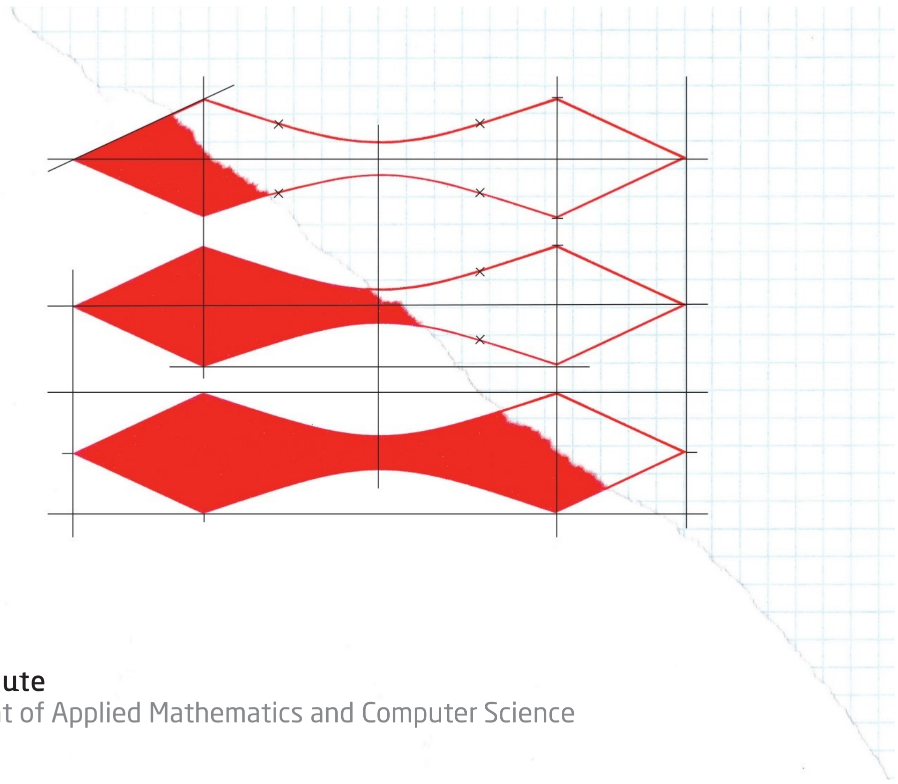
M.Sc. Thesis
Master of Science in Engineering



DTU Compute
Department of Applied Mathematics and Computer Science

Sketching for Scientific Computing

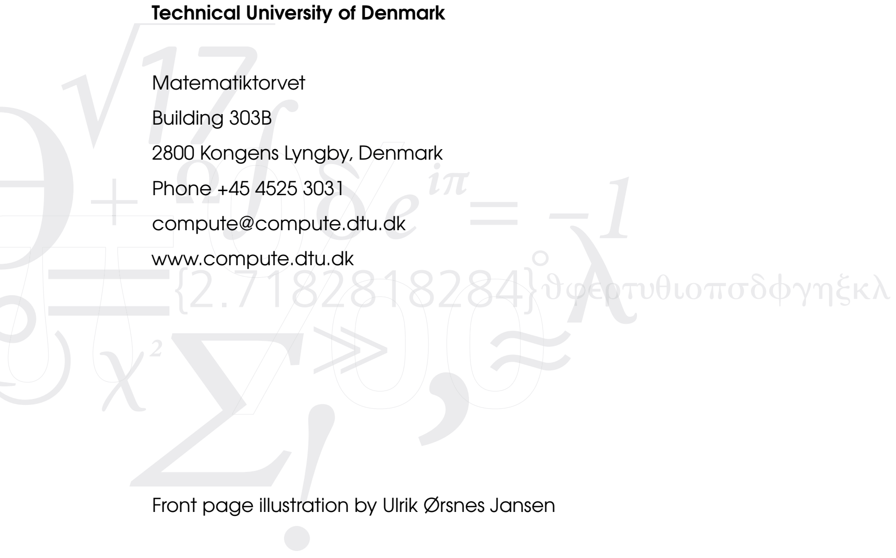
A practical guide

Joachim Brink Harck
Kristoffer Ørsnes Jansen

Kongens Lyngby 2018

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk



Front page illustration by Ulrik Ørsnes Jansen

Abstract

An exciting, recent development in numerical linear algebra is the use of randomisation as a resource, often through a process known as sketching, where a large matrix is replaced by smaller matrix that approximates it well. Much of the literature on sketching is either deeply rooted in theoretical computer science, with little attention on practical performance, or concentrated on narrow areas of applications, obscuring the simplicity of the underlying principles. This thesis aims to bridge the gap between theory and practice, and create a framework for sketching applications that makes it easy to identify and apply sketching in a large range of algorithms.

We describe the fundamental theory behind sketching, focusing on the mathematical concept of subspace embeddings. Four different sketching methods are presented and compared in terms of theoretical guarantees and complexities. The practical performance of the above methods is then tested in various linear algebra problems.

Through our review of literature and the performed experiments, we identify three basic operations. These form the cornerstones of most uses of sketching, allowing for a modular approach to constructing algorithms. We find that simple implementations can yield significant computational advantages, often using sketch dimensions well below the theoretical bounds. Usually, the gains come at the price of approximation errors, the measurement and consideration of which is essential for the success of the methods in practice.

Sketching has the potential to be a valuable tool in many areas of scientific computing. It is at its most useful when coupled with domain-specific knowledge used to provide reasonable assumptions, tighter analysis and suitable error measures. Our hope is that this report can serve as a practical guide to understanding the basic theory and computational aspects of sketching, while making the underlying ideas and methods accessible to a larger audience.

Resumé

Brugen af tilfældighed som en beregningsmæssig resource er en ny og spændnde udvikling indenfor feltet numerisk lineær algebra. Ofte sker det gennem en proces kaldet sketching, hvor en stor matrix erstattes af en mindre matrix med lignende egenskaber. Meget af literaturen på området er enten dybt forankret i teoretisk datalogi uden nævneværdige beskrivelser af praktiske aspekter, eller koncentreret om specifikke anvendelsesmuligheder, hvorfor enkeltheden af de underliggende idéer let overskygges. Formålet med denne afhandling er at mindske kløften mellem teori og praksis, og skabe en strukturel ramme for brugen af sketching, der gør det let at identificere og anvende sketching i et bredt udvalg af algoritmer.

Vi beskriver først den grundlæggende teori med fokus på det matematiske begreb "subspace embeddings". Der præsenteres fire forskelige sketching metoder, hvis teoretiske egenskaber og kompleksiteter sammenlignes, hvorefter metodernes praktiske formåen efterprøves i forskellige matematiske problemer.

På baggrund af den gennemgåede literatur og de udførte ekseperimenter, identificerer vi tre basale operationer. Disse fungerer som grundsten i de fleste anvendelser af sketching, og gør det muligt at konstruere modulopbyggede algoritmer. Vi ser, at selv simple implementeringer med parametre langt fra de teoretiske krav kan give betydelige beregningsmæssige fordele. Dette er ofte på bekostning af approksimationsfejl, og det er dermed essentielt at kunne vurdere fejlens betydning, før sketchingmetoder kan anvendes i praksis.

Sketching har potentiale som et værdifuldt redskab i mange områder af den anvendte matematik. Metoderne fungerer bedst, når de kan kobles med domænespecifik viden, der kan bruges til at opstille rimelige antagelser, bedre teoretiske garantier og passende fejlmål. Vi håber, at denne afhandling kan fungere som en praktisk introduktion til sketching, der tydeliggør den underliggende teori og de beregningsmæssige aspekter, og dermed gør metoderne lettere tilgængelige.

Preface

This thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark (DTU Compute) in fulfilment of the requirements for acquiring a M.Sc. degree in Mathematical Modelling and Computation. The thesis is jointly written by students Joachim Brink Harck (s134611) and Kristoffer Ørsnes Jansen (s134570) representing a workload of 30 ECTS points for both students. The work was carried out from January 22nd 2018 to June 22nd 2018 under supervision of Associate Professor Martin Skovgaard Andersen and Professor Per Christian Hansen, both of DTU Compute.

Contributions. Both authors contributed equally to all chapters of this thesis. The following table shows the main author for specific sections.

Chapter	Main author	Sections
1 Introduction	Joachim	
2 Mathematical foundations	Joachim	2.1, 2.1.1, 2.1.4
	Kristoffer	2.1.2, 2.1.3, 2.2
3 Sketching in practice	Joachim	3, 3.1.1, 3.2.2, 3.3.1, 3.3.2
	Kristoffer	3.1.2, 3.1.3, 3.2.1, 3.2.3, 3.3.3
4 Assessment of applicability	Joachim	4.1
	Kristoffer	4.2, 4.3
5 Conclusion	Kristoffer	

Acknowledgements. We would like to thank Martin and Per Christian for their constructive discussions, valuable feedback and overall helpfulness. We are also grateful to Matias Lolk Andersen for taking the time and effort to proofread our work.

