Math 5601 Independent Study Project

Jacob Hauck

1 Introduction

The problem with matrices

As we have seen throughout Math 5601, many numerical problems ultimately boil down to solving a system of linear equations $Ax = b$ for an unknown vector x. We also saw that most numerical methods for solving this problem involve performing many computations on the entries of the matrix A.

Performing some process on a matrix A is common outside of simply solving a linear system of equations, as matrices appear naturally in the analysis of many important and difficult problems. In many of these situations, the matrix A might be enormous, and it may be necessary to process many such matrices.

In some cases, the vast majority of the entries of the matrix A are zero, as we see in the final project, where the stiffness matrix has mostly zero entries. If this is the case, then the matrix is called **sparse**. Not all matrices one encounters are sparse, however. For example, to work with the samples on a uniform grid of a function defined on the unit cube $[0, 1]^3$ with an apparently innocuous resolution of $\frac{1}{1000}$ effectively requires a stack of 1000 separate 1000×1000 matrices, which have altogether 10^9 entries, which could consume the entirety of some systems' memory if each entry were stored as an individual, 64-bit, floating point number.

If A is a square $n \times n$ matrix[1], then typical processes operating on A require at least $\mathcal{O}(n^2)$ operations. Indeed, simply storing a matrix in memory by creating a list of its entries already has $\mathcal{O}(n^2)$ complexity, so it is no surprise that more complex operations incur at least as much time and memory as is required for storage alone.

Considering that n may be so large that the entries of the matrix might not even fit into available memory, it is clear that more efficient methods for performing computations on matrices are desirable, if not indispensable.

Compressed representations

If a matrix is very sparse, in the sense that most of its entries are zero, we have no need to store all the entries. Instead, we can store only the nonzero entries, along with the locations of those entries. This can lead to an enormously compressed representation of a matrix, with the cost in time and memory of storing the matrix being $\mathcal{O}(s)$, where s is the number of nonzero entries. Moreover, if the format of the sparse data format is chosen intelligently, then many common matrix algorithms can be rewritten

[1] For simplicity, we focus on square matrices in complexity estimates, but matters are essentially the same with non-square matrices as well.

so that their memory costs are reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(ns)$ (or lower); the reduction from quadratic cost in n to something less than quadratic can lead to tremendous gains in performance, often making otherwise infeasible problems feasible (depending on how s is related n, which varies by case).

To give a simple example, computing the matrix-vector product Ac of the $n \times n$ matrix and the $n \times 1$ column vector c can be done in $\mathcal{O}(n+s)$ operations if A is known to havee only s nonzero entries. Since this operation is used in many larger algorithms, like iterative solvers for the linear system $Ax = b$, we immediately see the significance of this performance gain. Other algorithms experience similar improvements.

What, then, to do about matrices that are not sparse? Even if a matrix has no nonzero entries, it may still have a rank r much lower than n. In this case, only r of the columns (or rows) of the matrix are linearly independent; that is, the *other* $n - r$ columns can be written as a linear combination of the r linearly independent ones. By storing only the r linearly independent columns and the $(n - r)r$ linear combination coefficients determining the other $n - r$ columns, the matrix can be stored with a time and memory cost of $\mathcal{O}(nr)$.

Again, depending on the relationship between r and n, this reduction from quadratic to less than quadratic complexity in a basic operation like storage can lead to massive performance benefits in other algorithms. It can be beneficial, that is, *if* the compressed data format can be used effectively with common matrix operations.

This is all good and well if the matrix has a low rank, but what if the matrix has a high or full rank?

Lossy compression

Even if a matrix is not sparse and not low-rank, we may still be able to approximate the matrix using a compressed representation (though representing it exactly may be off the table). In many cases, an approximation is sufficient, especially if we have a useful theoretical bound on the error of the approximation.

Since the low-rank compressed representation didn't require sparsity but still achieved good compression, we might look to approximate a high-rank matrix by a low-rank one, then represent that low-rank approximation by an exact compressed form. This idea is intimately related to the following important and well-known theorem.

Theorem 1. (Singular value decomposition) *Let $A \in \mathbf{R}^{n\times m}$. Then there exist orthogonal $U \in \mathbf{R}^{n\times n}$ and $V \in \mathbf{R}^{m\times m}$, and a matrix $\Sigma \in \mathbf{R}^{n\times m}$ defined by $\Sigma_{ii} = \sigma_i \geq 0$ for $1 \leq i \leq \min\{m, n\}$ and $\Sigma_{ij} = 0$ otherwise, such that*

$$A = U\Sigma V^T. \tag{1}$$

Furthermore, the best rank-r approximation of A in both Frobenius and spectral norm is given by

$$A' = U\Sigma' V^T, \tag{2}$$

where $\Sigma'_{ij} = \Sigma_{ij}$ if $\max\{i, j\} \leq r$, and $\Sigma'_{ij} = 0$ otherwise. Finally,

$$\|A - A'\|_2 = \sigma_{r+1}, \qquad \|A - A'\|_F = \left(\sum_{i>r} \sigma_i^2\right)^{\frac{1}{2}}, \tag{3}$$

where $\|\cdot\|_2$ and $\|\cdot\|_F$ are the spectral and Frobenius matrix norms, and we set $\sigma_{\min\{m,n\}+1} = 0$ by convention.

This theorem gives us some idea as to how to construct a good, indeed, the *best* (in some sense), approximation of A by a low-rank matrix. The decomposition (1) is called the **singular value decomposition (SVD)** of A, and (2) is called the **(rank-r) truncated singular value decomposition** of A. The numbers $\{\sigma_i\}_{i=1}^{\min\{m,n\}}$ are called the **singular values** of A, and are unique up to permutation.

Not only does Theorem 1 provide an idea as to how to obtain a low-rank approximation, it also provides the *exact value* of the error of such an approximation. This has made SVD a go-to tool in the approximation of matrices.

One shortcoming of SVD, however, is that the matrices U and V can be difficult to interpret, and, in general, practical algorithms for computing the SVD of a square $n \times n$ matrix A have a time complexity $\mathcal{O}(n^2)$, which, as we have seen, may be problematic in some applications.

A fast compression method

In the ideal scenario, we would like too have not only a compressed representation of a square $n \times n$ matrix A, we would also like to *construct* that compressed representation efficiently. If the construction of the compressed representation is too slow, whatever process we want to do with our matrix may still be impossible. How, though, could we possibly do better than $\mathcal{O}(n^2)$ time complexity in constructing an approximate representation of a square $n \times n$ matrix, considering that even storing such a matrix already requires $\mathcal{O}(n^2)$ operations?

Enter: submatrices. A submatrix is a smaller matrix formed from the elements of a larger matrix. A submatrix of a matrix A can be used to form a low-rank approximation of the larger matrix through the so-called **psuedo-skeleton decomposition** [3]. Although the pseudo-skeleton decomposition is, by Theorem 1, not the optimal low-rank approximation, we will see that it can still be pretty good.

The main advantage that pseudo-skeleton decomposition enjoys over SVD is that it can be computed very quickly. Indeed, using the algorithm described in this paper, we will see that a rank-r psuedo-skeleton decomposition can be computed in $\mathcal{O}(nr^2)$ operations, which is significantly faster than SVD if r is much smaller than n, and it is nearly as low as the $\mathcal{O}(nr)$ cost of the resulting compressed representation.

More importantly, because pseudo-skeleton decomposition only involves selecting elements from the original matrix to form a submatrix, it is not even necessary to know (or load into memory) every element of the original matrix. In other words, if the matrix being compressed has a good low-rank approximation, then a good pseudo-skeleton decomposition can be computed *without event looking at the entire matrix.*

In SVD, by contrast, every element of the U and V matrices depends on every element of the original matrix, so computing the SVD almost unavoidably involves looking at every entry of the original matrix. We also remark that the pseudo-skeleton decomposition is based on a submatrix, whose elements are just elements of the original matrix, which is easier to interpret than the more opaque U and V matrices in the SVD.

In this project, we will investigate an algorithm known as `maxvol` [2], whose purpose is to find a submatrix that is “good enough” for the corresponding psuedo-skeleton decomposition to approximate the original matrix well without incurring a computational cost of much greater complexity than is required to store the psuedo-skeleton decomposition (unlike truncated SVD). We will discuss the theory and motivation for the algorithm, provide a `NumPy` implementation, and perform some numerical experiments to demonstrate the power of the algorithm.

Before we move on, we remark that the theory and implementation in this project are woven together from various works, especially those of Goreinov, Oseledets, Tyrtyshnikov and others [3, 2, 9, 7, 8, 4]. We provide one general citation here at the beginning, as many of the proofs and sections below draw on ideas from multiple sources and mix them with my own work to fill in their missing details and connect related results and concepts, making the citation of every specific idea overly complicated. Nevertheless, I have done my best to refer major results to the sources that seem most appropriate.

2 Theory

Before we can discuss what a pseudo-skeleton decomposition is and how to find one, we need introduce some notation for describing submatrices. We begin with a concise notation for referring to a set of indices in a matrix.

Definition 1. (Index slice) *Let $m \leq n$, where m and n are integers. Define the* ***index slice from** m **to** n* *by the sequence*

$$m : n = \{i\}_{i=m}^{n}. \tag{4}$$

It will also be convenient to have a concise notation for the standard basis vectors.

Definition 2. (Standard basis) *Let $e_j \in \mathbf{R}^n$ denote the jth standard basis vector in $\mathbf{R}^n$, defined by*

$$(e_j)_i = \begin{cases} 1 & i = j, \\ 0 & i \neq j. \end{cases} \tag{5}$$

Be aware that we will reuse the symbol e_j for different values of n, sometimes in the same context.

Next, we give a formal definition of a submatrix. We will frequently need to use both the submatrix itself in matrix formulas and its corresponding row and column indices, so we define a compact indexing notation relating a matrix and the submatrix along given rows anad columns.

Definition 3. (Submatrix) *Let $A \in \mathbf{R}^{m \times n}$ be a matrix. Let $I = \{I_i\}_{i=1}^{r}$ be a sequence of distinct row indices of A, and let $J = \{J_j\}_{j=1}^{c}$ be a sequence of distinct column indices of A. The* ***submatrix of** A **with rows** I **and columns** J* *is the matrix $A(I, J) \in \mathbf{R}^{r \times c}$ with entries*

$$[A(I, J)]_{ij} = A_{I_i J_j}. \tag{6}$$

If the special symbol $:$ is used as row indices or column indices, it means the entire sequence $1 : m$ or $1 : n$.

If I or J is a single integer i or j instead of a sequence, we take this to mean $I = i : i$ or $J = j : j$, the sequence consisting of that one integer.

2.1 Skeleton Decomposition

The following is a simple theorem about matrix multiplication with submatrices; it is trivial, but we will use it many times in the subsequent proofs, and it also serves as the motivation for the form of the pseudo-skeleton decomposition.

Lemma 1. (Submatrix of a product) *Let $A \in \mathbf{R}^{m\times r}$, and let $B \in \mathbf{R}^{r\times n}$. Then for any row indices I of A,*

$$(AB)(I,:) = A(I,:)B, \tag{7}$$

and for any column indices J of B,

$$(AB)(:,J) = AB(:,J). \tag{8}$$

Proof. Observe that

$$[(AB)(I,:)]_{ij} = (AB)_{I_i j} = \sum_{k=1}^{r} A_{I_i k} B_{kj} = \sum_{k=1}^{r} A(I,:)_{ik} B_{kj} = [A(I,:)B]_{ij}, \quad i \in 1:m,\ j \in 1:n, \tag{9}$$

so $(AB)(I,:) = A(I,:)B$.

Similarly,

$$[(AB)(:,J)]_{ij} = (AB)_{iJ_j} = \sum_{k=1}^{r} A_{ik} B_{kJ_j} = \sum_{k=1}^{r} A_{ik} B(:,J)_{kj} = [AB(:,J)]_{ij}, \quad i \in 1:m,\ j \in 1:n, \tag{10}$$

so $(AB)(:,J) = AB(:,J)$. □

This simple result makes it easy to construct a low-rank matrix that agrees with a given submatrix on the **cross** of the submatrix, which are the entries in the original matrix in the same row or column as any entry in the submatrix. To give a concrete example, suppose that

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} \end{bmatrix}, \tag{11}$$

and consider the submatrix on the entries

$$A((1,4),(2,5)) = \begin{bmatrix} A_{12} & A_{15} \\ A_{42} & A_{45} \end{bmatrix}. \tag{12}$$

Using this submatrix, we can easily apply Lemma 1 to construct a low-rank approximation S (we will see how shortly in Theorem 2) that agrees with A on the cross of the submatrix. In other words, coloring every entry of S green that agrees with A, we have

$$S = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ S_{21} & A_{22} & S_{23} & S_{24} & A_{25} \\ S_{31} & A_{32} & S_{33} & S_{34} & A_{35} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} \\ S_{51} & A_{52} & S_{53} & S_{54} & A_{55} \end{bmatrix}. \tag{13}$$

From this, it is evident that S agrees with A on the row and column emanating from every element in the submatrix $A((1,4),(2,5))$; this is where the name "cross" comes from.

