_k H_k^T A\right\|_F^2$ as:

$$\begin{aligned}
\left\|A - H_k H_k^T A\right\|_F^2 &= \mathbf{Tr}\left((A - H_k H_k^T A)^T (A - H_k H_k^T A)\right) \\
&= \mathbf{Tr}\left(A^T A - 2A^T H_k H_k^T A + A^T H_k H_k^T H_k H_k^T A\right) \\
&= \mathbf{Tr}\left(A^T A\right) - \mathbf{Tr}\left(A^T H_k H_k^T A\right) \\
&= \|A\|_F^2 - \left\|A^T H_k\right\|_F^2 . \qquad (10)
\end{aligned}$$

We may relate $\left\|A^T H^k\right\|_F^2$ and $\sum_{t=1}^k \sigma_t^2(C)$ by the following:

$$\begin{aligned}
\left| \left\|A^T H_k\right\|_F^2 - \sum_{t=1}^k \sigma_t^2(C) \right| &\leq \sqrt{k} \left(\sum_{t=1}^k \left(\left\|A^T h^t\right\|_2^2 - \sigma_t^2(C) \right)^2 \right)^{1/2} \\
&= \sqrt{k} \left(\sum_{t=1}^k \left(\left\|A^T h^t\right\|_2^2 - \left\|C^T h^t\right\|_2^2 \right)^2 \right)^{1/2} \\
&= \sqrt{k} \left(\sum_{t=1}^k \left({h^t}^T (AA^T - CC^T) h^t \right)^2 \right)^{1/2} \\
&\leq \sqrt{k} \left\|AA^T - CC^T\right\|_F . \qquad (11)
\end{aligned}$$

The first inequality follows by applying the Cauchy-Schwartz inequality; the last inequality follows by writing AA^T and CC^T with respect to a basis containing $\{h^t\}_{t=1}^k$. By again applying the Cauchy-Schwartz inequality, noting that $\sigma_t^2(X) = \sigma_t(XX^T)$ for a matrix X, and applying the Hoffman-Wielandt inequality, (7), we may also relate $\sum_{k=1}^k \sigma_t^2(C)$ and $\sum_{k=1}^k \sigma_t^2(A)$ by the following:

$$\begin{aligned}
\left| \sum_{t=1}^k \sigma_t^2(C) - \sum_{t=1}^k \sigma_t^2(A) \right| &\leq \sqrt{k} \left(\sum_{t=1}^k \left(\sigma_t^2(C) - \sigma_t^2(A) \right)^2 \right)^{1/2} \\
&= \sqrt{k} \left(\sum_{t=1}^k \left(\sigma_t(CC^T) - \sigma_t(AA^T) \right)^2 \right)^{1/2} \\
&\leq \sqrt{k} \left(\sum_{t=1}^m \left(\sigma_t(CC^T) - \sigma_t(AA^T) \right)^2 \right)^{1/2} \\
&\leq \sqrt{k} \left\|CC^T - AA^T\right\|_F . \qquad (12)
\end{aligned}$$

Combining the results of (11) and (12) allows us to relate $\left\|A^T H_k\right\|_F^2$ and $\sum_{t=1}^k \sigma_t^2(A)$ by the following:

$$\| \left\|A^T H_k\right\|_F^2 - \sum_{t=1}^k \sigma_t^2(A) \| \leq 2\sqrt{k} \left\|AA^T - CC^T\right\|_F . \qquad (13)$$

Combining (13) with (10) yields the theorem.

$\diamond$

Let's conclude today with several observations about this theorem.

- This theorem says that the error in any low-rank approximation, above and beyond that provided by the best rank-k approximation, can be related to an error in approximating the product of two matrices. Thus, if we can make that matrix multiplication error small, then we get a good low-rank matrix approximation.

- In particular, if we use probabilities $\{p_i\}_{i=1}^{n}$ that are close to the Euclidean norms squared of A, in the sense that
$$p_i \geq \beta \frac{\left\|A^{(i)}\right\|_2^2}{\|A\|_F^2}, \tag{14}$$
for some positive $\beta \leq 1$ (e.g., just set $\beta = 1$ and use the Euclidean norms squared of the columns of A), then one has that worst-case additive-error bounds of the form
$$\left\|A - H_k H_k^T A\right\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon \|A\|_F^2 \tag{15}$$
hold in expectation and with high probability, if one chooses $c \gtrsim k/\epsilon^2$ in the algorithm.

- If the Euclidean norms of the columns of A are approximately uniform, then uniform sampling is approximately optimal, in the sense that the probabilities $p_i = 1/n$ are close to the probabilities $p_i = \left\|A^{(i)}\right\|_2^2 / \|A\|_F^2$, e.g., in the sense that β is not too small (and thus $1/\beta$ is not too large). Then, with an appropriately small choice of β that can be absorbed into the sampling complexity, one can get bounds of the form Eqn. (15) with $c \gtrsim k/\beta\epsilon^2$ (for a value of β which doesn't make this expression too large).

- On the other hand, if the uniform sampling probabilities are very different than the norm-squared probabilities, then one must choose β to be very small (and thus $1/\beta$ to be very large) to get bounds of the form Eqn. (15). For example, $c \gtrsim k/\beta\epsilon^2$—where the value of β makes this very large. In particular, this can be a very large number of uniformly-sampled columns if β depends on n (in theory) of if, say, $\beta = 1/1000$ (in practice).

- Alternatively, one can sample $c \gtrsim k/\epsilon^2$ columns uniformly, i.e., where c has no β dependency, and obtain bounds of the form
$$\left\|A - H_k H_k^T A\right\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon n^2. \tag{16}$$
This is also an additive-error bound, but the scale of the additive error is *much* worse that before—so much worse, in fact, that bounds with additive scale factor don't even provide a qualitative guide to the practical performance of the algorithm.