Kongens Lyngby, June 28, 2018

[illegible] Kristoffer Ø. Jansen

Joachim Brink Harck Kristoffer Ørsnes Jansen

List of symbols

$\lvert S\rvert$	Cardinality of the set S
$\lVert\mathbf{A}\rVert_2$	Spectral norm of the matrix $\mathbf{A}$
$\lVert\mathbf{A}\rVert_F$	Frobenius norm of the matrix $\mathbf{A}$
∇f	Gradient of the function f
$\nabla^2 f$	Hessian of the function f
$\mathbf{A}_{ij}$	(i,j)-th entry of the matrix $\mathbf{A}$
$\mathbf{A}_{(i)}$	i-th row of the matrix $\mathbf{A}$
$\mathbf{A}^{(j)}$	j-th column of the matrix $\mathbf{A}$
$\mathbf{D}$	Diagonal matrix
δ	Failure probability parameter
ε	Approximation parameter
$\mathbb{E}\left[X\right]$	Expectation of the stochastic variable X
$\mathbf{G}$	Matrix with independent standard normally distributed entries
$\mathbf{H}$	Hadamard matrix
$\mathbf{I}$	Identity matrix
k	Sketch size parameter
$\kappa\left(\mathbf{A}\right)$	Condition number of the matrix $\mathbf{A}$
ℓ	Leverage scores
ln	Natural logarithm
$\mathbb{N}$	Set of natural numbers
$\mathcal{N}\left(\mu,\sigma^2\right)$	Normal distribution with mean μ and variance σ^2
$\mathrm{nnz}\left(\mathbf{A}\right)$	Number of nonzero entries in the matrix $\mathbf{A}$

$O(x)$	Asymptotic upper bounds with respect to the parameter x
$\mathbf{P}$	Projection matrix
$\mathbf{\Pi}$	Distribution over specific matrices
$\mathrm{poly}(x)$	Polynomial in x of unspecified order
$\Pr[E]$	The probability of the event E
$\mathbf{Q}$	Orthonormal matrix of the QR decomposition
$\mathbf{R}$	Upper triangular matrix of the QR decomposition
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^m$	Set of real vectors of length m
$\mathbb{R}^{m\times n}$	Set of real matrices of size $m \times n$
$\mathcal{R}(\mathbf{A})$	Range/column space of the matrix $\mathbf{A}$
$\mathbf{S}$	Sketching matrix
σ	Standard deviation
$\sigma(\mathbf{A})$	Singular value of the matrix $\mathbf{A}$
$\mathbf{\Sigma}$	Diagonal matrix containing singular values or covariance matrix
$\mathbf{U}$	Orthonormal basis or matrix of left singular vectors
$\mathbf{U_A}$	Orthonormal basis for the range/column space of the matrix $\mathbf{A}$
$\mathbf{V}$	Matrix of right singular vectors
x_i	i-th entry of the vector $\mathbf{x}$
$X \sim \mathcal{D}$	The random variable X follows the distribution $\mathcal{D}$

Contents

CHAPTER 1

Introduction

Big data has long since gone from a concept in the scientific community to a buzzword in mainstream media. Information is gathered from almost everywhere and at all times and the sheer volumes of data present both interesting opportunities and considerable challenges for those seeking to make use of it. Numerical linear algebra is one of the scientific fields that is instrumental in trying to develop and update traditional data processing techniques to handle the vast amounts of information.

In linear algebra, data is often represented as a matrix with the rows corresponding to for example observations, users or items and columns to properties, ratings or similar descriptive features. One could for example describe the ratings of movies by users of the streaming service Netflix in a matrix, which would then contain around 125 million rows corresponding to the number of subscribers in the first quarter of 2018, and several thousand columns for all the available films. Performing standard linear algebra operations using such data sets is computationally demanding, impractical and in some cases outright impossible.

A novel approach to developing efficient alternative algorithms has emerged within the last two decades and distinguishes itself by exploiting randomisation as a computational resource. Previously this was thought of as a desperate and last resort but recent results have revealed powerful algorithmic and statistical properties. To illustrate the merits of this line of thought, consider the ubiquitous overdetermined least squares problem, where given an $m \times n$ coefficient matrix $\mathbf{A}$ with $m > n$ and a response vector $\mathbf{b}$ of length m, the aim is to find the vector $\mathbf{x}$ minimising the residual error $\|\mathbf{Ax} - \mathbf{b}\|_2$. The following MATLAB code sets up such a problem by constructing a large random coefficient matrix and response vector:

```
>> m = 2^19; n = 2^10;
>> rng(0); A = randn(m,n); b = A*rand(n,1) + rand(m,1);
```

Solving the least squares problem can be done using `Matlab`'s \ or `mldivide` operator, which, when $\mathbf{A}$ is large, can be rather cumbersome:

```
>> tic; x = A\b; toc;
Elapsed time is 277.843295 seconds.
```

As the problem is heavily overdetermined, it seems reasonable to assume that the rows of $\mathbf{A}$ contain a considerable amount of redundancy. An alternative approach would therefore be only to use a subset of the rows when solving the problem. This can be easily implemented by uniformly sampling $k < m$ rows from $\mathbf{A}$ and solving the smaller problem:

```
>> tic; k = 2^15;
>> row_id = randsample(m,k); SA = A(row_id,:); Sb = b(row_id);
>> x_uniform = SA\Sb; toc;
Elapsed time is 17.207724 seconds.
>> norm(A*x_uniform - b)/norm(A*x - b)
ans = 1.0143
```

In the above, we sample $k = 2^{15}$ rows and obtain an approximate solution with a residual norm less than 1.02 times larger than without sampling rows, and the solution is computed about 16 times faster. However, this naïve method of uniform sampling is not very robust, as the following example shows:

```
>> A(1:end-1,end) = 1e-6*randn(m-1,1);
>> x = A\b;
>> SA = A(row_id,:); Sb = b(row_id);
>> x_uniform = SA\Sb;
>> norm(A*x_uniform - b)/norm(A*x - b)
ans = 1571.0422
```

In the above, the last row of $\mathbf{A}$ carries a huge amount of information and if this row is not sampled, the residual norm is far from impressive. A consistent method must therefore either use all the information of $\mathbf{A}$ or depend on the input matrix in some way. Let $\mathbf{S}$ be a $k \times m$ matrix where each column has a single nonzero entry, either $+1$ or -1, placed randomly. Multiplying a matrix with $\mathbf{S}$ corresponds to splitting the rows into k sets and randomly adding or subtracting the rows within each set to yield k new rows. This matrix has some very interesting properties and applying it to the least squares problem, again using $k = 2^{15}$, we see the following:

```
>> tic;
>> randomsigns = 2*randi(2,m,1) - 3;
>> S = sparse(randi(k,m,1),1:m,randomsigns,k,m);
>> SA = S*A; Sb = S*b;
>> x_sketch = SA\Sb;
>> toc;
Elapsed time is 21.687518 seconds.
>> norm(A*x_sketch - b)/norm(A*x - b)
ans = 1.0167
```

This method might be slightly slower than uniform sampling but it is still significantly faster than solving the original least squares problem and yields a relative residual norm of less than 1.7%.

A reduction of a linear algebra problem as in the considered example of the least squares problem is one of the central concepts in randomised numerical linear algebra and is known as *sketching*. In accordance with the conventional meaning of the word, the computed *sketch* of the input matrix should be obtained quickly and represent essential information of the original problem. The sketching procedure is expressed as a matrix multiplication with a *sketching matrix* which, as apparent from the motivating example presented above, should have certain properties. This requires a definition of the "essential information" that should be captured in the sketch.

A matrix can be viewed as an operator on vectors and emulating the action it has on these is therefore an intuitive requirement for the sketch. A *subspace embedding* for a given matrix is a linear operator that preserves distances in the range space of the matrix and it turns out that sketching matrices that are chosen as subspace embeddings have very nice properties when it comes to applications in linear algebra problems. There are many different ways of constructing subspace embedding matrices, one is to follow the steps taken when forming $\mathbf{S}$ in the previous example. This type of sketching matrix is an example of a random projection while the uniform sampling approach could be viewed as a sampling matrix. As mentioned however, in order to feasibly obtain a subspace embedding based on sampling, we need to adjust the sampling probabilities to the input matrix so as to avoid missing indispensable rows.

The theory of sketching matrices focuses heavily on the number of rows required for them to be subspace embeddings. Often the theoretical bounds presented require an exorbitant number of rows but in practice it is often possible to achieve decent results using a very reasonable number of rows as was evident in the example above.

Returning to the overdetermined least squares problem, we have now identified both the uniform sampling and the matrix $\mathbf{S}$ as sketching matrices, albeit with very different properties. Observe that actually solving the linear system

$$(\mathbf{SA})^\top(\mathbf{SA})\mathbf{x} = (\mathbf{SA})^\top(\mathbf{Sb})$$

has the same computational complexity as solving $\mathbf{A}^\top\mathbf{A}\mathbf{x} = \mathbf{A}^\top\mathbf{b}$, since $(\mathbf{SA})^\top(\mathbf{SA})$ and $\mathbf{A}^\top\mathbf{A}$ are of the same size. The advantage of sketching is therefore entirely in the matrix-matrix and matrix-vector products on the left and right hand side, respectively. This shows that the application of sketching in solving the normal equation essentially boils down to approximating matrix multiplication. It turns out that a large number of sketching applications can be traced back to a few central linear algebra operations which suggests that a modular set up could be used to describe most uses of sketching.

The aim of this thesis is to give a practical introduction to sketching in scientific computing and establish a framework for analysing the use of sketching in algorithms. Much of the existing literature focuses on sketching in specific applications or on theoretical error guarantees and complexity analyses of the methods. We present a more general approach, highlighting the intuition behind and interplay between different sketching applications. Our hope is to give the reader an understanding of both the advantages and pitfalls of sketching and to provide a toolbox for implementing sketching in a wide range of applications.