To construct S, we simply take the columns of the submatrix and the rows of the submatrix and put the inverse in between. When we apply Lemma 1, we see that S and A agree on the cross of the submatrix.

Theorem 2. *Let $A \in \mathbf{R}^{m\times n}$. If $A(I,J) \in \mathbf{R}^{r\times r}$ is a square submatrix of A with rank r, and $S = A(:,J)A(I,J)^{-1}A(I,:)$, then*

$$A(I,:) = S(I,:), \qquad A(:,J) = S(:,J). \tag{14}$$

Proof. By Lemma 1,

$$S(I,:) = (A(:,J)GA(I,:))(I,:) = A(I,J)A(I,J)^{-1}A(I,:) = A(I,:), \tag{15}$$

and

$$S(:,J) = (A(:,J)GA(I,:))(:,J) = A(I,:)A(I,J)^{-1}A(I,J) = A(:,J). \tag{16}$$

□

To give a more fanciful interpretation of the green rows and columns in (13), we can think about them as the "bones" of the matrix A, in the sense that they contain most of the essential "structure" of the matrix.

Indeed, if A has the same rank as the submatrix S, then S and A are in fact identical, meaning that essentially all the information in A is contained in the submatrix, which provides a highly compressed representation of the original matrix. In this cse, the representation S is known as the **skeleton decomposition** of the matrix, after the bone analogy above.

Theorem 3. (Skeleton decomposition) *Let $A \in \mathbf{R}^{m\times n}$ be a matrix with rank r. If $A(I,J) \in \mathbf{R}^{r\times r}$ is a square submatrix of A with rank r, then*

$$A = A(:,J)A(I,J)^{-1}A(I,:). \tag{17}$$

This factorization is called a ***skeleton decomposition***[2] *of A.*

Proof. The columns of A at indices J (that is, $\{A(:,J_j)\}_{j=1}^r$) are linearly independent because

$$\sum_{j=1}^r \alpha_j A(:,J_j) = 0 \implies \sum_{j=1}^r \alpha_j A(I,J_j) = 0 \implies \alpha_j = 0, \quad j \in 1:r \tag{18}$$

because the columns $\{A(I,J_j)\}_{j=1}^r$ of $A(I,J)$ must be linearly independent by the fact that $A(I,J)$ has rank r.

Thus, since A has rank r, every other column of A must be a linear combination of the columns at indices J. That is, there exists $\{\alpha_{\ell j}\}$ for $j \in 1:r$ and $\ell \in 1:n$ such that

$$A(:,\ell) = \sum_{j=1}^r \alpha_{\ell j} A(:,J_j). \tag{19}$$

[2] Since $C = A(:,J)$ is a matrix of columns of A, and $R = A(I,:)$ is a matrix of rows of A, if we define $U = A(I,J)^{-1}$, then the corresponding skeleton decomposition is given by $A = CUR$, which is where the common alternative name "CUR decomposition" comes from. We remark that CUR decomposition is obviously a much more boring name. On the other hand, the term "skeleton decomposition" is not nearly as safe for Googling...

Define $\varphi : \mathbf{R}^r \to \operatorname{span}\{A(:, J_j) \mid j \in 1 : r\}$ by $\varphi(e_j) = A(:, J_j)$. Clearly, φ is linear and onto. By the linear independence of $\{A(:, J_j)\}$, φ maps an r-dimensional space onto an r-dimensional space, so φ must also be one-to-one. Thus, φ is invertible, with $\varphi^{-1}(A(:, J_j)) = e_j$ for $j \in 1 : r$.

Let $x \in \mathbf{R}^n$. Viewing A and $A(I, J)$ as linear mappings defined by matrix-vector multiplication, we have

$$\begin{aligned}
(A(I,J) \circ \varphi^{-1} \circ A)(x) = (A(I,J) \circ \varphi^{-1}) \left(\sum_{\ell=1}^{n} A(:,\ell) x_\ell \right) &= \sum_{\ell=1}^{n} x_\ell A(I,J) \varphi^{-1}(A(:,\ell)) && (20) \\
&= \sum_{\ell=1}^{n} x_\ell A(I,J) \varphi^{-1} \left(\sum_{j=1}^{r} \alpha_{\ell j} A(:, J_j) \right) && (21) \\
&= \sum_{\ell=1}^{n} x_\ell A(I,J) \sum_{j=1}^{r} \alpha_{\ell j} e_j = \sum_{\ell=1}^{n} x_\ell \sum_{j=1}^{r} \alpha_{\ell j} A(I, J_j) && (22) \\
&= \sum_{\ell=1}^{n} x_\ell \left(\sum_{j=1}^{r} \alpha_{\ell j} A(:, J_j) \right) (I, :) = \sum_{\ell=1}^{n} A(I,\ell) x_\ell && (23) \\
&= A(I, :)x. && (24)
\end{aligned}$$

Since x was arbitrary, and $A(I, J)$ and φ^{-1} are invertible, it follows that

$$A = \varphi \circ A(I,J)^{-1} \circ A(I, :) \tag{25}$$

as a linear map.

For any $x \in \mathbf{R}^n$, we can write $A(I, J)^{-1}A(I, :)x$ as a linear combination of $\{e_j\}_{j=1}^r$; that is, there exists $\{\beta_j\}$ such that

$$A(I,J)^{-1} A(I, :)x = \sum_{j=1}^{r} \beta_j e_j. \tag{26}$$

Then

$$Ax = \varphi \left(\sum_{j=1}^{r} \beta_j e_j \right) = \sum_{j=1}^{r} \beta_j A(:, J_j) = A(:, J) \sum_{j=1}^{r} \beta_j e_j = A(:, J) A(I,J)^{-1} A(I, :)x. \tag{27}$$

Since x was arbitrary, (17) follows. □

Psuedo-skeleton decomposition

The skeleton decomposition gives us a way to represent a rank-r matrix that only requires us to know that entries of the matrix in the rows I and columns J, which means $2nr$ entries, if the matrix is $n \times n$. This is that all-important property that we were looking for: avoiding the necessity to use all the entries of the matrix.

Unfortunately, we still need a way to find those $2nr$ entries efficiently. More importantly, though, skeleton decomposition requires us to know the rank of the matrix, and it doesn't allow us to approximate a high-rank matrix by a low-rank one.

Even so, by Theorem 2, a rank-r submatrix may still be used to represent the original matrix on the cross of the submatrix, even if the original matrix has higher than r. This leads to the natural question (and the title of our main reference [2]): can we choose a "good enough" submatrix so that the matrix S in Theorem 2 not only agrees with A on the cross, but also approximates A *off* of the cross?

To facilitate the discussion of this question, we need to introduce the formal concept of pseudo-skeleton decomposition, which is a generalization of the skeleton decomposition[3].

Definition 4. (Pseudo-skeleton decomposition) *Let $A \in \mathbf{R}^{m\times n}$ be a matrix, and let I and J be sequences of row indices of A of length r. If $G \in \mathbf{R}^{r\times r}$, then*

$$B = A(:, J)GA(I, :) \tag{28}$$

*is called a **pseudo-skeleton decomposition** of A with **core** G, **row indices** I, and **column indices** J.*

Now the question we need to answer, and the main focus of this analysis, is this: can we choose the core G and submatrix $A(I, J)$ so that the pseudo-skeleton approximation is nearly as good as the best rank-r approximation? At this point, we will fix $G = A(I, J)^{-1}$ (and assume that $A(I, J)$ is invertible) after the skeleton approximation. This will allow us to focus on simply finding a good enough submatrix $A(I, J)$. Moreover, retaining the form of the skeleton decomposition means that we will also avoid having to use all the entries A to find the pseudo-skeleton decomposition, just as we observed in the case of skeleton decomposition.

2.2 A Good Submatrix: Existence and Criteria

The first step in our analysis is to show that a pseudo-skeleton decomposition can be a good low-rank approximation. This analysis will lead naturally to the next step, which in turn will lead to the construction of the pseudo-skeleton decomposition algorithm.

The proof of the existence of a good psuedo-skeleton decomposition with our choice of $G = A(I, J)^{-1}$ is fairly technical, but I will try to give some motivation for the key ideas nonetheless.

Motivation

To quantify what we mean by a good pseudo-skeleton decomposition, we introduce a norm in which to measure the error.

Definition 5. (Chebyshev Norm) *If $A \in \mathbf{R}^{m\times n}$, define the **Chebyshev norm** of A by*

$$\|A\|_\infty = \max_{i,j} |A_{ij}|. \tag{29}$$

We want to show that there is a pseudo-skeleton decomposition such that the error in Chebyshev norm is not too much bigger than the error of the best low-rank approximation. In other words, we want to show that if $A(I, J)$ is a "good" rank-r submatrix, then

$$\left\|A - A(: J)A(I, J)^{-1}A(J, :)\right\|_\infty \lesssim \sigma_{r+1}, \tag{30}$$

[3] We remark that the psuedo-skeleton decomposition is not really a decomposition in the sense that it provides an exact factorization of the matrix; rather, it is a factorized *approximation* of the matrix. For this reason, some authors instead call it a "pseudo-skeleton approximation" or "pseudo-skeleton component" of the matrix [3, 9].

where σ_{r+1} is the $(r+1)$-th largest singular value of A, which is the same as the error of the best rank-r approximation of A in the spectral norm by Theorem 1.

The Chebyshev error is the maximum absolute difference of the elements of the matrices, which is a natural way to measure error. The spectral norm is natural as well, in its own way, but we will see that it is easier to control the error of a particular element of a pseudo-skeleton decomposition than it is to control the spectral norm error of the entire decomposition.

In any case, all matrix norms are equivalent, and the spectral norm can be optimally controlled by the Chebyshev norm as follows.

Lemma 2. *For any matrix $A \in \mathbf{R}^{m\times n}$,*

$$\|A\|_2 \leq \sqrt{mn}\,\|A\|_\infty \tag{31}$$

where $\|\cdot\|_2$ is the spectral norm. This inequality is sharp.

Proof. Let $x \in \mathbf{R}^n$. Overloading $\|\cdot\|_2$ to also mean the Euclidean vector norm in $\mathbf{R}^n$, the Cauchy-Schwarz inequality implies that

$$\|Ax\|_2 = \sqrt{\sum_{i=1}^m |Ax_i|^2} = \sqrt{\sum_{i=1}^m \left|\sum_{j=1}^n A_{ij}x_j\right|^2} \leq \sqrt{\sum_{i=1}^m \left(\sum_{j=1}^n |A_{ij}|^2\right)\left(\sum_{j=1}^n |x_j|^2\right)}. \tag{32}$$

By definition, $|A_{ij}|^2 \leq \|A\|_\infty^2$, so

$$\|Ax\|_2 \leq \sqrt{\sum_{i=1}^m n\,\|A\|_\infty^2\,\|x\|_2^2} = \sqrt{mn}\,\|A\|_\infty\,\|x\|_2\,. \tag{33}$$

If $\|x\|_2 \neq 0$, then

$$\frac{\|Ax\|_2}{\|x\|_2} \leq \sqrt{mn}\,\|A\|_\infty\,. \tag{34}$$

Taking the supremum over $x \neq 0$ on both sides completes the proof of the inequality.

For the sharpness, take $x \in \mathbf{R}^n$ such that $x_j = n^{-\frac{1}{2}}$ for $j \in 1:n$, so that $\|x\|_2 = 1$, and take $A \in \mathbf{R}^{m\times n}$ such that $A_{ij} = 1$ for all i, j. Then $\|A\|_\infty = 1$, and

$$\|Ax\|_2 = \sqrt{\sum_{i=1}^m \left|\sum_{j=1}^n n^{-\frac{1}{2}}\right|^2} = \sqrt{mn} = \sqrt{mn}\,\|A\|_\infty\,. \tag{35}$$

This implies that $\|A\|_2 \geq \sqrt{mn}\,\|A\|_\infty$, so the inequality is sharp. □

Thus, Lemma 2 and the naturalness of the Chebyshev norm together give a justification for using this norm aside from the after-the-fact observation that it is easier to prove something about the Chebyshev error.

How should we control the elements of the error in the pseudo-skeleton approximation, though? The first important link in the chain of inequalities that we need to construct to arrive at our goal (30) is the Cauchy interlacing theorem, and its important corollary for singular values.

Essentially, this theorem allows us to estimate the singular values of submatrix from the singular values of the larger matrix and vice versa. This is an important link because on the left side of (30) we have an expression involving a submatrix, and on the right side of (30) we have a singular value of the larger matrix.

Here is the interlacing theorem and its corollary for singular values.