Roadmap. We commence our investigation of sketching as a computational resource by presenting the fundamental theory in Chapter 2. We formally define the notion of a subspace embedding and present four ways of constructing sketching matrices with a focus on their theoretical properties.

In Chapter 3, we take a more practical approach and consider nine different applications of sketching divided into three separate stages. The first stage consists of the basic building blocks to which most sketching applications can be traced, the second stage of generic linear algebra problems such as overdetermined least squares, while the third stage is dedicated to concrete applications demonstrating the computational advantages of sketching in more realistic settings. Throughout the chapter, we illustrate the methods with instructive algorithms and experimental results.

Additional analysis of results across the various sketching methods and applications is carried out and discussed in Chapter 4, and the introduced building block framework is assessed. We also present topics and applications for further research. Chapter 5 wraps up the report with a conclusion.

CHAPTER 2 Mathematical foundations

As the interest in, availability of and access to huge data sets continue to grow, so does the need for fast and efficient data processing algorithms, and randomised numerical linear algebra has a central role to play. The randomness lies in the creation of the sketch which must capture essential information of a given input matrix in order to be used as a surrogate in algorithms. The probabilistic nature of the approach gives rise to statistical results where important properties are proven to hold with a certain probability.

What is the nature of the essential information that must be captured by the sketch? How can appropriate sketching matrices be constructed? Which statistical guarantees can be obtained for sketching methods? These are some of the key issues that will be addressed in this chapter.

The chapter is structured as follows. We introduce the notion of subspace embeddings which will provide the main theoretical framework for the sketching methods that are subsequently described. Four different methods spanning four different approaches will be covered: leverage score sampling, Gaussian projection, the subsampled randomised Hadamard transform and sparse projection. Finally, we compare the characteristics and expected performance of these methods with a focus on the sketch size and computation time. With the aim of providing an intuitive understanding of the mathematics behind sketching, larger proofs are referred to the appendix or to the relevant literature.

2.1 The sketching matrix

The goal of sketching is to create a substitute for a given input matrix $\mathbf{A}$ that encapsulates the most important characteristics. This is done by computing a sketch $\mathbf{SA}$, where $\mathbf{S}$ is a sketching matrix. Looking at matrices as linear operators between vector spaces, it seems intuitive to require that the sketch acts similarly on sets of vectors as the matrix $\mathbf{A}$. One way of enforcing this is to demand that $\mathbf{S}$ approximately preserves distances in the column space of $\mathbf{A}$, i.e. that

$$(1-\varepsilon)\|\mathbf{y}\| \leq \|\mathbf{Sy}\| \leq (1+\varepsilon)\|\mathbf{y}\|,$$

for all $\mathbf{y} \in \mathcal{R}(\mathbf{A})$ and some $\varepsilon > 0$, where $\mathcal{R}(\mathbf{A}) \subseteq \mathbb{R}^m$ is the range or column space of $\mathbf{A} \in \mathbb{R}^{m\times n}$. This leads to the definition of a subspace embedding.

Definition 1 (Subspace embedding)**.** For $p \in \mathbb{N}$, a matrix $\mathbf{S} \in \mathbb{R}^{k\times m}$ is called an $(1 \pm \varepsilon)$ ℓ_p-subspace embedding for some fixed matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ if, for all $\mathbf{x} \in \mathbb{R}^n$, it holds that

$$(1-\varepsilon)\left\|\mathbf{Ax}\right\|_p \leq \left\|\mathbf{SAx}\right\|_p \leq (1+\varepsilon)\left\|\mathbf{Ax}\right\|_p.$$

Our primary focus will be on ℓ_2-subspace embeddings for which the following characterisations will prove useful.

Theorem 1 ([NN13, p. 5])**.** *For any matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ and rank $r \leq n$, let $\mathbf{U_A} \in \mathbb{R}^{m\times r}$ be a corresponding orthonormal basis. For $\mathbf{S} \in \mathbb{R}^{k\times m}$, the following three properties are equivalent:*

(SE 1) $\mathbf{S}$ *is an* $(1 \pm \varepsilon)$ *ℓ_2-subspace embedding for* $\mathbf{A}$*.*

(SE 2) $\left\|(\mathbf{SU_A})^\top(\mathbf{SU_A}) - \mathbf{I}\right\|_2 \leq \varepsilon$*.*

(SE 3) $\max_i \left|1-\sigma_i^2\left(\mathbf{SU_A}\right)\right| \leq \varepsilon$*, where $\sigma_i\left(\mathbf{A}\right)$ is the i-th singular value of* $\mathbf{A}$*.*

These characterisations further motivate the use of subspace embeddings in the context of sketching. An ℓ_2-subspace embedding for $\mathbf{A}$ resembles the input matrix in the sense that distances in the column space of $\mathbf{A}$ (SE 1), the orthogonal property of $\mathbf{U_A}$ (SE 2) and the magnitude of the singular values (SE 3) are all approximately preserved. Thus by constructing sketching matrices satisfying the subspace embedding property, we can hope to use our sketch as a substitute for the original matrix.

In the literature (see for example [Woo14; Mah+11; Mat08]), various methods of constructing subspace embeddings have been proposed. In this report, we will focus on four specific kinds of subspace embeddings based on leverage score sampling, Gaussian projection, subsampled randomised Hadamard transforms and sparse projection. These are examples of both sampling and dense, structured and sparse transformations, and thereby cover the most important properties and ideas of sketching.

2.1.1 Leverage score sampling

Sampling is the selection of elements from a given set based on a probability distribution. In particular, the procedure of sampling rows from a matrix can be represented as a multiplication from the left by a row-sampling matrix. We can construct a $k \times m$ row-sampling matrix, based on a discrete probability distribution $\{p_j\}_{j=1}^{m}$ over its columns, in the following way: initialise an all-zero matrix, and for each row i pick column j with probability p_j, setting the (i, j)-th entry to 1.

A naïve and simple way to sample rows would be to sample from a fixed probability distribution such as the uniform distribution. This is a fast method but it has potential pitfalls when used on arbitrary input matrices as seen in the introductory example of Chapter 1. The problem with the naïve approach is that when sampling whole rows of the input matrix we need to make sure that the selected rows carry over most of the essential information. This cannot be ensured when choosing a fixed probability distribution independent of the input matrix. The sampling procedure can be improved by using an importance distribution that reflects the potential non-uniformity of the input matrix. Doing this, our sampling procedure adapts to whatever input is given and we avoid the problem from Chapter 1.

Motivated by the discussion in [Mah+11], we focus on leverage scores as such an importance distribution. Leverage scores have a long history in statistics where they are often used to identify potential outliers, since high leverage scores indicate observations that have a high influence on a fitted regression model. In the context of sketching, they form a weighting of the rows, enabling us to use the leverage scores to sample highly influential rows with high probability.

Definition 2 (Leverage scores)**.** Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ of rank r, let $\mathbf{U_A} \in \mathbb{R}^{m \times r}$ be an orthonormal basis for $\mathbf{A}$. The i-th leverage score, ℓ_i, corresponding to the i-th row of $\mathbf{A}$ is defined as $\ell_i = \left\|(\mathbf{U_A})_{(i)}\right\|_2^2$.

As the definition states, the leverage scores are computed from an orthonormal basis of the input matrix. An orthonormal basis always exists, and all orthonormal bases for a given matrix give rise to the same leverage scores due to the unitary invariance of the spectral norm. We can establish a probability distribution on the rows of $\mathbf{A}$ by assigning to each row a probability equal to the normalised leverage score, such that $p_i = \ell_i/r$ for any row $i = 1, \ldots, m$. These values are clearly all positive and sum to one since $\sum_i \ell_i = \|\mathbf{U_A}\|_F^2 = r$.

Leverage scores are often computed using a singular value decomposition or QR factorisation. It is worth noticing that these methods have complexities of $O\left(mn^2\right)$ and are computationally expensive for massive data sets [GV12]. To avoid this, we will base our importance distribution on an approximation of the leverage scores. We will demand from our approximation that for a rank r matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, the approximate leverage scores $\{q_i\}_{i=1}^m$ should constitute a probability distribution and satisfy $q_i \geq \frac{\beta}{r}\ell_i$, where ℓ_i are the exact leverage scores for $\mathbf{A}$ and β is a positive constant. The following theorem states that the approximate probabilities can be found and the complexity required to calculate them.

Theorem 2 ([Woo14, Theorem 2.13])**.** *Fix any constant $\beta \in (0, 1)$. Let ℓ_i be the leverage score distribution of an orthonormal basis $\mathbf{U_A} \in \mathbb{R}^{m\times r}$. Then it is possible to compute a distribution $q = \{q_i\}_{i=1}^m$ for which, with probability $\frac{9}{10}$, it holds simultaneously for all $i = 1, \ldots, m$ that $q_i \geq \frac{\beta}{r}\ell_i$. The time complexity is of the order $O\left(\operatorname{nnz}(\mathbf{A}) \ln m\right) + \operatorname{poly}(r \ln m)$.*

The proof of this theorem reveals a possible way of approximating the leverage scores with a complexity lower than $O\left(mn^2\right)$. It turns out that this is done by sketching the problem! Approximating leverage scores is thus in itself an area for which sketching can be used. We are still to cover the applied sketching methods and will therefore for now refer to [Woo14] for the proof of this theorem, but we will revisit this application of sketching in Section 3.2.1.