Lemma 3. (Cauchy interlacing theorem) *Let $A \in \mathbf{R}^{n\times n}$ be a symmetric matrix with eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Let $A(I,I) \in \mathbf{R}^{r\times r}$ be a submatrix with the same row and column indices[4] I of A. If $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_r$ are the eigenvalues of $A(I,I)$, then*

$$\lambda_k \leq \mu_k \leq \lambda_{k+(n-r)}, \qquad k \in 1:r. \tag{36}$$

The condition in (36) is known as the ***interlacing of the eigenvalues****.*

Proof. See, for example, the proof of Parlett's Theorem 10.1.1 and the following Remark 10.1.1 [10]. □

Lemma 4. (Interlacing of singular values) *Let $A \in \mathbf{R}^{m\times n}$, and let $A(I,J) \in \mathbf{R}^{r\times r}$ be a submatrix of A with row indices I and column indices J. If $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\ell$ are the singular values of A, where $\ell = \min\{m,n\}$, and $\rho_1 \geq \rho_2 \geq \cdots \rho_r$ are the singular values of $A(I,J)$, then*

$$\sigma_k \geq \rho_k, \qquad k \in 1:r \tag{37}$$

$$\rho_k \geq \sigma_{k+m+n-2r}, \qquad k \in 1:(\max\{m,n\}-2r). \tag{38}$$

Proof. Define

$$M = \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix} \in \mathbf{R}^{(m+n)\times(m+n)}. \tag{39}$$

We note that $\lambda \neq 0$ is an eigenvalue of M if and only if

$$0 = \det(M - \lambda I_{(m+n)\times(m+n)}) = \det \begin{bmatrix} -\lambda I_{n\times n} & A^T \\ A & -\lambda I_{m\times m} \end{bmatrix} \tag{40}$$

$$= \det(-I_{n\times n}\lambda)\det(-I_{n\times n}\lambda + \lambda^{-1}A^TA) \tag{41}$$

by Lemma 5. The lattermost expression is 0 if and only if $\det(A^TA - \lambda^2 I_{n\times n}) = 0$, that is, if and only if λ is a singular value of A. Thus, the nonzero singular values of A and the nonzero eigenvalues of M are the same.

Define

$$N = \begin{bmatrix} 0 & A(I,J)^T \\ A(I,J) & 0 \end{bmatrix}. \tag{42}$$

By nearly identical reasoning to that above, we can show that the nonzero eigenvalues of N and the singular values of $A(I,J)$ are the same.

[4] Usually called a **principal submatrix**.

Noting that M is symmetric, and N is a submatrix of M with the row and column indices

$$I' = \{J_1, \dots, J_r, I_1, \dots, I_r\}, \qquad J' = \{J_1, \dots, J_r, I_1, \dots, I_r\} = I', \tag{43}$$

we can apply Lemma 3 to M and N. Let $\lambda_1 \le \lambda_2 \cdots \le \lambda_{m+n}$ be the eigenvalues of M, and let $\mu_1 \le \mu_2 \le \cdots \le \mu_{2r}$ be the eigenvalues of N. Accounting for the fact that the nonzero eigenvalues of N are $\rho_r \le \rho_{r-1} \le \cdots \le \rho_1$, and the nonzero eigenvalues of M are $\sigma_\ell \le \sigma_{\ell-1} \le \cdots \le \sigma_1$, we must have

$$\lambda_{m+n-k} = \sigma_{k+1}, \quad k \in 0:(\ell-1), \qquad \mu_{2r-k} = \rho_{k+1}, \quad k \in 0:(r-1), \tag{44}$$

and all the other eigenvalues of M and N are 0.

Then, by Lemma 3,

$$\begin{align}
& \lambda_k \le \mu_k \le \lambda_{k+(m+n-2r)}, \qquad k \in 1:2r \tag{45}\\
\Longrightarrow\ & \lambda_{2r-k} \le \mu_{2r-k} \le \lambda_{m+n-k}, \qquad k \in 0:(2r-1) \tag{46}\\
\Longrightarrow\ & \lambda_{m+n-(m+n-2r+k)} \le \rho_{k+1} \le \sigma_{k+1}, \qquad k \in 0:(r-1) \tag{47}\\
\Longrightarrow\ & \rho_k \le \sigma_k, \quad k \in 1:r, \qquad \sigma_{m+n-2r+k} \le \rho_k, \quad k \in 1:r \text{ and } m+n-2r+k \le \ell \tag{48}\\
\Longrightarrow\ & \rho_k \le \sigma_k, \quad k \in 1:r, \qquad \sigma_{m+n-2r+k} \le \rho_k, \quad 1 \le k \le 2r - \max\{m,n\}. \tag{49}
\end{align}$$

□

From Lemma 4, we see that the singular values of a submatrix are controlled by the corresponding singular values of the larger matrix. In order to make another link to the left side of (30), we need to connect the singular values of the submatrix to the magnitude of the elements, as this connects directly to the Chebyshev norm.

We have already seen one way to connect the spectral norm (which gives us the largest singular value by Theorem 1) and the Chebyshev norm, namely in Lemma 2. Applying Lemma 2 to the left side of (30) will give us an inequality in the wrong direction.

Indeed, very loosely speaking, by the interlacing of singular values, we expect to find something like

$$\rho_1 \lesssim \sigma_1, \tag{50}$$

but by applying Lemma 2 to the left side of (30) we expect to find something like

$$\rho_1 \lesssim \text{chebyshev norm.} \tag{51}$$

Obviously, we can't chain these inequalities as we would like to. Moreover, we only get the largest singular value σ_1, when what we want is σ_{r+1}. Thus, we use a simple trick. Taking the inverse on both sides of the last singular value interlacing inequality, we have

$$\sigma_{r+1}^{-1} \lesssim \rho_{r+1}^{-1}. \tag{52}$$

The appearance of ρ_{r+1}^{-1} is not too discouraging, actually, as ρ_{r+1}^{-1} is the *largest* singular value of the *inverse* of an $(r+1) \times (r+1)$ submatrix whose smallest singular value is ρ_{r+1}, which seems promising, considering that the inverse of a submatrix appears on the left side of our target inequality (30).

Unfortunately, the inverse of a submatrix that appears is the inverse of the $r \times r$ submatrix $A(I, J)$. In order to use our trick and relate σ_{r+1} to the left side of (30), we feel that we should extend $A(I, J)$ somehow by one row and one column to make it an $(r+1) \times (r+1)$ submatrix. Moreover, the extension needs to be invertible, so the only possible choice is to add a row and column from A that is not already present in $A(I, J)$.

This seems like a reasonable choice considering that we already know from Theorem 2 that the error on the cross of $A(I, J)$ is zero; adding a row and column from off of the cross will hopefully give us a relationship between the potentially nonzero, off-cross elements of the error matrix and σ_{r+1}.

Suppose we form the matrix $\widehat{A}$ by adding an off-cross row and column from A to $A(I, J)$. Ultimately, we want the elements of $\widehat{A}$ to be controlled by σ_{r+1}, but what we have is a lower bound on the largest singular value of $\widehat{A}^{-1}$ by σ_{r+1}^{-1}. This is where our trick pays off, as we can now chain this estimate with one from Lemma 2 to obtain a bound on the elements of $\widehat{A}^{-1}$:

$$\sigma_{r+1}^{-1} \lesssim \rho_{r+1}^{-1} = \left\| \widehat{A}^{-1} \right\|_2 \lesssim \left\| \widehat{A}^{-1} \right\|_\infty . \tag{53}$$

This is a tantalizing inequality: if we could pull the -1 power out of the Chebyshev norm, then we could simply invert on both sides and be done (as the row and column we added to $\widehat{A}$ were arbitrary).

Regrettably, matters are not this simple. As it turns out, though, making the above idea rigorous will lead us to a requirement on $A(I, J)$ from which we can build an efficient algorithm for finding a submatrix that satisfies our target inequality, so we will forge ahead.

There is an explicit relationship between the elements of an inverse matrix and the elements of the original matrix: Cramer's rule. According to Cramer's rule,

$$\left| \widehat{A}_{ij}^{-1} \right| = \left| \det(M) \det\left(\widehat{A}^{-1}\right) \right| , \tag{54}$$

where M is $\widehat{A}$ with one column replaced by e_j. Using expansion by cofactors on M, we see that $|\det(M)|$ is the absolute determinant of some $r \times r$ submatrix of A.

At this point, we can't do much about M, but we can compute $\det\left(\widehat{A}^{-1}\right) = \frac{1}{\det(\widehat{A})}$ explicitly using the block matrix determinant rule, as $\widehat{A}$ is a block matrix with $A(I, J)$ as a block. We recall this formula below.

Lemma 5. (Submatrix determinants) *If A and D are square matrices, then*

$$\det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) = \begin{cases} \det(A)\det(D - CA^{-1}B) & \text{if } A^{-1} \text{ exists,} \\ \det(D)\det(A - BD^{-1}C) & \text{if } D^{-1} \text{ exists.} \end{cases} \tag{55}$$

Proof. See, for example, the proof of (6.2.1) by Meyer [6]. □

Lemma 5 implies that

$$\det\left(\widehat{A}^{-1}\right) = \det\left(A(I, J)^{-1}\right)\gamma^{-1} \tag{56}$$

for some number γ depending on $A(I, J)$ and the row and column added to $A(I, J)$. Equivalently,

$$\det(A(I, J))\det\left(\widehat{A}^{-1}\right) = \gamma^{-1}. \tag{57}$$

The left side of this equation is very close to the right side of Cramer's ruler for $\widehat{A}_{ij}^{-1}$. At this point, there isn't much more we can do, but this similarity does give us some inspiriation as to what properties $A(I,J)$ needs to have to be good enough for us to reach our target inequality.

Indeed, by some stroke of fate, it turns out that γ is none other than the element of the error matrix at the intersection of the row and column that we added to $A(I,J)$. In other words, γ is an arbitrary nonzero element of the error matrix, precisely what we needed to control. The fact that γ is an element of the error matrix comes from the fact that $\widehat{A}$ is a block matrix, and there is an explicit inversion formula for block matrices, which is related to the determinant formula in Lemma 5.

Lemma 6. (Submatrix Inversion) *If A and D are square matrices, and A^{-1} exists, then the block matrix*

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{58}$$

is invertible if and only if $\Gamma = D - CA^{-1}B$ is invertible, and

$$X^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B\Gamma^{-1}CA^{-1} & -A^{-1}B\Gamma^{-1} \\ -\Gamma^{-1}CA^{-1} & \Gamma^{-1} \end{bmatrix}. \tag{59}$$

Proof. We can show that Γ is invertible implies that X is invertible by showing that the formula given for X^{-1} is a left inverse of X by direct computation, which, coincidentally, proves the correctness of the formula as well:

$$\begin{aligned}
&\begin{bmatrix} A^{-1} + A^{-1}B\Gamma^{-1}CA^{-1} & -A^{-1}B\Gamma^{-1} \\ -\Gamma^{-1}CA^{-1} & \Gamma^{-1} \end{bmatrix} \cdot \begin{bmatrix} A & B \\ C & D \end{bmatrix} \\
&\qquad = \begin{bmatrix} I + A^{-1}B\Gamma^{-1}C - A^{-1}B\Gamma^{-1}C & A^{-1}B + A^{-1}B\Gamma^{-1}CA^{-1}B - A^{-1}B\Gamma^{-1}D \\ -\Gamma^{-1}C + \Gamma^{-1}C & -\Gamma^{-1}CA^{-1}B + \Gamma^{-1}D \end{bmatrix} \\
&\qquad = \begin{bmatrix} I & A^{-1}B + A^{-1}B\Gamma^{-1}\left(CA^{-1}B - D\right) \\ 0 & \Gamma^{-1}\left(D - CA^{-1}B\right) \end{bmatrix} \\
&\qquad = \begin{bmatrix} I & A^{-1}B - A^{-1}B \\ 0 & I \end{bmatrix} = I.
\end{aligned} \tag{60}$$

If X is not invertible, then by Lemma 5 we have $0 = \det(X) = \det(A)\det(D - CA^{-1}B) = \det(A)\det(\Gamma)$, which implies that Γ is not invertible because $\det(A) \neq 0$. □

We can take at least some credit for this convenient coincidence; it is not entirely unexpected, considering the inequality (53). At any rate, thanks to this coincidence, if we can make sure that $|\det(M)| \lesssim |\det(A(I,J))|$, then we can chain all our inequalities together to obtain

$$\sigma_{r+1}^{-1} \lesssim |\gamma|^{-1}, \tag{61}$$

which implies our target inequality (30).

We recall that $|\det(M)|$ is the absolute value of the determinant of some $r \times r$ submtrix of A. Thus, the most straightforward way to ensure that $|\det(M)| \lesssim |\det(A(I,J))|$ is to choose $A(I,J)$ so that the absolute value of its determinant is maximal among all $r \times r$ submatrices of A. We note that this has the added perk of ensuring that $A(I,J)$ is nonsinguluar if the rank of A is at least[5] r.

[5] If the rank of A is less than r, then there is still a way to apply the algorithm anyway using QR decomposition [2].

The absolute value of the determinant of a matrix has a special geometric interpretation: it is the volume of the parallelipiped bounded by the column vectors of the matrix. Inspired by this, we define the **volume** of a square matrix to be the absolute value of its determinant.