The time complexity expression given in Theorem 2 includes a $\operatorname{poly}(r \ln m)$ term, the exact structure of which becomes important for large r. Following the proof in [Woo14, Theorem 2.13], this term represents a complexity of $O\left(r^4 + r^2 \ln m\right)$. Clarkson and Woodruff [CW13, Section 6] improve slightly on this expression and state that approximate leverage scores can be found in $O\left(\operatorname{nnz}(\mathbf{A}) \ln m + r^3 \ln^2 r + r^2 \ln m\right)$ time.

So far we have allowed the matrix $\mathbf{A}$ to be rank-deficient as the definition of leverage scores is based on an orthonormal basis for $\mathbf{A}$. In order to simplify notation and to make it easier to compare with the methods introduced later, we will now assume that $\mathbf{A}$ has full column rank. This implies that the dimension of any orthonormal basis is the same as the dimension of $\mathbf{A}$. We note that this is quite a strong assumption in general, however, given a rank-deficient matrix $\mathbf{A}$, one can simply apply the following theorem to an orthonormal basis of $\mathbf{A}$, which is necessarily of full rank. Using the approximate leverage score probabilities, we can construct a subspace embedding for a given matrix using a specific type of row-sampling matrix.

Theorem 3 (Leverage score sketching matrix)**.** *Let* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *be a full rank matrix,* $\beta \in (0,1)$ *and* q *the corresponding approximate leverage scores of* $\mathbf{A}$ *from Theorem 2. Let* $\mathbf{\Pi}_{\mathrm{LEV}}^{q}$ *be the distribution on* $k \times m$ *scaled row-sampling matrices, such that for* $\mathbf{S} \sim \mathbf{\Pi}_{\mathrm{LEV}}^{q}$*,* $\mathbf{S}_{ij} = \frac{1}{\sqrt{kq_j}}$ *with probability* q_j *for all rows* i *and columns* j*. For* $0 < \delta < 1$*, if*

$$k \geq \frac{8}{3} n \beta^{-1} \varepsilon^{-2} \ln\left(\frac{2n}{\delta}\right),$$

then, with probability at least $1 - \delta$*, a matrix drawn from* $\mathbf{\Pi}_{\mathrm{LEV}}^{q}$ *is a* $(1 \pm \varepsilon)$ *subspace embedding for* $\mathbf{A}$*.*

Proof. See Appendix A.1. □

The theorem shows that a sampling matrix based on approximated leverage scores is an appropriate choice of sketching matrix. Computationally, using such sketching matrices has the advantage that **SA** can be computed in negligible time as this operation only involves sampling rows of **A**. However, we are still left to show how to approximate leverage scores which is in fact the expensive part of this sketching method. This problem will be considered in Section 3.2.1.

2.1.2 Gaussian projection

As seen above, subspace embeddings can be generated with high probability by considering the properties of the input matrix. Ideally, we would like a class of projections that can provide subspace embeddings regardless of the input. This desire is summarised in the following definition.

Definition 3 (Oblivious subspace embedding)**.** Let $\mathbf{\Pi}$ be a distribution over $k \times m$ matrices. We call $\mathbf{\Pi}$ an (ε, δ, n) oblivious subspace embedding if, with probability $1 - \delta$, a matrix drawn from $\mathbf{\Pi}$ is a $(1 \pm \varepsilon)$ subspace embedding for any $\mathbf{A} \in \mathbb{R}^{m \times n}$.

In the definition of an oblivious subspace embedding, we allow the drawn matrix to fail with a certain probability. This is a necessary consequence of choosing a random mapping that does not depend on the given matrix **A**. The hope that oblivious subspace embeddings for low-dimensional target spaces exist, stems largely from the following theorem which was originally presented as a lemma by Johnson and Lindenstrauss in 1984.

Theorem 4 ([JL84, Lemma 1]). *Let $0 < \varepsilon < 1$ and $p \in \mathbb{N}$. For any p-element subset $V \subset \mathbb{R}^m$, there is a map $f : \mathbb{R}^m \to \mathbb{R}^k$ for some $k = O\left(\varepsilon^{-2} \ln p\right)$, such that*

$$(1-\varepsilon)\left\|\mathbf{u}-\mathbf{v}\right\|_2 \leq \left\|f(\mathbf{u})-f(\mathbf{v})\right\|_2 \leq (1+\varepsilon)\left\|\mathbf{u}-\mathbf{v}\right\|_2 \qquad \textit{for all } \mathbf{u}, \mathbf{v} \in V.$$

We will return to the proof of the theorem after showing its significance in establishing oblivious subspace embeddings. [LN16] prove that the lower bound for k matches the upper bound for almost the full range of ε. Although the asymptotic expression for the dimension of the target space gives a good indication of the dependence on ε and p, it might be necessary to know more about the suppressed constants of the expression in applications. [DG03] give one of the best bounds for the required target space, stating that $k \geq 4\left(\varepsilon^2/2+\varepsilon^3/3\right)^{-1} \ln p$. This gives a randomised map satisfying the theorem, with probability at least $1/p$. If we wish to guarantee a specific failure probability δ, following the same approach as in [DG03] results in

$$k \geq 2\left(\varepsilon^2/2+\varepsilon^3/3\right)^{-1} \ln\left(\left(p^2-p\right)/\delta\right). \tag{2.1}$$

From the proofs in for example [DG03; Sar06; Mat08], it turns out that the map f in Theorem 4 can in fact be assumed linear, in which case it corresponds to a linear projection of V onto a subspace of $\mathbb{R}^k$ such that all distances are approximately preserved. With the aim of utilising the result to obtain oblivious subspace embeddings, we relax the property of the theorem to be satisfied with a certain probability and present a natural definition.

Definition 4 (Johnson–Lindenstrauss transform). A distribution $\mathbf{\Pi}$ over $k \times m$ matrices forms an (ε, δ, p) Johnson–Lindenstrauss transform if the following holds: for any p-element subset $V \subset \mathbb{R}^m$, with probability at least $1-\delta$, a matrix $\mathbf{S}$ drawn from $\mathbf{\Pi}$ satisfies

$$(1-\varepsilon)\|\mathbf{u}-\mathbf{v}\|_2 \leq \|\mathbf{S}(\mathbf{u}-\mathbf{v})\|_2 \leq (1+\varepsilon)\|\mathbf{u}-\mathbf{v}\|_2 \qquad \text{for all } \mathbf{u}, \mathbf{v} \in V.$$

The following theorem now states that a Johnson–Lindenstrauss transform indeed gives rise to an oblivious subspace embedding.

Theorem 5. *If $\mathbf{\Pi}$ is an $\left(\frac{\varepsilon}{4}, \delta, 5^n\right)$ Johnson–Lindenstrauss transform, then $\mathbf{\Pi}$ is an (ε, δ, n) oblivious subspace embedding.*

Proof. See Appendix A.2. □

The key strength of the above result is extending the Johnson–Lindenstrauss property for a finite subset of elements to an entire subspace. Theorem 5 motivates the interest in the Johnson–Lindenstrauss lemma for sketching purposes, however, taking this approach just shifts the problem of finding an oblivious subspace embedding to finding a Johnson–Lindenstrauss transform.

Since the publication of [JL84], many different proofs of Theorem 4 have been presented, aiming to simplify existing proofs, provide lower bounds on the dimensions of the target space or to improve the computational aspects of the linear mapping. As remarked by Matoušek in [Mat08], most of these, if not all, proceed in the same manner; given p and m as in Theorem 4, one seeks to find a distribution $\mathbf{\Pi}$ on all linear maps $\mathbb{R}^m \to \mathbb{R}^k$ for suitable k, such that for every $\mathbf{x} \in \mathbb{R}^m$

$$\Pr_{\mathbf{S}\sim\mathbf{\Pi}}\Big[(1-\varepsilon)\|\mathbf{x}\|_2 \leq \|\mathbf{S}\mathbf{x}\|_2 \leq (1+\varepsilon)\|\mathbf{x}\|_2\Big] \geq 1 - \frac{1}{p^2}. \tag{2.2}$$

This property is known as the Random Projection Lemma, and Theorem 4 follows directly from this by considering the pairwise distances of points in the given p-element subset V. The probability that each pairwise distance is distorted more than ε is at most $\frac{1}{p^2}$, and hence the probability that any of the $\binom{p}{2}$ pairwise distances are distorted more than $(1\pm\varepsilon)$ is less than $\binom{p}{2}/p^2 = \frac{1}{2}\big(1-\frac{1}{p}\big)$. Thus the chosen projection succeeds with probability at least $\frac{1}{2}$.

In [IN07] and [DG03], it is shown that matrices whose entries are independent random variables drawn from the standard normal distribution $\mathcal{N}(0,1)$ satisfy the Random Projection Lemma. Utilising Theorem 4 and Theorem 5, this gives us our first oblivious subspace embedding.

Theorem 6 (Gaussian sketching matrix)**.** *Let $0 < \varepsilon, \delta < 1$ and let $\mathbf{\Pi}_{\mathrm{G}}$ be a distribution on $k \times m$ matrices $\mathbf{S} = \frac{1}{\sqrt{k}}\mathbf{G}$ where the entries of $\mathbf{G}$ are independent standard normally distributed variables. If*

$$k \geq 64\left(\varepsilon^2 - \varepsilon^3/6\right)^{-1}\left(\ln\left(5^{2n} - 5^n\right) + \ln\left(1/\delta\right)\right),$$

then $\mathbf{\Pi}_{\mathrm{G}}$ is an (ε, δ, n) oblivious subspace embedding.