Definition 6. (Volume) *Let $A \in \mathbf{R}^{r\times r}$ be a square matrix. Then the **volume** of A is defined to be*

$$\mathcal{V}(A) = |\det(A)|. \tag{62}$$

Now we can give a precise definition to the criterion that we think will guarantee that $A(I, J)$ is a good submatrix.

Definition 7. (Maximum volume submatrix) *Let $A \in \mathbf{R}^{m\times n}$. A submatrix $A_{\blacksquare} = A(I, J) \in \mathbf{R}^{r\times r}$ of A is a **rank-**r **maximum volume submatrix** of A if*

$$\mathcal{V}(A_{\blacksquare}) = \max\Big\{\mathcal{V}(A(I', J')) \mid A(I', J') \in \mathbf{R}^{r\times r} \text{ is a submatrix of } A\Big\}. \tag{63}$$

We will typically denote maximum volume submatrices of A by $A_{\blacksquare}$.

Rigorous proof

Now that we have some idea of what makes a good submatrix (maximal volume) as well as some idea of how to prove that it is a good submatrix, we go ahead and make rigorous the loose arguments from above.

Theorem 4. (Maximum volume pseudo-skeleton [7]) *Let $A \in \mathbf{R}^{m\times n}$ be a matrix with singular values $\{\sigma_i\}$ in nonascending order. If $A_{\blacksquare} = A(I, J)$ is a rank-r maximum volume submatrix of A, then*

$$\left\|A - A(:, J)A_{\blacksquare}^{-1}A(I, :)\right\|_\infty \leq (r + 1)\sigma_{r+1}, \tag{64}$$

where $\sigma_{\min\{m,n\}+1} = 0$ by convention, and $\mathscr{S}_{A_{\blacksquare}^{-1}}$ is the pseudo-skeleton decomposition of A using rows and columns I and J, with core $A_{\blacksquare}^{-1}$.

Proof. Set $\mathscr{S}_{A_{\blacksquare}^{-1}} = A(: J)A_{\blacksquare}^{-1}A(I, :)$, and define $E = A - \mathscr{S}_{A_{\blacksquare}^{-1}}$. By Theorem 2, $E_{ij} = 0$ if $i = I_{i'}$ for some i' or $j = J_{j'}$ for some j'. If i does not appear in the sequence I and j does not appear in the sequence J, then define $\gamma = E_{ij}$, so that

$$\begin{aligned}\gamma = A(i, j) - \mathscr{S}_{A_{\blacksquare}^{-1}}(i, j) &= A(i, j) - \sum_{k=1}^{r}\sum_{\ell=1}^{r} A(i, J_k)A_{\blacksquare}^{-1}(k, \ell)A(I_\ell, j) && (65)\\ &= A(i, j) - A(i, J)A_{\blacksquare}^{-1}A(I, j). && (66)\end{aligned}$$

If we can show that this arbitrary (potentially) nonzero element of E satisfies $|\gamma| \leq (r + 1)\sigma_{r+1}$, then $\|E\|_\infty \leq (r + 1)\sigma_{r+1}$, and the proof is complete.

Extend I to I' by setting $I'_{r+1} = i$, and extend J to J' by setting $J'_{r+1} = j$. Define the matrix $\widehat{A} = A(I', J')$. By construction, we have

$$\widehat{A} = \begin{bmatrix} A_{\blacksquare} & A(I, j) \\ A(i, J) & A(i, j) \end{bmatrix}. \tag{67}$$

We note that by Lemma 6, the matrix $\widehat{A}$ is invertible if and only if $\gamma = A(i,j) - A(i,J)A_{\blacksquare}^{-1}A(I,j)$ is invertible, that is, nonzero. If $\gamma = 0$, then certainly $|\gamma| \leq (r+1)\sigma_{r+1}$.

Suppose that $\gamma \neq 0$. Then $\widehat{A}$ is invertible, and by Lemma 6,

$$\widehat{A}^{-1} = \begin{bmatrix} A_{\blacksquare}^{-1} + A_{\blacksquare}^{-1}A(I,j)\gamma^{-1}A(i,J)A_{\blacksquare}^{-1} & -A_{\blacksquare}^{-1}A(I,j)\gamma^{-1} \\ -\gamma^{-1}A(i,J)A_{\blacksquare}^{-1} & \gamma^{-1} \end{bmatrix}. \tag{68}$$

Hence $\left\|\widehat{A}^{-1}\right\|_{\infty} \geq |\gamma^{-1}|$. On the other hand, by Lemma 1, for $\ell \in 1:(r+1)$, the column $\widehat{A}^{-1}(:,\ell)$ satisfies the equation

$$\widehat{A}\widehat{A}^{-1}(:,\ell) = I_{(r+1)\times(r+1)}(:,\ell) = e_{\ell}. \tag{69}$$

Let $k \in 1:(r+1)$. Since $\widehat{A}$ is invertible, Cramer's ruler implies that

$$\left|\widehat{A}^{-1}(k,\ell)\right| = \left|\frac{\det(M)}{\det\left(\widehat{A}\right)}\right|, \tag{70}$$

where M is the matrix with $M(:,k) = e_{\ell}$, and $M(:,k') = \widehat{A}(:,k')$ if $k' \neq \ell$. That is,

$$M = \begin{bmatrix} \widehat{A}(:,1) & \cdots & \widehat{A}(:,k-1) & e_{\ell} & \widehat{A}(:,k+1) & \cdots \widehat{A}(:,r+1) \end{bmatrix}. \tag{71}$$

Let $I'' = \{1,2,\dots k-1,k+1,\dots,r+1\}$. Expanding by cofactors on the kth column of M, we get $|\det(M)| = |\det(M')|$, where $M' = M(I'',I'')$. Since M coincides with $\widehat{A}$ on all but the kth column, it follows that $M' = M(I'',I'') = \widehat{A}(I'',I'') = A(I',J')(I'',I'')$. Hence, M' is an $r \times r$ submatrix of A.

By the maximality of the volume of $A_{\blacksquare}$, it follows that

$$\left|\widehat{A}^{-1}(k,\ell)\right| = \frac{|\det(M')|}{\left|\det\left(\widehat{A}\right)\right|} = |\det(M')| \cdot \left|\det\left(\widehat{A}^{-1}\right)\right| \leq |\det(A_{\blacksquare})| \cdot \left|\det\left(\widehat{A}^{-1}\right)\right|. \tag{72}$$

By Lemma 5,

$$\det\left(\widehat{A}\right) = \det(A_{\blacksquare})\left(A(i,j) - A(i,J)A_{\blacksquare}^{-1}A(I,j)\right) = \det(A_{\blacksquare})\gamma, \tag{73}$$

so

$$|\det(A_{\blacksquare})| \cdot \left|\det\left(\widehat{A}^{-1}\right)\right| = \left|\gamma^{-1}\right|. \tag{74}$$

Therefore, $\left|\widehat{A}^{-1}(k,\ell)\right| \leq |\gamma^{-1}|$. Since k and ℓ were arbitrary, it follows that $\left\|\widehat{A}^{-1}\right\|_{\infty} \leq |\gamma^{-1}|$. Thus, $\left\|\widehat{A}^{-1}\right\|_{\infty} = |\gamma^{-1}|$.

Recall that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{m,n\}}$ are the singular values of A, and let $\rho_1 \geq \rho_2 \geq \cdots \geq \rho_{r+1}$ be the singular values of $\widehat{A}$. By Lemma 4, we must have $\sigma_{r+1} \geq \rho_{r+1}$. Since $\{\rho_k^{-1}\}_{k=1}^{r+1}$ are the singular values of $\widehat{A}^{-1}$, it follows that ρ_{r+1}^{-1} is the largest singular value of $\widehat{A}^{-1}$; hence, by Lemma 2,

$$\sigma_{r+1}^{-1} \leq \rho_{r+1}^{-1} = \left\|\widehat{A}^{-1}\right\|_2 \leq (r+1)\left\|\widehat{A}^{-1}\right\|_{\infty} = (r+1)\left|\gamma^{-1}\right|. \tag{75}$$

It follows that

$$|\gamma| \leq (r+1)\sigma_{r+1}. \tag{76}$$

Since γ was an arbitrary nonzero element of E, we conclude that

$$\|E\|_\infty \leq (r+1)\sigma_{r+1}. \tag{77}$$

□

The reasoning in the motivation section ultimately required us to choose $A(I, J)$ so that $\mathcal{V}(M) \lesssim \mathcal{V}(A(I, J))$, but it didn't require a hard inequality. Therefore, it seems likely that we can extend Theorem 4 to provide an estimate on the error of the pseudo-skeleton decomposition when $A(I, J)$ merely has *nearly* maximal volume. Indeed, it is not difficult at all.

Theorem 5. (Quasi-maximum volume pseudo-skeleton [7]) *Let $A \in \mathbf{R}^{m\times n}$ be a matrix with singular values $\{\sigma_i\}$ in nonascending order. If $A_\blacksquare = A(I, J)$ is a rank-r maximum volume submatrix of A, and $B = A(I', J')$ is a rank-r submatrix of A that has **quasi-maximal volume in** A in the sense that there exists $\nu > 0$ such that*

$$\mathcal{V}(B) \geq \nu\mathcal{V}(A_\blacksquare), \tag{78}$$

then

$$\|A - \mathscr{S}_{B^{-1}}\|_\infty \leq \nu^{-1}(r+1)\sigma_{r+1}. \tag{79}$$

Proof. We proceed in a manner nearly the same as in the proof of Theorem 4. If we define $E = A - \mathscr{S}_{B^{-1}}$, then E is zero at row indices I' and column indices J' by Theorem 2. If we take an arbitrary nonzero entry $\gamma = E_{ij}$, then i and j do not occur in I' or J'.

Define

$$\widehat{B} = \begin{bmatrix} B & B(I,j) \\ B(i,J) & B(i,j) \end{bmatrix}. \tag{80}$$

By reasoning analogous to that used in the proof of Theorem 4, we can show that for any $k, \ell \in 1 : (r+1)$,

$$\left|\widehat{B}^{-1}(k,\ell)\right| = |\det(M')| \cdot \left|\det\left(\widehat{B}^{-1}\right)\right| \tag{81}$$

for some $r \times r$ submatrix M' of A. Applying the quasi-maximal volume property of B, we get

$$\left|\widehat{B}^{-1}(k,\ell)\right| \leq |\det(A_\blacksquare)| \cdot \left|\det\left(\widehat{B}^{-1}\right)\right| \leq \nu^{-1}|\det(B)| \cdot \left|\det\left(\widehat{B}^{-1}\right)\right|. \tag{82}$$

Applying Lemma 5 in the same way we did in Theorem 4, we get

$$\left|\gamma^{-1}\right| = |\det(B)| \cdot \left|\det\left(\widehat{B}^{-1}\right)\right|. \tag{83}$$

Therefore,

$$\left|\widehat{B}^{-1}(k,\ell)\right| \leq \nu^{-1}\left|\gamma^{-1}\right|. \tag{84}$$

This implies that $\left\|\widehat{B}^{-1}\right\|_\infty \leq \nu^{-1}\left|\gamma^{-1}\right|$, as k and ℓ were arbitrary. On the other hand, we can also obtain the estimate

$$\sigma_{r+1}^{-1} \leq (r+1)\left\|\widehat{B}^{-1}\right\|_\infty \tag{85}$$

by the same reasoning that was used in Theorem 4. Thus,

$$|\gamma| \leq \nu^{-1}(r+1)\sigma_{r+1}. \tag{86}$$

Since γ was an arbitrary nonzero element of E, it follows that $\|E\|_\infty \leq \nu^{-1}(r+1)\sigma_{r+1}$. □

2.3 Dominance

At this point, we have reduced the problem of finding a good submatrix for the pseudo-skeleton decomposition to the problem of finding a submatrix of quasi-maximal volume. The brute-force approach of computing the volume of every possible submatrix is even worse than just computing the SVD of the original matrix and not bothering with pseudo-skeleton decomposition at all, so we hope that there is an efficient way to find a quasi-maximum volume submatrix.

To this end, we simplify our problem. Instead of looking for an $r \times r$ submatrix of maximal volume in an $m \times n$ matrix, let us first consider the case of an $m \times r$ submatrix. If we had an efficient way to do this, then we could apply it to a random set of r columns of an $m \times n$ matrix, obtaining a corresponding set of rows so that the submatrix on those rows and columns has high volume. Then we could apply the method to the transpose of the r selected rows, obtaining a corresponding set of columns so that the submatrix on those columns has high volume. Then we might hope that iterating this process a few times would lead to the identification of a high-volume submatrix.

As we know from Theorem 5, we only need to find a submatrix of quasi-maximum volume. To this end, we need a bound on the determinant of a matrix in order to prove that our algorithm achieves quasi-maximum volume. The easiest such bound is Hadamard's inequality, which states that the volume of a matrix, that is, the volume of the paralleipiped bounded by the column vectors of the matrix, is no greater than the volume of the rectangular prism bounded by vectors of the same lengths.

Lemma 7. (Hadamard's Inequality) *Let $A \in \mathbf{R}^{m\times m}$. Then*

$$\mathcal{V}(A) \leq \prod_{j=1}^{m} \|A(:,j)\|_2\,, \tag{87}$$

where $\|\cdot\|_2$ is the Euclidean vector norm.

Proof. See Example 6.1.4 in Meyer's linear algebra textbook [6]. □

Recall that B has quasi-maximal volume in A if $\mathcal{V}(B) \geq \nu\mathcal{V}(A_\blacksquare)$ for some $\nu > 0$, where $A_\blacksquare$ has maximal volume in B. This is equivalent to

$$\mathcal{V}\left(A_\blacksquare B^{-1}\right) \leq \nu^{-1}. \tag{88}$$

We can obtain this inequality via Hadamard's inequality if

$$\nu^{-1} = \prod_{j=1}^{r} \left\|\left(A_\blacksquare B^{-1}\right)(:,j)\right\|_2\,. \tag{89}$$

In other words, we want to choose the submatrix $B = A(I,J)$ such that the right side of the above equation is pretty small. The minimum, of course, is to choose $B^{-1} = A_\blacksquare^{-1}$, in which case $\nu = 1$. How can we ensure that ν^{-1} is not too big without too much effort?

Perhaps the simplest way is to require that every entry of $A_\blacksquare B^{-1}$ is bounded, say by 1. In this case, we would have

$$\nu^{-1} \leq \prod_{j=1}^{r} r^{\frac{1}{2}} = r^{\frac{r}{2}}. \tag{90}$$

We don't know what $A_\blacksquare$ is, but if $A_\blacksquare = A(I, J)$, then by Lemma 1,

$$\left(A_\blacksquare B^{-1}\right)(i, j) = A(i, :)B^{-1}(:, j). \tag{91}$$

Thus, if every element of AB^{-1} is bounded by 1, then every element of $A_\blacksquare B^{-1}$ is, as well. Checking that every element of AB^{-1} is small is not too hard, thanks to our assumption that A is only $m \times r$. Indeed, since r is small, we can afford a linear search through the nr entries of the $n \times r$ matrix AB^{-1}. This leads us to define the notion of a **dominant submatrix**.

Definition 8. (Dominant submatrix of a tall matrix) *Let $A \in \mathbf{R}^{m \times r}$ have rank r (which means that $m \geq r$). A nonsingular, square submatrix $A_\square = A(I, :) \in \mathbf{R}^{r \times r}$ of A is a **dominant submatrix** of A if*

$$\left\|AA_\square^{-1}\right\|_\infty \leq 1. \tag{92}$$

We will typically denote dominant submatrices of A by $A_\square$.

Following our reasoning above, we can show that dominant matrices are quasi-maximum volume matrices. First, we generalize our application of Lemma 1, and, second, we prove the quasi-maximality of dominant submatrices.

Lemma 8. *Let $A \in \mathbf{R}^{m \times r}$, and let $B \in \mathbf{R}^{r \times r}$ be nonsingular. If $A(I, :), A(I', :) \in \mathbf{R}^{r \times r}$ are square submatrices of A, and $A(I', :)$ is nonsingular, then $(AB)(I', :)$ is nonsingular, and*

$$\frac{\mathcal{V}(A(I, :))}{\mathcal{V}(A(I', :))} = \frac{\mathcal{V}((AB)(I, :))}{\mathcal{V}((AB)(I', :))}. \tag{93}$$

Proof. By Lemma 1, $(AB)(I, :) = A(I, :)B$, and $(AB)(I', :) = A(I', :)B$. Thus,

$$\det((AB)(I', :)) = \det(A(I', :)B) = \det(A(I', :))\det(B) \neq 0. \tag{94}$$

Similarly, $\det((AB)(I, :)) = \det(A(I, :))\det(B)$. Therefore,

$$\frac{\det((AB)(I, :))}{\det((AB)(I', :))} = \frac{\det(A(I, :))\det(B)}{\det(A(I', :))\det(B)} = \frac{\det(A(I, :))}{\det(A(I', :))}. \tag{95}$$

Taking the absolute value on both sides gives (93). □

Theorem 6. (Approximation by dominant submatrix [2]) *Let $A \in \mathbf{R}^{m \times r}$ have rank r, and let $A_\blacksquare$ be a maximum volume submatrix of A. Then*

$$\mathcal{V}(A_\square) \geq r^{-\frac{r}{2}}\mathcal{V}(A_\blacksquare) \tag{96}$$

for all dominant submatrices $A_\square$ of A. The inequality is sharp.

Proof. Let $A_\square$ be a dominant submatrix of A, and let $B = AA_\square^{-1}$. By definition, $\|B\|_\infty \leq 1$. Thus, if we take r rows of B at indices I, then $\|B(I,:)\|_\infty \leq 1$ as well, which implies that $\|B(I,j)\|_2 \leq \sqrt{r}$. By Hadamard's inequality (Lemma 7), then,

$$\mathcal{V}(B(I,:)) \leq \prod_{j=1}^{r} \|B(I,j)\|_2 \leq r^{\frac{r}{2}}, \tag{97}$$

with equality holding if $\{B(I,j)\}_{j=1}^{r}$ forms an orthogonal set.

In particular, choose I such that $A_\blacksquare = A(I,:)$. By Lemma 1, we have

$$r^{\frac{r}{2}} \geq \mathcal{V}(B(I,:)) = |\det(B(I,:))| = |\det(A(I,:))\det(A_\square^{-1})| = \frac{\mathcal{V}(A_\blacksquare)}{\mathcal{V}(A_\square)}. \tag{98}$$

Then (96) follows.

If we choose $A = (1,1)^T$, then the maximum volume submatrix of A is $A_\blacksquare = [1]$, with volume 1. If we set $A_\square = A_\blacksquare$ and note that $r = 1$ for this choice[6] of A, we see that $\mathcal{V}(A_\square) = 1 = r^{-\frac{r}{2}}\mathcal{V}(A_\blacksquare)$ □

Since dominant submatrices are quasi-maximal in volume, we focus our attention on searching for a dominant submatrix instead of searching for some arbitrary quasi-maximum volume submatrix. This decision is reinforced by the fact that maximum volume submatrices are dominant, meaning that if we look for a dominant submatrix, we may well find a maximum volume submatrix. We present a detailed proof of this fact, as it will provide the key insight that we need to construct a fast algorithm for finding dominant submatrices.

Theorem 7. (Maximum volume submatrices are dominant [2]) *Let $A \in \mathbf{R}^{m\times r}$ have rank r, and let $A_\blacksquare \in \mathbf{R}^{r\times r}$ be a maximum volume submatrix of A. Then $A_\blacksquare$ is a dominant submatrix of A.*

Proof. Since the rank of A is r, there must be a set of r linearly independent rows of A, say at indices I'. Then $A(I',:)$ is nonsingular, and $\mathcal{V}(A(I',:)) > 0$. This implies that $\mathcal{V}(A_\blacksquare) \geq \mathcal{V}(A(I',:)) > 0$.

Since $\mathcal{V}(A_\blacksquare) > 0$, it follows that $A_\blacksquare$ is invertible. Define $B = AA_\blacksquare^{-1}$. There is some row index sequence I such that $A_\blacksquare = A(I,:)$. By Lemma 8, $A_\blacksquare$ has maximal volume in A if and only if $B(I,:)$ has maximal volume in B, as multiplication by the invertible matrix $A_\blacksquare^{-1}$ preserves the ratios of $r \times r$ submatrix volumes.

Furthermore, $B(I,:)$ is the identity matrix $I_{r\times r}$ because, by Lemma 1,

$$B(I,:) = \left(AA_\blacksquare^{-1}\right)(I,:) = A(I,:)A_\blacksquare^{-1} = A_\blacksquare A_\blacksquare^{-1} = I_{r\times r}. \tag{99}$$

Thus, $B(I,:)$ is dominant in B if and only if $\left\|BB(I,:)^{-1}\right\|_\infty = \|B\|_\infty = \left\|AA_\blacksquare^{-1}\right\|_\infty \leq 1$, that is, if and only if $A_\blacksquare$ is dominant in A.

We now prove the claim by contradiction. Suppose that $A_\blacksquare$ is not dominant in A. Then $B(I,:)$ is not dominant in B; that is, there exists $k \in 1:m$ and $j \in 1:r$ such that $|B_{kj}| > 1$.

[6] Sharp examples can be given for larger values of r. For instance, if r is a multiple of 4, and B contains an $r \times r$ Hadamard matrix, then equality holds as a result of the sharpness of Hadamard's inequality for Hadamard matrices.

Let $I'_i = I_i$ if $i \neq j$, and let $I'_j = k$. Then every row of $B(I',:)$ is a row of $I_{r\times r}$ except for the jth row, which is the kth row of B. That is,

$$B(I',:) = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ & & B(k,:)\ (j\text{th row}) & & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}. \tag{100}$$

Expanding by cofactors and expanding on the jth row of $B(I',:)$ last shows that

$$|\det(B(I',:))| = |B_{kj}| > 1. \tag{101}$$

This means that $\mathcal{V}(B(I',:)) > 1 = \mathcal{V}(B(I,:))$, so $B(I,:)$ is not maximal in B. Then $A_{\blacksquare}$ is not maximal in A, which is a contradiction.

Hence, $A_{\blacksquare}$ is dominant in A. □

`maxvol` preview

The method of proof-by-contradiction employed in Theorem 7 shows us that if a submatrix $A(I,:)$ of a tall matrix A is *not* dominant in A, then we can construct a submatrix of A that has greater volume than $A(I,:)$ by simplying swapping one row. Moreover, we see that the row we need to swap is any one corresponding to an element B_{kj} of $B = AA(I,:)^{-1}$ such that $|B_{kj}| > 1$. Even better, we see that the greatest possible increase in volume that can be achieved by changing one row of $A(I,:)$ is to swap with the row corresponding to the maximal element of B.

This method for strictly increasing the volume of a submatrix leads naturally to an iterative algorithm for finding a dominant submatrix. Simply swap one row of the current submatrix $A(I,:)$ with the row corresponding to the maximal element of $B = AA(I,:)^{-1}$, update B, and repeat until the elements of B are all no greater than 1. If $\|B\|_\infty \leq 1$, then $A(I,:)$ is dominant, and we are done.

3 The `maxvol` Algorithm

Going even faster

We recall that the choice of 1 in $\left\|AA(I,:)^{-1}\right\|_\infty \leq 1$, the dominance criterion for $A(I,:)$, was somewhat arbitrary. Indeed, if we increased this by, say, $\delta > 0$, then we can easily prove a volume estimate not much worse than the one in Theorem 6 while changing our stopping criterion to $\left\|AA(I,:)^{-1}\right\|_\infty \leq 1 + \delta$, which is looser and may significantly decrease the number of iterations without much change in performance. Adding this small buffer will also help the algorithm be more robust against floating-point round-off error.

To this end, we generalize the definition of dominance as follows.

Definition 9. (δ-dominant submatrix) *Let $A \in \mathbf{R}^{m\times r}$ have rank r (which means that $m \geq r$), and let $\delta > 0$. A nonsingular, square submatrix $A_{\square} = A(I,:) \in \mathbf{R}^{r\times r}$ of A is a **δ-dominant submatrix** of A if*

$$\left\|AA_{\square}^{-1}\right\|_\infty \leq 1 + \delta. \tag{102}$$

As we claimed, a δ-dominant submatrix has almost as much volume as a dominant submatrix, and also has quasi-maximal volume, as a result. The proof is nearly the same as the proof of Theorem 6.

Theorem 8. (Approximation by δ-dominant submatrix) *Let $A \in \mathbf{R}^{m\times r}$ have rank r, and let $A_{\blacksquare}$ be a maximum volume submatrix of A. Then*

$$V(A_{\square,\delta}) \geq ((1+\delta)r)^{-\frac{r}{2}}\mathcal{V}(A_{\blacksquare}) \tag{103}$$

for all δ-dominant submatrices $A_{\square,\delta}$ of A.

Proof. Let $A_{\square,\delta}$ be a dominant submatrix of A, and let $B = AA_{\square,\delta}^{-1}$. By definition, $\|B\|_\infty \leq 1+\delta$. Thus, if we take r rows of B at indices I, then $\|B(I,:)\|_\infty \leq 1+\delta$ as well, which implies that $\|B(I,j)\|_2 \leq \sqrt{(1+\delta)r}$. By Hadamard's inequality (Lemma 7), then,

$$\mathcal{V}(B(I,:)) \leq \prod_{j=1}^{r}\|B(I,j)\|_2 \leq ((1+\delta)r)^{\frac{r}{2}}. \tag{104}$$

In particular, choose I such that $A_{\blacksquare} = A(I,:)$. By Lemma 1, we have

$$((1+\delta)r)^{\frac{r}{2}} \geq \mathcal{V}(B(I,:)) = |\det(B(I,:))| = |\det(A(I,:))\det(A_{\square}^{-1})| = \frac{\mathcal{V}(A_{\blacksquare})}{\mathcal{V}(A_{\square})}. \tag{105}$$

Then (103) follows. □

Correctness and complexity

With the tolerance δ added, we are ready to formalize the algorithm that we have been building up to for so long, which is called `maxvol` because its purpose is to find a δ-dominant submatrix, which, as we saw, is quasi-maximal in volume. The steps in the algorithm are collected formally in Algorithm 1.