The dependence on ε in the above bound for the number of rows can be clarified by noting that it is sufficient to take $k \geq c_1\varepsilon^{-2}\left(c_2 n + \ln\left(1/\delta\right)\right)$, for constants $c_1 \geq 64\cdot 6/5$ and $c_2 \geq 2\ln 5$. The exact expression in the theorem comes from setting $p = 5^n$ and replacing ε by $\varepsilon/4$ in Equation (2.1).

The Gaussian sketching matrix provides, with probability $1-\delta$, a subspace embedding that is very easy to implement with efficient random number generators available on most platforms. However, the projection matrix is dense and computing the projection of a vector of length m takes $O\left(km\right)$ time. In order to reduce computation time, one could hope to find subspace embedding matrices with a lower number of rows. It turns out, however, that the number of rows as given in Theorem 6 is practically tight [Alo03]. Faced with this, a natural alternative is attempting to construct sparse projection matrices that would lead to faster projection through faster matrix multiplication.

Achlioptas [Ach03] showed that the standard normal distribution can be replaced by two simpler distributions, where the entries of the matrix attain values from the set $\{-1, 0, +1\}$. The first of these is the Rademacher distribution, where the values attained are $+1$ and -1, each with probability $\frac{1}{2}$. This distribution allows for faster matrix multiplication than the normal distribution method above since the projection involves only addition and subtraction of entries and not multiplication. The second distribution gives a further speedup in computation, since the values $+1$, -1 and 0 are assigned probabilities $\frac{1}{6}$, $\frac{1}{6}$ and $\frac{2}{3}$, respectively, resulting in a sparse sketching matrix.

Unfortunately, it is not possible to improve significantly on this sparsity level for Johnson–Lindenstrauss transforms since sparse matrices will often distort sparse vectors [AC06]. This has led to focus on structured matrices that can be applied quickly to arbitrary vectors, an example of which is the centre of attention in the following section.

2.1.3 Subsampled randomised Hadamard transform

In 2006, Ailon and Chazelle [AC06] introduced the Fast Johnson–Lindenstrauss Transform, a random projection matrix that could be applied efficiently to any input vector. Their idea was to exploit the Uncertainty Principles known from quantum mechanics and later applied to signal processing, stating that a signal cannot be sharply localised in both the time and frequency domain (see for example [Kre78, Theorem 11.2-2], or more explicitly [Ste13, Chapter 2]). In the context of this report, this translates to the property that if some vector $\mathbf{x}$ is sparse, then the Fourier transform of $\mathbf{x}$ cannot be too sparse [Woo14, p. 16]. To ensure that a dense vector is not sparsified by this procedure, the Fourier transform is randomised by a diagonal matrix with diagonal entries drawn uniformly from the Rademacher distribution. The randomised Fourier transform thereby acts as a preconditioner of the input vector allowing us to apply a sparse projection without distorting the distances in the low-dimensional target space.

The Walsh-Hadamard transform is a class of generalised Fourier transforms and is often preferred due to its computational simplicity [AC06]. The preconditioning step is therefore often referred to as a randomised Hadamard transform. The Hadamard transform of an m-length vector can be viewed as multiplication with the $m \times m$ Hadamard matrix, which is defined recursively as

$$\mathbf{H}_m = \begin{bmatrix} \mathbf{H}_{m/2} & \mathbf{H}_{m/2} \\ \mathbf{H}_{m/2} & -\mathbf{H}_{m/2} \end{bmatrix} \qquad \text{where} \quad \mathbf{H}_1 = 1.$$

In this definition, it is clear that m must be a power of 2, however various constructions exist for other values of m (in fact, an open mathematical problem is the Hadamard conjecture, also known as Paley's conjecture, that Hadamard matrices of order $4k$ exist for all positive integers k. See [H+78] for a discussion of this). In practice, the considered vector is padded with zeros to a suitable size before the Hadamard transform is applied. This is for example the case in MATLAB's `fwht` implementation. One of the major advantages of a Fourier-type preconditioner is that it need not be implemented as a matrix multiplication which for an m-length vector would have a computational cost of $O(m^2)$. Instead, it can be applied in $O(m \ln m)$ time by a Fast Fourier Transform algorithm.

Since its introduction, several different versions of the Fast Johnson–Lindenstrauss transform have been considered. They are all based on the product of three matrices, such that the transform $\mathbf{\Phi}\colon \mathbb{R}^m \to \mathbb{R}^k$ can be written as $\mathbf{\Phi} = \mathbf{PHD}$ where

- **P** is a scaled $k \times m$ projection matrix,
- **H** is the $m \times m$ normalised Hadamard matrix, such that $\mathbf{H} = \frac{1}{\sqrt{m}}\mathbf{H}_m$,
- **D** is an $m \times m$ diagonal matrix with diagonal entries drawn independently from the Rademacher distribution.

The differences lie entirely in the projection step. Ailon and Chazelle [AC06] consider projection matrices where the entries are zero, with probability $1 - q$, and otherwise drawn from the normal distribution $\mathcal{N}\left(0, q^{-1}\right)$, with probability q, where $0 < q \leq 1$ is a sparsity parameter. Just like Achlioptas showed that the Gaussian subspace embedding could be replaced by a random sign matrix as mentioned in Section 2.1.2, Matousek showed in [Mat08] that the nonzero entries of Ailon and Chazelle's projection matrix could be drawn from the Rademacher distribution.

The next simplification was that the projection matrix **P** could in fact be a row-sampling matrix, however, it must be based on sampling without replacement. This corresponds to the restriction that each column can only contain a single nonzero entry. The embedding is then basically a random selection of rows from the preconditioned input $\mathbf{HDx}$. This approach was used in both [Dri+11] and [NDT09], and has since been dubbed the subsampled randomised Hadamard transform or SRHT.

Definition 5 (SRHT)**.** A $k \times m$ subsampled randomised Hadamard transform matrix is a matrix $\mathbf{S} = \mathbf{PHD}$, where $\mathbf{P} \in \mathbb{R}^{k \times m}$ is a matrix representing uniform sampling without replacement, and $\mathbf{H}$ and $\mathbf{D}$ are as defined above.

An intuitive understanding of why preconditioning the input with a Hadamard transform actually works can be attained through consideration of the leverage scores. As described in Section 2.1.1 and formally defined in Definition 2, the leverage scores of a matrix are a measurement of the importance of each row. Applying the Hadamard transform uniformises these leverage scores and therefore allows us to sample uniformly. The following lemma gives a precise bound for the leverage scores of preconditioned orthonormal matrices.

Lemma 7 ([Tro11, Lemma 3.3])**.** *Let* $\mathbf{U} \in \mathbb{R}^{m \times n}$ *be an orthonormal matrix, and let the matrices* $\mathbf{H}$ *and* $\mathbf{D}$ *be as described above. Define*

$$\ell_{\max}(\delta) = \frac{1}{m}\left(\sqrt{n} + \sqrt{8 \ln(m/\delta)}\right)^2 .$$

Then $\mathbf{HDU}$ *is orthonormal and, with probability at least* $1 - \delta$, $\max_{i=1,\dots,m} \ell_i \leq \ell_{\max}(\delta)$.

Proof. See [Tro11, Section 3.2]. $\square$

Remark. We remark that this is just one of many bounds on the leverage scores for orthonormal matrices. [Dri+11, Lemma 3] show the bound $\frac{1}{m}(2n \ln(2mn/\delta))$ which is subsequently also used in [Dri+12]. The bound given in Lemma 7 is the best bound that we have come across and is also the bound used implicitly in [Woo14, Theorem 2.4].

The bound on the leverage scores and the fact that $\mathbf{HDU}$ is orthonormal, allows us to use Theorem 3 with uniform probabilities $q_i = \frac{1}{m}$. To obtain the desired result, with probability at least $1 - \delta$, we take $\ell_{\max}(\delta_1)$ from Lemma 7 with $\delta_1 = \frac{1}{3}\delta$, and set $\delta_2 = \frac{2}{3}\delta$ and $\beta = n/(m\ell_{\max}(\delta_1))$ in Theorem 3. The result then follows from the union bound.

Theorem 8 (SRHT sketching matrix). *Let $0 < \varepsilon, \delta < 1$ and let $\mathbf{\Pi}_{\mathrm{SRHT}}$ be the distribution on subsampled randomised Hadamard transform matrices of size $k \times m$. If*

$$k \geq \frac{8}{3} m \varepsilon^{-2} \ln\left(\frac{3n}{\delta}\right) \ell_{\max}\left(\frac{\delta}{3}\right),$$

where $\ell_{\max}(\delta)$ is as in Lemma 7, then $\mathbf{\Pi}_{\mathrm{SRHT}}$ is an (ε, δ, n) oblivious subspace embedding.

Up to scaling, this result is the same as in [Tro11; BG13; Woo14]. The proofs in these articles, as well as in [Dri+11, Lemma 4], all utilise a matrix Chernoff bound (see for example Lemma 16 in Appendix A.1) and proceed as in the proof of Theorem 3. Clearly, the dependency of k on the failure probability δ is slightly worse for the SRHT than for sampling with leverage scores, however, as hinted at earlier, this is because the uniform probabilities are only guaranteed to satisfy the requirements of Theorem 3 with a certain probability.

We note that the preconditioning of the input vectors using the Hadamard transformation allow us to bypass the approach from Section 2.1.2, where we show that a sketching matrix satisfies the Random Projection Lemma as in Equation (2.2), which is a sufficient condition for providing a subspace embedding. However, the distribution on SRHT matrices does provide a Johnson–Lindenstrauss transform as shown in for example [AC06; Sar06; Tro11].

As mentioned previously, the subsampled randomised Hadamard transform can be applied to an $m \times n$ matrix in $O(mn \ln m)$ time (and possibly even in $O(mn \ln k)$ time, if we only consider the rows that are sampled [Mah+11, p. 15]). This is substantially lower than for a dense, non-structured matrix such as the Gaussian sketching matrix, but if the input matrix is sparse, we could hope to do even better. Ideally, we would like a dependency on the number of nonzeros in the matrix. This is the motivation behind the sparse projection matrix as introduced in the next section.

2.1.4 Sparse projection

The basis of the two previous sections has been to use Johnson–Lindenstrauss transforms and the Random Projection Lemma to ensure that the norms of all vectors in $\mathbb{R}^m$ are preserved. Compared to the definition of a subspace embedding, this is actually stronger than what is necessary, as we are only interested in preserving distances in the column space of $\mathbf{A}$ which is a subspace of dimension n [Woo14, p. 19]. This is a weaker restriction and opens op for new ways to construct oblivious subspace embeddings. Instead of seeking Johnson–Lindenstrauss transforms, we will instead require that the Johnson–Lindenstrauss moment property is satisfied. We will explain the significance of this after a formal definition.

Definition 6 (JL moment property)**.** A distribution $\mathbf{\Pi}$ of $k \times m$ matrices has the (ε, δ, l) Johnson–Lindenstrauss (JL) moment property if for all $\mathbf{x} \in \mathbb{R}^m$ with $\|\mathbf{x}\|_2 = 1$, it holds that

$$\mathop{\mathbb{E}}_{\mathbf{S}\sim\mathbf{\Pi}} \left[\left| \|\mathbf{S}\mathbf{x}\|_2^2 - 1 \right|^l \right] \leq \varepsilon^l \delta.$$

To see how the JL moment property relates to the definition of a subspace embedding, take for simplicity $l = 2$ and think of $\|\mathbf{S}\mathbf{x}\|_2^2$ as a random variable with mean $\|\mathbf{x}\|_2 = 1$. Then $\mathbb{E}\left[\left(\|\mathbf{S}\mathbf{x}\|_2^2 - 1 \right)^2 \right]$ is by definition the variance of $\|\mathbf{S}\mathbf{x}\|_2^2$, which by the JL moment property is bounded above by a small constant. If this constant is small enough, we know that with high probability $\|\mathbf{S}\mathbf{x}\|_2^2$ will be close to its mean $\|\mathbf{x}\|_2 = 1$. This is directly related to the requirement of having

$$(1 - \varepsilon) \|\mathbf{x}\|_2 \leq \|\mathbf{S}\mathbf{x}\|_2 \leq (1 + \varepsilon) \|\mathbf{x}\|_2$$

with high probability and hence the two definitions are closely linked.

The following theorem states that a distribution satisfying the JL moment property also provides an oblivious subspace embedding.

Theorem 9. *Let $\mathbf{\Pi}$ be a distribution of $k \times m$ matrices satisfying the $\left(\frac{\varepsilon}{3n}, \delta, 2\right)$ JL moment property. Then $\mathbf{\Pi}$ is an (ε, δ, n) oblivious subspace embedding.*

There are multiple ways of proving this theorem. In [CW13], the proof is based on controlling vectors with heavy coordinates and treating vectors with small and large entries separately. [Woo14, p. 21] presents an alternative proof which uses a lemma of approximate matrix multiplication as presented below.

Lemma 10 ([Woo14, Theorem 2.8])**.** *Let $\mathbf{\Pi}$ be a distribution on matrices $\mathbf{S}$ with m columns that satisfy the (ε, δ, l) JL moment property for some $l \geq 2$. Then for matrices $\mathbf{A}$ and $\mathbf{B}$, both with m rows,*

$$\Pr_{\mathbf{S}\sim\mathbf{\Pi}} \left[\left\| (\mathbf{S}\mathbf{A})^\top (\mathbf{S}\mathbf{B}) - \mathbf{A}^\top \mathbf{B} \right\|_F > 3\varepsilon \|\mathbf{A}\|_F \|\mathbf{B}\|_F \right] \leq \delta.$$

Proof. The proof can be found in Appendix A.3. □

We note that the result of Lemma 10 is very close to the equivalent definition of a subspace embedding presented in Theorem 1 (SE 2). What is left in the proof of Theorem 9 is to choose $\mathbf{A}$ and $\mathbf{B}$ appropriately.

Proof of Theorem 9. Setting $\mathbf{A} = \mathbf{B} = \mathbf{U_A}$ in Lemma 10, we immediately get

$$\Pr_{\mathbf{S}\sim\mathbf{\Pi}}\left[\left\|(\mathbf{SU_A})^\top(\mathbf{SU_A}) - \mathbf{I}\right\|_F > 3n\varepsilon\right] \leq \delta.$$

Since $\|\mathbf{C}\|_2 \leq \|\mathbf{C}\|_F$ for any matrix $\mathbf{C}$, then, with probability at least $1-\delta$,

$$\left\|(\mathbf{SU_A})^\top(\mathbf{SU_A}) - \mathbf{I}\right\|_2 < \varepsilon$$

for $\mathbf{S} \sim \mathbf{\Pi}$. Hence, $\mathbf{\Pi}$ is an (ε, δ, n) oblivious subspace embedding by Theorem 1 (SE 2). □

Our interest is therefore now in finding distributions satisfying the JL moment property. It turns out that the CountSketch matrix, known from the streaming literature [CCF02], satisfies this property with certain restrictions on the number of rows. The CountSketch matrix has a single nonzero entry per column, the value of which is a random variable drawn from the Rademacher distribution. For each column, the position of the nonzero value is chosen uniformly at random. In the context of sketching, [Woo14] calls this matrix a sparse embedding matrix, however, we will use the terminology sparse projection matrix. Its relevance in this context is summarised in the following theorem.

Theorem 11 ([Woo14, Theorem 2.9]). *Let $\mathbf{\Pi}_{\text{SPM}}$ be a distribution on $k \times m$ sparse projection matrices with at least $k = \frac{2}{\delta\varepsilon^2}$ rows. Then $\mathbf{\Pi}_{\text{SPM}}$ satisfies the $(\varepsilon, \delta, 2)$ JL moment property.*

Together, Theorem 9 and Theorem 11 yield a way of constructing an oblivious subspace embedding using sparse projection matrices.

Theorem 12 (Sparse projection sketching matrix). *Let $0 < \varepsilon, \delta < 1$ and let $\mathbf{\Pi}_{\text{SPM}}$ be the distribution on $k \times m$ sparse projection matrices. If*

$$k \geq 18n^2\varepsilon^{-2}\delta^{-1},$$

then $\mathbf{\Pi}_{\text{SPM}}$ is an (ε, δ, n) oblivious subspace embedding.

Proof. The proof is a natural consequence of applying Theorem 9 and 11. What is left to show is that $k \geq 18\frac{n^2}{\delta\varepsilon^2}$ rows in the sparse projection matrix is necessary. Observe that we need at least $\frac{2}{\delta\varepsilon^2}$ rows for $\mathbf{\Pi}_{\text{SPM}}$ to satisfy the (ε, δ, n) JL moment property. But for $\mathbf{\Pi}_{\text{SPM}}$ to also be an (ε, δ, n) oblivious subspace embedding, we need enough rows for $\mathbf{\Pi}_{\text{SPM}}$ to satisfy the $\left(\frac{\varepsilon}{3n}, \delta, 2\right)$ JL moment property. Thus the number of rows must at least be $2\delta^{-1}\left(\frac{\varepsilon}{3n}\right)^{-2} = 18\frac{n^2}{\delta\varepsilon^2}$. □

The matrices drawn from the oblivious subspace embedding constructed in this section present themselves as sparse alternatives to the projection methods previously mentioned. The benefit of using sparse sketching matrices is that they can be multiplied with an input matrix in time proportional to the sparsity of the input. However, it should also be noted that the dimensionality reduction of sparse projection matrices is often weaker than that of other alternatives.

This is in part due to the linear dependence on δ^{-1}. As shown in [Lia+14] and remarked in [Woo14], a logarithmic dependence in δ^{-1} can be achieved for the sparse projection matrix if one is willing to increase the computation time to $O(\mathrm{nnz}(\mathbf{A}) \ln(1/\delta))$. This is done by constructing $O(\ln(1/\delta))$ constant probability sparse embeddings for the input matrix, and selecting an embedding that succeeds, with probability at least $1-\delta$, in a cross-validation style procedure (see [Lia+14, Algorithm 5 and Theorem 13]). It is important to note, however, that the method is then no longer oblivious.

Sketching matrices similar to the sparse projection matrix, but with more than one nonzero entry per column as presented in [NN13], have been shown to need significantly fewer rows, most importantly avoiding the dependence on n^2. A representative result from this paper, which also appears as in [Woo14, Theorem 2.7], states that there exists a sparse $(1 \pm \varepsilon)$ subspace embedding for any fixed $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $n\varepsilon^{-2}\mathrm{poly}(\ln(n/(\delta\varepsilon)))$ rows and error probability δ. These methods are called Oblivious Sparse Norm-Approximating Projections or OSNAP for short. We refer to the literature for a deeper investigation of these.