Algorithm 1: `maxvol`

```
Input: A matrix $A \in \mathbf{R}^{n\times r}$ of rank $r$
Input: Tolerance $\delta \geq 0$
Output: A matrix $A_{\square} \in \mathbf{R}^{r\times r}$ that is $\delta$-dominant in $A$
Initialize a nonsingular submatrix $A_{\boxdot}$ of $A$;
repeat
    $B \leftarrow AA_{\boxdot}^{-1}$;
    $i, j \leftarrow \operatorname{argmax}_{i',j'} |B_{i'j'}|$;
    if $|B_{ij}| > 1+\delta$ then
        $A_{\boxdot}(j,:) \leftarrow A(i,:)$;
    end
until $|B_{ij}| \leq 1+\delta$;
$A_{\square} \leftarrow A_{\boxdot}$;
```

Having a formal description of the algorithm, we are ready to prove that it will actually work; that is, the output of `maxvol` is, indeed, a δ-dominant submatrix of the input, and the number of steps required to find the δ-dominant submatrix is finite.

Theorem 9. (Correctness of `maxvol` [2]) *Let $A_{\boxdot}^{(k)}$ be the matrix $A_{\boxdot}$ in the `maxvol` algorithm after k steps of the loop. Then*

1. *$A_{\boxdot}^{(k)}$ is invertible,*
2. *the sequence of volumes $\left\{\mathcal{V}\left(A_{\boxdot}^{(k)}\right)\right\}$ is strictly increasing,*
3. *the `maxvol` algorithm terminates in a finite number of steps c,*
4. *the output $A_{\square}$ is δ-dominant in A,*
5. *if $\delta > 0$, then the number of steps c before the algorithm terminates is bounded by*

$$c \leq \frac{\log(\mathcal{V}(A_{\square})) - \log\left(\mathcal{V}\left(A_{\boxdot}^{(0)}\right)\right)}{\log(1+\delta)} \leq \frac{\log(\mathcal{V}(A_{\blacksquare})) - \log\left(\mathcal{V}\left(A_{\boxdot}^{(0)}\right)\right)}{\log(1+\delta)}. \tag{106}$$

Proof. The first matrix $A_{\boxdot}^{(0)}$ is invertible by construction (the initialization of $A_{\boxdot}^{(0)}$ as an invertible submatrix can always be done because the rank of A is r). Suppose for induction that $A_{\boxdot}^{(k)}$ is invertible for some $k \geq 0$. If $k = c$, then we are done. Otherwise, if $(i,j) = \underset{i',j'}{\operatorname{argmax}} |B_{i'j'}|$, where $B = A\left(A_{\boxdot}^{(k)}\right)^{-1}$, then $|B_{ij}| \geq 1 + \delta$.

Let $I^{(k)}$ be the row indices in A of the submatrix $A_{\boxdot}^{(k)}$, and let $I^{(k+1)}$ be the row indices in A of $A_{\boxdot}^{(k+1)}$. Then, by line 6 of Algorithm 1, $I_{\ell}^{(k+1)} = I_{\ell}^{(k)}$ if $\ell \neq j$, and $I_{j}^{(k+1)} = i$. By Lemma 1,

$$B\left(I^{(k)}, :\right) = A\left(I^{(k)}, :\right)\left(A_{\boxdot}^{(k)}\right)^{-1} = A_{\boxdot}^{(k)}\left(A_{\boxdot}^{(k)}\right)^{-1} = I_{r\times r}, \tag{107}$$

and

$$B\left(I^{(k+1)}, :\right) = A\left(I^{(k+1)}, :\right)\left(A_{\boxdot}^{(k)}\right)^{-1} = A_{\boxdot}^{(k+1)}\left(A_{\boxdot}^{(k)}\right)^{-1}. \tag{108}$$

On the other hand, since $I^{(k+1)}$ differs from $I^{(k)}$ only in the jth entry, we must have

$$B\left(I^{(k+1)}, :\right) = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ & & B(i,:)\ (j\text{th row}) & & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}. \tag{109}$$

Expanding by cofactors and expanding on the jth row last, we obtain

$$\frac{\mathcal{V}\left(A_{\boxdot}^{(k+1)}\right)}{\mathcal{V}\left(A_{\boxdot}^{(k)}\right)} = \left|\det\left(A_{\boxdot}^{(k+1)}\left(A_{\boxdot}^{(k)}\right)^{-1}\right)\right| = \left|\det\left(B\left(I^{(k+1)}, :\right)\right)\right| = |B_{ij}| > 1 + \delta. \tag{110}$$

This implies that

$$\mathcal{V}\left(A_{\boxdot}^{(k+1)}\right) > (1+\delta)\mathcal{V}\left(A_{\boxdot}^{(k)}\right) > 0, \tag{111}$$

which shows that $A_{\boxdot}^{(k+1)}$ is invertible. By induction, $A_{\boxdot}^{(k)}$ is invertible for all k. Moreover, (111) also shows that $\left\{\mathcal{V}\left(A_{\boxdot}^{(k)}\right)\right\}$ is strictly increasing.

Since the volume of $A_{\boxdot}^{(k)}$ increases with k, each $A_{\boxdot}^{(k)}$ is distinct. There are only finitely many submatrices of A, and at least one is δ-dominant (one of the maximum volume submatrices certainly is). Since each $A_{\boxdot}^{(k)}$ is distinct, $A_{\boxdot}^{(k)}$ must eventually be δ-dominant for some k. The stopping criterion on line 8 of Algorithm 1 is satisfied if and only if $A_{\boxdot}^{(k)}$ is δ-dominant, so the algorithm terminates in finitely many steps c, and the output is δ-dominant.

Iterating the inequality in (111), we obtain

$$\mathcal{V}\left(A_{\boxdot}^{(c)}\right) \geq (1+\delta)^{c}\mathcal{V}\left(A_{\boxdot}^{(0)}\right), \tag{112}$$

which implies the first inequality in (106) because $A_{\square} = A_{\boxdot}^{(c)}$. The second inequality is trivially true by the maximality of $\mathcal{V}\left(A_{\blacksquare}\right)$. □

We observe that the main computational cost of the `maxvol` algorithm is likely in determining the new B matrix on each step. The search for a maximal element, the conditional branching, and the row swapping in A are trivial by comparison.

Fortunately, B only changes by one row being changed in the inverse of $A_{\boxdot}$, and there is an efficient formula for rank-one updates of inverse matrices, called the **Sherman-Morrison formula**.

Lemma 9. (Sherman-Morrison formula) *Suppose that* $A \in \mathbf{R}^{n\times n}$ *is invertible, and let* $u, v \in \mathbf{R}^{n\times 1}$ *be nonzero column vectors. Then* $A + uv^T$ *is invertible if and only if* $1 + v^T A^{-1} u \neq 0$*, and*

$$\left(A + uv^T\right)^{-1} = A^{-1} - \frac{\left(A^{-1}u\right)\left(v^T A^{-1}\right)}{1 + v^T A^{-1} u}. \tag{113}$$

Proof. See Section 2 in an old paper by Bartlett [1]. □

With this efficient method for updating $B = AA_{\boxdot}^{-1}$, we can fill in the details of `maxvol` and analyze the computational complexity of the algorithm.

Theorem 10. (Complexity of `maxvol` [2]) *Let* $B^{(k)}$ *be the matrix* B *and let* $A_{\boxdot}^{(k)}$ *be the matrix* $A_{\boxdot}$ *in the* `maxvol` *algorithm after* k *steps. Then* $B^{(k+1)}$ *and* $A_{\boxdot}^{(k+1)}$ *can be computed in* $\mathcal{O}(nr)$ *operations from* $B^{(k)}$ *and* $A_{\boxdot}^{(k)}$*. Therefore, the overall cost of the iterative portion of* `maxvol` *is* $\mathcal{O}(cnr)$*, where* c *is the number of iteration steps.*

Proof. We can write $A_{\boxdot}^{(k+1)}$ in terms of $A_{\boxdot}^{(k)}$ as

$$A_{\boxdot}^{(k+1)} = A_{\boxdot}^{(k)} + e_j\left(A(i,:) - A_{\boxdot}^{(k)}(j,:)\right), \tag{114}$$

where e_j is the jth standard basis vector as a column vector. If we define $q = e_j$, and if we define $v = \left(A(i,:) - A_{\boxdot}^{(k)}(j,:)\right)^T$, then

$$A_{\boxdot}^{(k+1)} = A_{\boxdot}^{(k)} + qv^T. \tag{115}$$

By the Sherman-Morrison formula (Lemma 9),

$$\left(A_{\boxdot}^{(k+1)}\right)^{-1} = \left(A_{\boxdot}^{(k)}\right)^{-1} - \frac{\left(A_{\boxdot}^{(k)}\right)^{-1} q v^T \left(A_{\boxdot}^{(k)}\right)^{-1}}{1 + v^T \left(A_{\boxdot}^{(k)}\right)^{-1} q}. \tag{116}$$

By Lemma 1,

$$\begin{aligned} v^T \left(A_{\boxdot}^{(k)}\right)^{-1} &= \left(A(i,:) - A_{\boxdot}^{(k)}(j,:)\right) \left(A_{\boxdot}^{(k)}\right)^{-1} && (117)\\ &= A(i,:) \left(A_{\boxdot}^{(k)}\right)^{-1} - e_j^T && (118)\\ &= B^{(k)}(i,:) - e_j^T. && (119) \end{aligned}$$

Therefore, $v^T \left(A_{\boxdot}^{(k)}\right)^{-1} q = B^{(k)}(i,:)e_j - e_j^T e_j = B_{ij}^{(k)} - 1$. Multiplying both sides of (116) by A gives

$$\begin{aligned} B^{(k+1)} &= B^{(k)} - \frac{1}{B_{ij}^{(k)}} B^{(k)} e_j v^T \left(A_{\boxdot}^{(k)}\right)^{-1} && (120)\\ &= B^{(k)} - \frac{1}{B_{ij}^{(k)}} B^{(k)}(:,j) \left(B^{(k)}(i,:) - e_j^T\right). && (121) \end{aligned}$$

The update rule for $B^{(k)}$ given in (121) involves a $n \times 1$ by $1 \times r$ matrix multiplication, scalar and $n \times r$ matrix multplication, and $n \times r$ matrix subtraction. Hence, it requires $\mathcal{O}(nr)$ operations to complete. The update rule for $A_{\boxdot}^{(k)}$ is $\mathcal{O}(1)$ because we only need to keep track of which rows of A are in $A_{\boxdot}^{(k)}$, and on each step only one row changes.

□

4 Implementation in NumPy

4.1 Practical update rules

Most of `maxvol` is trivial to implement in `NumPy`. The trickiest part is the efficient updating of B and $A_{\boxdot}$ (lines 3 and 6 in Algorithm 1). Let $B^{(k)}$ be the matrix B in `maxvol` after k steps, and let $I^{(k)}$ be the row indices in A of $A_{\boxdot}$ after k steps. Let $J^{(k)}$ be the other row indices of A that do not occur in the sequence $I^{(k)}$.

Updating $A_{\boxdot}$

To update $A_{\boxdot}$, we only need to update $I^{(k)}$. Let i, j be the indices obtained on line 4 of Algorithm 1. The update for $A_{\boxdot}$ is that the jth row of $A_{\boxdot}$ becomes the ith row of A, so

$$I_{\ell}^{(k+1)} = \begin{cases} I_{\ell}^{(k)} & \ell \neq j \\ i & \ell = j. \end{cases} \tag{122}$$

If we reuse the memory for $I^{(k)}$ for $I^{(k+1)}$, this means only doing one update operation.

Using Z instead of B

We know from our analysis that on each step $B^{(k)}\left(I^{(k)},:\right) = I_{r\times r}$ (recall (107)). Therefore, it would be more efficient to store only the rows of $B^{(k)}$ at indices $J^{(k)}$. Let $Z^{(k)} = B^{(k)}\left(J^{(k)},:\right)$ denote the matrix of these rows.

Let $Z^{(k)}_{i'j'}$ be the maximum modulus element of $Z^{(k)}$. If we used $Z^{(k)}_{i'j'}$ in place of $B^{(k)}_{ij}$ in Algorithm 1, then the result of the algorithm would be unchanged. Indeed, on the kth loop iteration, there are two possibilities.

1. $Z^{(k)}_{i'j'} = B^{(k)}_{ij}$, in which case we may take $j = j'$ and $i = J^{(k)}_{i'}$. The rest of the loop iteration proceeds as it would using $B^{(k)}_{ij}$.
2. $Z^{(k)}_{i'j'}$ is not the maximum modulus element in $B^{(k)}$. In this case, $\left|Z^{(k)}_{i'j'}\right| \le \left|B^{(k)}_{ij}\right| = 1 \le 1+\delta$, so whether we use $Z^{(k)}_{i'j'}$ or $B^{(k)}_{ij}$, the loop should exit immediately.