2.2 Comparison of sketching matrices

The above sections introduce four different sketching matrices based on leverage score sampling, Gaussian projection, SRHT and sparse projection. To give a better understanding of the differences between these methods and the advantages each hold, we will in this section compare the number of rows required to obtain $(1 \pm \varepsilon)$ subspace embeddings, with probability $1-\delta$, and the time taken to construct each sketch.

In the previous sections, we have already obtained explicit lower bounds for the dimension of the sketch matrix for it to provide a desired subspace embedding. The time taken to construct each sketch is comprised of several method-dependent factors such as generating random numbers, approximating leverage scores, sampling rows, constructing matrices and computing the matrix product $\mathbf{SA}$. We will refer to the combination of all relevant time factors as the *sketch time*. For the oblivious subspace embedding methods, the sketch time will be dominated by the matrix multiplication, whether performed explicitly or by use of the Hadamard transform. For the approximate leverage score sampling method, the primary time factor is in approximating the leverage scores, as the actual sampling of the input matrix is not costly.

As a summary of the above, Table 2.1 shows the main properties of the four sketching methods. The rows required are stated in the listed theorems and the sketch time is as explained in the preceding sections.

The asymptotic expressions for the number of rows needed give a better understanding of the dependence on the relevant factors. Of particular interest is the dependence on the size of the input matrix. From Table 2.1 we see that only for the SRHT method does k depend on m, however, for all the methods, there is a significant dependence on n. Fixing all other parameters (ε, δ, β and m) to constants, the growth of k as a function of n for each of the sketching methods is visualised in Figure 2.1.

Since k controls the size of the sketching matrix, it is the main underlying factor

Table 2.1: A comparison of the four sketching methods discussed previously. The table shows the theorems stating that the matrices are subspace embeddings, the number of rows required for this to hold and the total sketch time, for $\mathbf{S} \in \mathbb{R}^{k\times m}$ and $\mathbf{A} \in \mathbb{R}^{m\times n}$ with full column rank.

Type	Main result	Rows required (exact and asymptotic)	Sketch time
Leverage score sampling	Theorem 3	$k \geq \frac{8n\ln(2n/\delta)}{3\beta\varepsilon^2}$ $k = O\left(\varepsilon^{-2} n \ln(n/\delta)\right)$	$O(\mathrm{nnz}(\mathbf{A}) \ln m)$ $+ \mathrm{poly}(n)$
Gaussian projection	Theorem 6	$k \geq \frac{64\left(\ln\left(5^{2n} - 5^n\right) + \ln(1/\delta)\right)}{\varepsilon^2 - \varepsilon^3/6}$ $k = O\left(\varepsilon^{-2}\left(n + \ln(1/\delta)\right)\right)$	$O(kmn)$
Subsampled randomised Hadamard transform	Theorem 8	$k \geq \frac{8\ln(3n/\delta)\left(\sqrt{n} + \sqrt{8\ln(3m/\delta)}\right)^2}{3\varepsilon^2}$ $k = O\left(\varepsilon^{-2}\ln(n/\delta)\left(\sqrt{n} + \sqrt{\ln(m/\delta)}\right)^2\right)$	$O(nm\ln m)$ or potentially $O(nm\ln k))$
Sparse projection	Theorem 12	$k \geq \frac{18n^2}{\delta\varepsilon^2}$ $k = O\left(\varepsilon^{-2}\delta^{-1}n^2\right)$	$O(\mathrm{nnz}(\mathbf{A}))$

of the computation time and storage connected with computing the sketched matrix. From Table 2.1, it is seen that for all four methods, k practically scales with ε^2 and thus setting ε constant has no significant influence on the relationship between methods. Sparse projection is the only method that scales linearly rather than logarithmically in δ^{-1}. However, the contribution of this is largely overshadowed by the factor of n^2. This justifies fixing the other parameters and considering the dependence of k on n.

Figure 2.1 shows that the number of rows needed for the sparse projection matrix grows much faster than for the other three methods, which is due to the linear dependence on n^2 rather than n or $n \ln n$. If the input matrix has 100 columns, the sparse projection matrix will require $1.8 \cdot 10^8$ rows to provide a subspace embedding with the specified values for δ and ε. Even if the failure probability and approximation parameter are relaxed to $\delta = \varepsilon = \frac{1}{2}$, the required number of rows still exceeds $1.44 \cdot 10^6$.

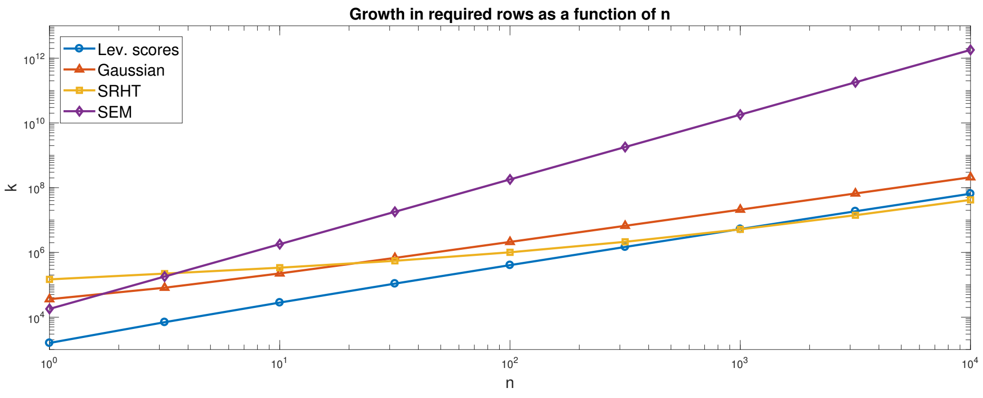


Figure 2.1: Comparison of the asymptotic growth of the number of rows for each sketching method as functions of n. All other parameters are constant with $\delta = 0.1$, $\varepsilon = 0.1$, $\beta = 0.5$ and $m = 10^6$.

In practice, there are numerous ways to improve the bound on the number of rows for the sparse projection matrix. One approach from [Woo14] is to compose the sparse projection matrix $\mathbf{S}$ with another sketching matrix $\widehat{\mathbf{S}}$ and computing $\widehat{\mathbf{S}}\mathbf{S}\mathbf{A}$. The matrix $\widehat{\mathbf{S}}$ can for example be chosen as a Gaussian or SRHT sketching matrix. Ideas from Section 2.1.4 to improve the linear dependence on δ^{-1} using the success probability boosting or to use other sparse sketching matrices with more than one nonzero per column could also be considered.

Figure 2.1 also shows how the SRHT sketching matrix initially requires a larger number of rows than the other methods as it is dominated at the offset by the term containing the large constant value of m. However, the growth then aligns with that of the leverage score sketching matrix, as is also evident from the expressions in Table 2.1. The growth of the number of rows for the Gaussian sketching matrix is slightly slower than that for the leverage score sketching matrix which is also to be expected. The difference between the two is due to the large constant factors appearing in the row bound for the Gaussian sketching matrix.

Despite Figure 2.1 serving as a useful visualisation of the growth in size of the different sketching matrices, it is important not to read too much into the exact values. Firstly, the fixed parameters are chosen as arbitrary, albeit reasonable, constants. Secondly, the expressions for the number of rows are all sufficient conditions for the sketching matrices to provide the desired subspace embeddings but better bounds may still exist. This is especially the case as soon as one has some knowledge of the input data and the bounds can be tailored to these cases. This might be information on the uniformity, sparsity or variations in the input. Lastly, these are all theoretical results and the behaviour of the sketching matrices in practice might be better in most cases. This will be explored in much further detail in the rest of this report.

Crucially, it is not only the size of the sketch matrix that is of importance. The structure and composition of each sketching method lead to vastly different computation times for sketching matrices as is also evident from Table 2.1. The sparse projection matrix may require a large number of rows but the sketch can be computed in input sparsity time whereas the Gaussian projection is a multiplication with a dense matrix and hence requires a lot more operations. The trade-off between size and time will play a key role in the practical investigation of the sketching methods.

CHAPTER 3

Sketching in practice

As presented in the previous chapter, a lot of theory concerning the approximation errors, performance guarantees and complexity results of sketching has been developed within the last 20 years. However, most of the analysis remains at a theoretical level and the actual performance and applicability of sketching in standard working environments can be difficult to understand and assess. In Section 2.2, we saw how the number of rows required for a sketching matrix to provide a $(1 \pm \varepsilon)$ subspace embedding for an $m \times n$ matrix grew as a function of n, but also noted that these are theoretical conditions and we could hope to achieve much better results in practice. The goal of this chapter is to clarify when and how sketching can be used in problems and algorithms and thus bridge the gap between theory and practice.

We do this by considering three stages of application. Firstly, we introduce basic building blocks that provide the foundations for most sketching applications. We then present three generic linear algebra problems where sketching can be used to gain a computational advantage. Finally, we consider three concrete applications from real-world problems that demonstrate how the described methods can be used to reduce computation time, lower storage costs and enhance interpretability of results, respectively.

In total, we consider nine different applications of sketching. In order to quantify the advantages gained in each application, every section includes a part called "experimental results". Within these, we describe the experimental setup and present results based on the corresponding algorithms. The included theorems are presented without proofs for simplicity but the relevant literature is always referenced. Dividing the applications in the three aforementioned stages serves to break up the use of sketching and provide a clear structure for when and how sketching can be used. A visualisation of the structure is shown in Figure 3.1.