In any case, then, using $Z^{(k)}_{i'j'}$ in place of $B^{(k)}_{ij}$ has no effect on the result of the algorithm.

Updating Z

Now updating $B^{(k)}$ amounts to updating $Z^{(k)}$. Recall the efficient, rank-1 update rule for B:

$$B^{(k+1)} = B^{(k)} - \frac{1}{B^{(k)}_{ij}} B^{(k)}(:,j)\left(B^{(k)}(i,:) - e_j^T\right), \tag{123}$$

where i, j are the indices obtained on line 4 of Algorithm 1. Taking the submatrix with row indices $J^{(k+1)}$ on both sides of (123), applying Lemma 1, and using $Z^{(k)}_{i'j'}$ in place of $B^{(k)}_{ij}$ as discussed above, we get

$$Z^{(k+1)} = B^{(k)}\left(J^{(k+1)},:\right) - \frac{1}{Z^{(k)}_{i'j'}} B^{(k)}\left(J^{(k+1)},j\right)\left(B^{(k)}(i,:) - e_j^T\right). \tag{124}$$

Evidently, we will also need to keep track of $J^{(k)}$ for all k in order to find $Z^{(k+1)}$. To update $J^{(k)}$, we need to ensure that $J^{(k+1)}$ contains all indices not in $I^{(k+1)}$. Only one index in $I^{(k+1)}$ is different from $I^{(k)}$; namely, $I^{(k+1)}_j = i$ instead of $I^{(k)}_j$. Thus, we need to remove i from $J^{(k+1)}$ and replace it with $I^{(k)}_j$. Let i' be the index such that $J^{(k)}_{i'} = i$. Then we can obtain $J^{(k+1)}$ by the rule

$$J^{(k+1)}_\ell = \begin{cases} J^{(k)}_\ell & \ell \ne i', \\ I^{(k)}_j & \ell = i'. \end{cases} \tag{125}$$

Like the update rule for $I^{(k)}$, this also only requires one operation if we reuse the memory for $J^{(k)}$ for $J^{(k+1)}$.

With this rule in place, we can relate $Z^{(k+1)}$ to $Z^{(k)}$ using (124). Let $L = \{1, 2, \ldots, i'-1, i'+1, \ldots, n-r\}$. Then $J^{(k+1)}_{L_\ell} = J^{(k)}_{L_\ell}$ for all ℓ. Therefore,

$$\left(B^{(k)}\left(J^{(k+1)},:\right)\right)(L,:) = \left(B^{(k)}\left(J^{(k)},:\right)\right)(L,:) = Z^{(k)}(L,:). \tag{126}$$

Taking the submatrix with row indices L on both sides of (124), we obtain

$$Z^{(k+1)}(L,:) = Z^{(k)}(L,:) - \frac{1}{Z^{(k)}_{i'j'}} Z^{(k)}(L,j) \left(B^{(k)}(i,:) - e_j^T \right). \tag{127}$$

Recalling that $j = j'$ and $i = J^{(k)}_{i'}$, we get

$$Z^{(k+1)}(L,:) = Z^{(k)}(L,:) - \frac{1}{Z^{(k)}_{i'j}} Z^{(k)}(L,j) \left(Z^{(k)}(i',:) - e_j^T \right). \tag{128}$$

Similarly, if we take the submatrix with row indices $\{i'\}$ on both sides of (124), then, by the definition of $J^{(k+1)}$, we get

$$\begin{aligned} Z^{(k+1)}(i',:) &= B^{(k)}\left(I^{(k)}_j,:\right) - \frac{1}{Z^{(k)}_{i'j}} B^{(k)}\left(I^{(k)}_j, j\right) \left(B^{(k)}(i,:) - e_j^T \right) & (129) \\ &= e_j^T - \frac{1}{Z^{(k)}_{i'j}} \left(Z^{(k)}(i',:) - e_j^T \right) & (130) \end{aligned}$$

because $B^{(k)}\left(I^{(k)}_j,:\right) = \left(B^{(k)}\left(I^{(k)},:\right)\right)(j,:) = I_{r \times r}(j,:) = e_j^T$.

Let D be a matrix with $D(L,:) = Z^{(k)}(L,:)$ and $D(i',:) = e_j^T$. Then, using D, we can incorporate the update rule (130) into (128):

$$Z^{(k+1)} = D - \frac{1}{Z^{(k)}_{i'j}} D(:,j) \left(Z^{(k)}(i',:) - e_j^T \right). \tag{131}$$

Thus, we can compute $Z^{(k+1)}$ from $Z^{(k)}$ using this update rule.

4.2 Step-by-step design in NumPy

Function signature

We begin with the signature of the `maxvol` function. We need the matrix A, of course, which we will store in a `NumPy` array called `a`. Next, we need the parameter δ, which we will store in the variable `delta`, and an iteration limit, which store in the variable `max_iter`. We also allow for an optional initial submatrix, specified by a list or array of row indices, which we name `initial_rows`. The return value should be the δ-dominant matrix $A_\square$, which we return in terms of its row indices in the given matrix A. Thus, we arrive at the signature in Listing 1.

Listing 1: function signature

```
def maxvol(
    a: NDArray[np.float64],  # shape = (n, r)
    initial_rows: Optional[NDArray[np.int64]] = None,  # shape = (r,)
    delta: float = 1e-2,
    max_iter: int = 100
) -> Optional[NDArray[np.int64]]:  # shape = (r,)
```

If we are given a square matrix A, then the rest of the algorithm will generate indexing errors; in any case, the maximum volume submatrix of a square tall matrix is the matrix itself, so we can return early if a square matrix is given. If `n` and `r` are the numbers of rows and columns of A, then this check is given by Listing 2.

Listing 2: square matrix check

```
if n == r:
    return np.arange(r)
```

We note that `np.arange(r)` generates an array whose entries are $1, 2, \ldots, r$. This is precisely the sequence of row indices of the entire square matrix, as desired.

Initialization of $A_{\boxdot}^{(0)}$

Next, we move on to the issue of initializing the nonsingular starting submatrix $A_{\boxdot}^{(0)}$. If `initial_rows` is supplied, then we will assume that the user has ensured that `initial_rows` determines a nonsingular submatrix. If `initial_rows` is not supplied, then it is up to us to determine a nonsingular submatrix. This can be done easily by applying Gaussian elimination with partial pivoting (that is, with row pivotoing). We note that Gaussian elimination on the $n \times r$ matrix A requires r elimination steps, each of which requires $\mathcal{O}(nr)$ computations, giving the entire process a computational complexity of $\mathcal{O}(nr^2)$, which is acceptable (we are mainly concerned with achieving linear complexity in n).

If `initial_rows` is given, then we need to compute the indices of the rows of A not in the initial submatrix as well, as we need them to work with $Z^{(k)}$. We can do Gaussian elimination using the `scipy.linalg.lu` function, which will return the permutation of the rows obtained by partial pivoting. This is given as an array of row indices; the first r elements of the permutation determine a nonsingular submatrix, and the remaining elements give us the rows of A not in the submatrix. We store the current submatrix row indices in the variable `submat_rows`, and the current remaining row indices in `other_rows`. Thus, the initialization is given in Listing 3.

Listing 3: $A_{\boxdot}^{(0)}$ initialization

```
if initial_rows is None:
    # p_indices=True to get row index array instead of permutation matrix.
    # Return of lu is a tuple (p, l, u). We only need the p entry,
    # which is the array of row indices of the permutation.
    p = scipy.linalg.lu(a, p_indices=True)[0]

    submat_rows = p[:r]  # get first r elements
    other_rows = p[r:]  # get remaining elements
else:
    submat_rows = initial_rows

    # find other rows by building a set of all indices and set-subtracting
    # given initial submatrix row indices. Then convert to array of indices.
    other_rows_set = set(range(n)).difference(map(int, submat_rows))
    other_rows = np.array(tuple(other_rows_set))
```

Initialization of $Z^{(0)}$

We have an efficient update rule for $Z^{(k)}$, but we still need to initialize $Z^{(0)}$. The only way to do this is by using the definition, that is (by Lemma 1),

$$Z^{(0)} = B^{(0)}\left(J^{(0)}, :\right) = A\left(J^{(0)}, :\right)\left(A_{\boxdot}^{(0)}\right)^{-1}. \tag{132}$$

The most stable and efficient way to do this is by using a linear system solver (rather than computing the inverse of $A_{\boxdot}^{(0)}$ explicitly). This can be done fairly easily with `np.linalg.solve`. The onl wrinkle is that we are multiplying by an inverse matrix on the *right*, and this command computes the product with an inverse matrix on the *left*. We can deal with this by transposing the inputs and transposing the output of `np.linalg.solve`.

Noting that $A\left(J^{(0)}, :\right)$ can be obtained by taking the rows of `a` stored in `other_rows`, and $A_{\boxdot}^{(0)}$ can be obtained by taking the rows of `a` stored in `submat_rows`, the initialization of $Z^{(0)}$, which we store in the variable `z`, is given in Listing 4.

Listing 4: $Z^{(0)}$ initialization

```
z = np.linalg.solve(a[submat_rows].T, a[other_rows].T).T
```

We remark that `np.linalg.solve` uses Gaussian elimination on $A_{\boxdot}^{(0)} \in \mathbf{R}^{r\times r}$ to apply $\left(A_{\boxdot}^{(0)}\right)^{-1}$ to n vectors of size r, so this step has a computational complexity of $\mathcal{O}(nr^2 + r^3) \subseteq \mathcal{O}(nr^2)$, which is acceptable.

Loop setup

Python doesn't have a `do-while/repeat-until` loop construct; since we want to terminate after `max_iter` iterations in any case, we can use a `for` loop and `if-break` to simulate the `repeat-until` in Algorithm 1. Furthermore, the `if` statement in Algorithm 1 is the same as the loop stopping condition, so we can use the `if-break` simultaneously to exit the loop and to do the `if` statement on line 5. Thus, the beginning of our loop computes the maximum modulus element of $Z^{(k)}$ (that is, the Chebyshev norm) and exits if its modulus is less than $1 + \delta$. If we need to exit the loop, then we also need to return `submat_rows` immediately, so we can do the exit and return all at once. See Listing 5.

Listing 5: loop setup

```
# use dummy index _, as we don't need the iteration index
for _ in range(max_iter):

    # np.argmax returns the index in the flattened array, so unravel
    # to get the 2-dimensional index.
    i_rel, j = np.unravel_index(np.argmax(np.abs(z)), z.shape)
    max_mod_el = z[i_rel, j]

    if np.abs(max_mod_el) < 1 + delta:
        return submat_rows
    # loop continues...
```

Updating Z

For the Z update, we recall the update rule (131). Since most of the content of D is the same as $Z^{(k)}$, we can store the i' row of $Z^{(k)}$, then replace the j row of $Z^{(k)}$ with e_j^T, so that $Z^{(k)}$ becomes D. This requires $2r$ and r units of extra memory instead of $n-r$ copy operations and $(n-r)r$ units of extra memory. Thus, the update of `z` is given in Listing 6.

Listing 6: Z update

```
    # ...still inside loop
    # Save i' row of z and subtract e_j^T.
    right = z[i_rel].copy()
    right[j] -= 1.

    # Store e_j^T in the i' row of z.
    z[i_rel, :] = 0.
    z[i_rel, j] = 1.

    # In-place update of z. Divide by max_mod_el before matrix multiply.
    z -= z[:, j : j+1] @ (right[None] / max_mod_el)
    # loop continues...
```

Updating row indices

The last thing to do is to update the row index sequence of the current submatrix and the row indices of the current Z matrix. Following the update rules for $I^{(k)}$ and $J^{(k)}$, we see that this amounts to swapping the values `submat_rows[j]` and `other_rows[i_rel]`, as in Listing 7.

Listing 7: row index update

```
    # ...still inside loop
    temp = submat_rows[j]
    submat_rows[j] = other_rows[i_rel]
    other_rows[i_rel] = temp
```

Return value

If the loop does not exit as a result of having found a δ-dominant submatrix, that is, if the loop completes `max_iter` iterations, then we want to return `None` to indicate the failure to converge. By default, if no `return` statement is encountered in a Python function, then the function returns `None`, so we simply leave the rest of the function blank after the loop.

4.3 Complete code

Bringing all the snippets above together, we obtain the full code for the `maxvol` algorithm (Listing 8).

Listing 8: `NumPy` implementation of `maxvol`

```
def maxvol(
        a: NDArray[np.float64],
        initial_rows: Optional[NDArray[np.int64]] = None,
        delta: float = 1e-2,
        max_iter: int = 100
) -> Optional[NDArray[np.int64]]:
    """
    :param a: An n x r matrix of rank r.
    :param initial_rows: A set of row indices in the matrix a giving us
    an initial nonsingular r x r submatrix. Uses Gaussian elimination to
    choose an initial set of rows if initial_rows is None.
    :param delta: delta-dominance delta value.
    :param max_iter: Maximum number of maxvol iterations.
    :return: Row indices of a delta-dominant submatrix of a. None if
    convergence fails to occur within max_iter iterations.
    """
    # input validation
    n, r = a.shape
    assert r <= n

    # In the edge case that a is square, the best we can do is return a
    # itself (that is, the submatrix rows are all the rows).
    if n == r:
        return np.arange(r)

    # Initialize a nonsingular submatrix.
    # We only track the rows of the submatrix, as we do
    # not need the submatrix itself in the algorithm, and we can always
    # retrieve the submatrix from the original matrix using the row indices.
    if initial_rows is None:
        # Use Gaussian elimination with partial pivoting to get rows
        # of a nonsingular submatrix. This operation is O(nr^2).
        p = scipy.linalg.lu(a, p_indices=True)[0]

        # Nonsingular submatrix rows are packed into the first r indices,
        submat_rows = p[:r]
        # and other rows are in the remaining indices.
        other_rows = p[r:]
    else:
        # Use given rows of a.
        submat_rows = np.array(initial_rows, dtype=int)
        # Get other rows of a.
        other_rows_set = set(range(n)).difference(map(int, submat_rows))
        other_rows = np.array(tuple(other_rows_set))

    # Get initial z = a[other_rows] @ (a[submat_rows])^{-1}.
    # Use np.linalg.solve to avoid computing matrix inverse.
    # Note that this operation is O(nr^2).
    z = np.linalg.solve(a[submat_rows].T, a[other_rows].T).T

    for _ in range(max_iter):
        # Get rows to swap by finding the maximum modulus element of z.
        i_rel, j = np.unravel_index(np.argmax(np.abs(z)), z.shape)
        max_mod_el = z[i_rel, j]

        # Stop if the current submatrix is delta-dominant.
        if np.abs(max_mod_el) < 1 + delta:
            return submat_rows

        # Update z.
        right = z[i_rel].copy()
        right[j] -= 1.

        z[i_rel, :] = 0.
        z[i_rel, j] = 1.

        z -= z[:, j : j+1] @ (right[None] / max_mod_el)

        # Update row index sequences.
        temp = submat_rows[j]
        submat_rows[j] = other_rows[i_rel]
        other_rows[i_rel] = temp

    # Default return value is None.
```

5 Experiments

In this section, we explore the capabilities of `maxvol` through several numerical experiments. All experiments can be replicated by running the `test.ipynb` notebook[7].

Validation

To validate that our implementation is working correctly, we start by applying it to very small matrices, small enough that we can use the brute-force method to find the submatrix of maximal volume. In Table 1 we see the volume of several submatrices obtained by applying `maxvol` to a 15×5 matrix, alongside the volume of the actual maximum-volume submatrix. The matrices are generated by choosing each entry uniformly and independently at random from the interval $[0, 1]$.

We observe that in the case of this small problem, `maxvol` frequently finds an actual maximum-volume submatrix with much less effort than the brute-force method. In the cases when `maxvol` does not find a maximum-volume submatrix, the submatrix it finds has nearly maximum volume.

We should also make sure that `maxvol` actually returns a δ-dominant submatrix. This can be verified easily by computing $\left\|AA_{\square,\delta}^{-1}\right\|_{\infty}$, where $A_{\square,\delta}$ is the submatrix returned by `maxvol`. This time, we work with random 20000×100 matrices to demonstrate the speed of `maxvol`. In Table 2, we present the computed Chebyshev norms for the returned submatrices, using $\delta = 0.01$. We see that all the returned submatrices are δ-dominant, some just barely.

Fast pseudo-skeleton decomposition

Our next experiment involves using `maxvol` to compute the pseudo-skeleton decomposition using the alternating row-column idea described at the beginning of section 2.3. We compare the speed and

[7] You can also view the output on GitHub easily here. In case the PDF link doesn't work: `https://github.com/jacobhauck/MST-Coursework/blob/main/Math%205601/Independent%20Study%20Project/test.ipynb`

Trial #	`maxvol` Volume	Maximum Volume
1	0.536675	0.536675
2	0.554545	0.554545
3	**0.516685**	**0.523402**
4	0.272270	0.272270
5	0.393877	0.393877
6	0.747953	0.747953
7	0.620132	0.620132
8	**0.527709**	**0.528394**

Table 1: Volume of submatrices returned by `maxvol` and actual maximal volumes. Cases where `maxvol` returned a submatrix with less than maximal volume are in bold (trials 3 and 8).

Trial #	$\left\|AA_{\square,\delta}^{-1}\right\|_\infty$
1	1.003222
2	1.000000
3	1.002376
4	1.007283
5	1.009181

Table 2: Checking the δ-dominance condition for submatrices returned by `maxvol`. We use $\delta = 0.01$.

accuracy of this method of finding a pseudo-skeleton decomposition to the accuracy and speed of the truncated SVD approximation of the same rank, which, by Theorem 1 is the best low-rank approximation.

To do this effectively, we need a large, dense, almost low-rank matrix. Such a matrix can be constructed easily by sampling from a 2D Gaussian random field. We use the code in Li et al. [5] with the default settings to generate a 1024 × 1024 Gaussian random field, and save it in the file `x.npy`.

The singular values of this matrix decay very rapidly, as shown in Figure 1, which means that it can be well-approximated by a low-rank matrix by Theorem 1.

We compute the best rank-20 approximation of this matrix using both truncated SVD and `maxvol`-based pseudo-skeleton decomposition. The pseudo-skeleton had an error of about 2 times the SVD error in Frobenius norm, which is quite good, considering that the SVD approximation is the best possible. More importantly, the pseudo-skeleton decomposition was computed **thirty times faster** on average than the SVD, despite the fact that we used the highly optimized `np.linalg.svd` function to compute the SVD and used slow, high-level Python/`NumPy` code to compute the `maxvol`-based pseudo-skeleton decomposition, suggesting that performance improvements of up to 100 times are likely possible.

Pictured in Figure 2 are visualizations of the original Gaussian random field alongside the SVD and pseudo-skeleton approximations.

Fast optimization with `maxvol`

The last experiment we did is taken from “How to Find a Good Submatrix” [2]. In this experiment,

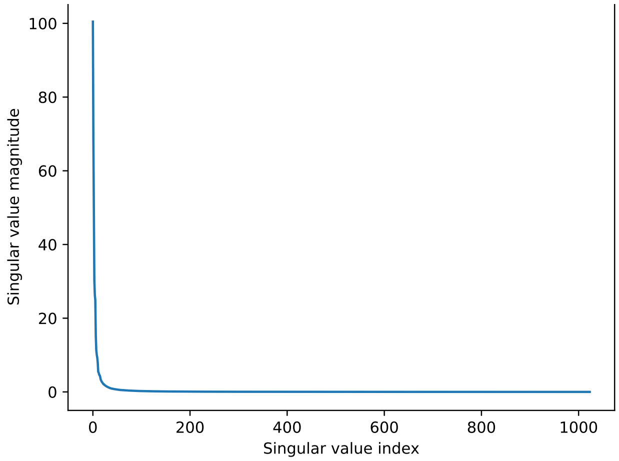

Figure 1: Singular values of a 2D Gaussian random field

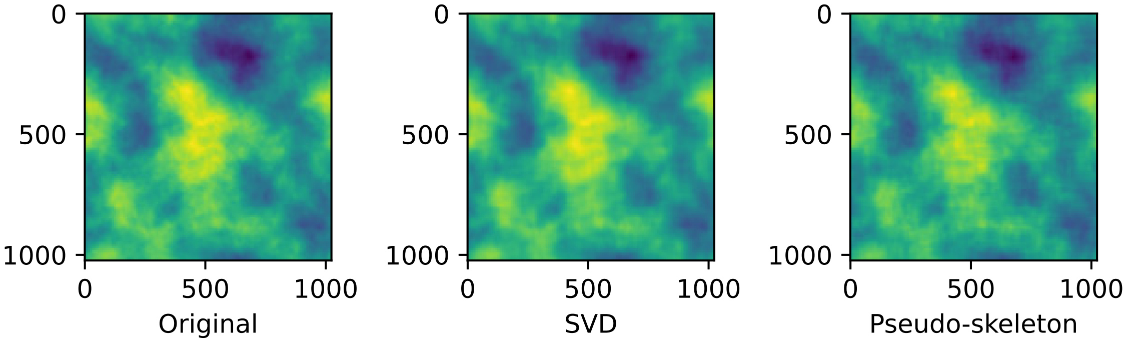

Figure 2: Rank-20 approximations of a 2D Gaussian random field.

we are given a large $m \times n$ matrix of (known) low rank r. We can use the alternating `maxvol` method used to compute the pseudo-skeleton approximation to find a submatrix with high volume. Then, we find the largest element in absolute value of that submatrix. This element should approximate the largest element in absolute value of the given $m \times n$ matrix.

To generate an $m \times n$ matrix of rank r, we choose the entries of matrices U and V of sizes $m \times r$ and $n \times r$ uniformly and random. Then, we take the QR decomposition of these matrices to obtain the Q factors Q_U and Q_V of full-rank. Lastly, we select the elements of a diagonal $r \times r$ matrix D all positive and form the random $m \times n$ matrix $M = Q_U D Q_V^T$ of precisely rank r.

Summarized in Table 3 are the results of this experiment on several random 1000×1000 matrices. We see that the determined maxima are fairly close to the true maxima, in the sense that they are much better than guessing at random, as evidenced by the mean and standard deviation of the absolute value of elements of the generated matrices.

Trial #	True max	True mean	True std.	Approximate max
1	0.003309	0.000451	0.000413	0.001745
2	0.003262	0.000468	0.000360	0.002201
3	0.003578	0.000514	0.000395	0.001665
4	0.003420	0.000454	0.000350	0.002207
5	0.003175	0.000480	0.000365	0.002400

Table 3: Approximate maxima found using pseudo-skeleton decomposition

6 Conclusion

Performing linear algebraic operations on matrices is a ubiquitous task, and, in many cases, the matrices involved are enormous. Reducing the size of a matrix by low-rank approximation is an effective way to make very slow or even infeasible computations much more accessible.

Unfortunately, the theoretically optimal way to reduce the matrix size, truncated singular value decomposition (Theorem 1), may be as computationally expensive to perform as the task for which the matrix needs to be reduced in the first place. Therefore, it is desirable to have an algorithm that can find a reduced approximation of a matrix with complexity on the order of the size of the *reduced* matrix, rather than the *original* matrix. This essentially boils down to finding a good submatrix of the original matrix.

We have shown that a good enough submatrix can yield an approximation that is nearly as good as truncated singular value decomposition. In the process of understanding this theory, we were led to the idea of maximum volume as the criterion for a good submatrix, and from there to the idea of the `maxvol` algorithm. We showed that this algorithm is capable of finding a nearly maximal volume submatrix with a total number of computations on the order of the size of the submatrix, not the original matrix.

Based on our experiments, we observed that `maxvol` can provide a speedup of several orders of magnitude in the construction of low-rank approximations, and these approximations are of fairly high quality.

As a final remark, we mention that the maximum volume idea can be extended readily to the more general area of **tensor decomposition**, which is the process of factorizing a tensor into the tensor product of other tensors. Much like how we used matrix decomposition to construct an efficient, low-rank representation of a matrix, tensor decomposition can be used to construct efficient, low-rank representations of tensors. Because the complexity of `maxvol` depends primarily on the rank of the approximation, it can be used to overcome the curse of dimensionality by efficiently constructing low-rank approximations of high-dimensional tensors. The computational cost of operating on such an approximation scales linearly in the dimension (as opposed to the exponential scaling incurred by working on a dense, full-rank tensor).

References

[1] M. S. Bartlett. An Inverse Matrix Adjustment Arising in Discriminant Analysis. *The Annals of Mathematical Statistics*, 22(1):107–111, March 1951.

[2] S. A. Goreinov, I. V. Oseledets, D. V. Savostyanov, E. E. Tyrtyshnikov, and N. L. Zamarashkin. *How to Find a Good Submatrix*, pages 247–256. WORLD SCIENTIFIC, April 2010.

[3] S.A. Goreinov, E.E. Tyrtyshnikov, and N.L. Zamarashkin. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications*, 261(1-3):1–21, August 1997.

[4] Keaton Hamm. Generalized Pseudoskeleton Decompositions, June 2022.

[5] Zijie Li, Dule Shu, and Amir Barati Farimani. Scalable Transformer for PDE Surrogate Modeling, May 2023.

[6] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra.* Society for Industrial and Applied Mathematics, Philadelphia, 2008.

[7] Vadim Olshevsky, editor. *Structured Matrices in Mathematics, Computer Science, and Engineering: Proceedings of an AMS-IMS-SIAM Joint Summer Research Conference, University of Colorado, Boulder, June 27-July 1, 1999.* Number 280-281 in Contemporary Mathematics. American Mathematical Society, Providence, R.I, 2001.

[8] Ivan Oseledets. Compact matrix form of the d-dimensional tensor decomposition, October 2009.

[9] Ivan Oseledets and Eugene Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, January 2010.

[10] Beresford N. Parlett. *The Symmetric Eigenvalue Problem.* Number 20 in Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, 1998.