The basic building blocks introduced in Section 3.1 consist of approximate matrix multiplication, computation of an orthonormalising transformation matrix and the construction of a low-dimensional approximate orthonormal basis. These are chosen

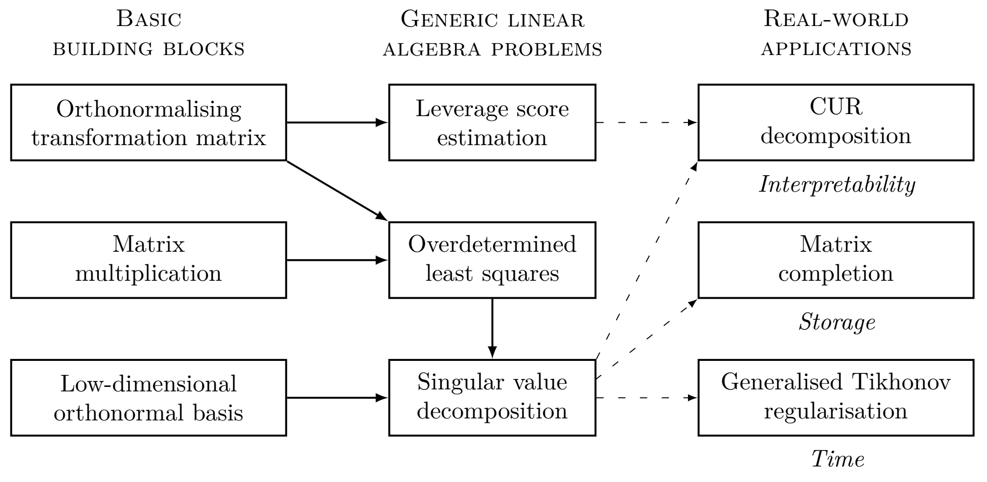

Basic building blocks
Generic linear algebra problems
Real-world applications
Orthonormalising transformation matrix
Matrix multiplication
Low-dimensional orthonormal basis
Leverage score estimation
Overdetermined least squares
Singular value decomposition
CUR decomposition
Interpretability
Matrix completion
Storage
Generalised Tikhonov regularisation
Time

Figure 3.1: Overview of stages and applications of Chapter 3. Arrows represent connections between problems.

for their simplicity and because most more complicated uses of sketching essentially boil down to exactly these three building blocks. Thus the problems are general, the use of sketching simple and the application areas vast.

Section 3.2 uses the basic building blocks in the presentation of three common linear algebra problems: leverage score estimation, overdetermined least squares and the singular value decomposition. Applications for solving these problems are useful on their own or as part of other more complicated algorithms, and also serve as examples and inspiration for how sketching can be implemented in standard linear algebra settings.

In Section 3.3, we show three concrete applications of sketching allowing for better intuition and interpretation of the obtained results and the overall merits of sketching. Matrix completion in recommender systems is an example where storage costs can be reduced significantly. The CUR decomposition is an alternative to other more standard matrix factorisations allowing for much greater interpretability of the data. Generalised Tikhonov regularisation is a setting in which sketching can be used to simplify a computationally demanding optimisation problem and thereby decrease the time costs of the solution.

Experimental setup. To test our sketching algorithms for various problems, we use a number of different test matrices. An important property of a matrix is its condition number given by $\kappa = \frac{\sigma_{\max}}{\sigma_{\min}}$. The following list presents the three main test matrices used in our experiments.

`srand` is a structured random matrix where a matrix of uniform random values is initialised after which each row is randomly scaled. In MATLAB this is done by

```
>> A = rand(m,n).*rand(m,1);
```

`cond10` is a matrix with condition number $\kappa = 10^{10}$, constructed in MATLAB by

```
>> A = rand(m,n); [U,~,V] = svd(A,'econ');
>> S = diag(logspace(0,-10,min(m,n))); A = U*S*V';
```

`polydecay` is a matrix with an exponentially decaying spectrum and with condition number $\kappa = \min(m,n)$. We initialise `A` and perform the SVD as for the `cond10` test matrix, and then do the following:

```
>> S = diag((1:min(m,n)).^(-1)); A = U*S*V';
```

For each application, we use problem dependent error measures in order to test approximation quality. Furthermore, we measure the relative speed of the approximation method in comparison to the computation of the exact or best possible deterministic solution method and state this as the *gain factor*. We use a common experimental setup across all applications: randomly generate a problem with specific structure, draw a sketching matrix of a certain kind, perform the problem dependent calculations and report the results. As described in Chapter 2, the sketching procedure can often be implemented efficiently without direct matrix multiplication. The drawing of the explicit sketching matrix and the notation of a matrix product is used throughout this chapter to highlight the underlying idea.

To compensate for the failure probability of the sketching matrix, we draw $n_{\mathbf{S}}$ instances of $\mathbf{S}$ after which the median of the results is saved. We compensate for the randomness in the problem type by running the experiment for $n_{\mathbf{A}}$ instances of $\mathbf{A}$ and reporting the mean of the results. In each considered application, the different sketching methods are tested on the same $n_{\mathbf{A}}$ instances of the input matrix. Unless otherwise specified, we set $n_{\mathbf{S}} = 5$ and $n_{\mathbf{A}} = 3$.

All experiments are run in MATLAB R2017a and limited to a single computational thread by use of the command `maxNumCompThreads(1)` before each experiment. This restricts the implicit parallelisation performed by MATLAB, easing the comparison of algorithms.

3.1 Basic building blocks

The problems considered in this section are matrix multiplication, finding an orthonormalising transformation matrix and computing a low-dimensional basis. For large problems, even these basic operations can act as computational bottlenecks. We show how sketching can be used to bypass the complexity of these operations and generate approximate solutions. Our focus is on portraying the simplicity of the sketching methods and explicitly stating corresponding algorithms that can be used in downstream applications.

3.1.1 Matrix multiplication

For large matrices, even the basic operation of matrix multiplication can be a computational challenge. This has led to the problem of approximate matrix multiplication where, given two large matrices $\mathbf{A} \in \mathbb{R}^{m \times n_1}$ and $\mathbf{B} \in \mathbb{R}^{m \times n_2}$, we seek to find a matrix $\mathbf{C} \in \mathbb{R}^{n_1 \times n_2}$ such that the quantity $\left\|\mathbf{C} - \mathbf{A}^\top \mathbf{B}\right\|$ is small for some chosen matrix norm. Our primary focus here is on cases where we wish to trade accuracy for speed but sketching also plays a role in memory and band-limited problems, where the matrices to be multiplied are too large to store locally or transfer from external storage.

We have already seen how sketching matrices satisfying the JL moment property can be used for approximate matrix multiplication in Lemma 10. The following theorem generalises the use of subspace embedding matrices in approximate matrix multiplication.

Theorem 13 ([CNW15, Lemma 1]). *Given $0 < \varepsilon < 1$, $\mathbf{A} \in \mathbb{R}^{m \times n_1}$ and $\mathbf{B} \in \mathbb{R}^{m \times n_2}$, let $\mathbf{S} \in \mathbb{R}^{k \times m}$ be a $(1 \pm \varepsilon)$ subspace embedding for the space spanned by the columns of $\mathbf{A}$ and $\mathbf{B}$. Then*

$$\left\|(\mathbf{S}\mathbf{A})^\top (\mathbf{S}\mathbf{B}) - \mathbf{A}^\top \mathbf{B}\right\|_2 \leq \varepsilon \left\|\mathbf{A}\right\|_2 \left\|\mathbf{B}\right\|_2 .$$

Theorem 13 gives a guarantee of an ε approximation in spectral norm when the number of rows k is chosen such that $\mathbf{S}$ is a $(1 \pm \varepsilon)$ subspace embedding. Clearly, if $k < m$, the product $(\mathbf{S}\mathbf{A})^\top (\mathbf{S}\mathbf{B})$ is faster to compute than its exact counterpart $\mathbf{A}^\top \mathbf{B}$. The theorem admits a straight-forward algorithm for approximate matrix multiplication using randomisation as presented in Algorithm 1.

The full matrix product $\mathbf{A}^\top \mathbf{B}$ can be computed in $O(mn_1n_2)$ time. The overall complexity for the matrix multiplication algorithm consists of two factors, namely the sketch time and the multiplication of the two sketches, which takes $O(kn_1n_2)$

Algorithm 1 Approximate matrix multiplication

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n_1}$, $\mathbf{B} \in \mathbb{R}^{m\times n_2}$ and sketch parameter $k$
Output: Approximate matrix product $\mathbf{C} \in \mathbb{R}^{n_1\times n_2}$
1: Construct sketching matrix $\mathbf{S} \in \mathbb{R}^{k\times m}$
2: Sketch input matrices $\mathbf{A}_{\text{sketch}} = \mathbf{S}\cdot\mathbf{A}$ and $\mathbf{B}_{\text{sketch}} = \mathbf{S}\cdot\mathbf{B}$
3: Calculate $\mathbf{C} = (\mathbf{A}_{\text{sketch}})^\top (\mathbf{B}_{\text{sketch}})$
```

time. The sketch time depends heavily on the choice of sketching matrix and is $O\left(km\left(n_1+n_2\right)\right)$ for the Gaussian projection, $O\left(m\ln m\left(n_1+n_2\right)\right)$ for leverage score sampling and SRHT, and $O\left(m\left(n_1+n_2\right)\right)$ for the sparse projection as shown in Table 2.1. For the sketching method to be faster than the direct calculation, we therefore need the savings in multiplication time, $O\left((m-k)n_1n_2\right)$, to be larger than the sketch time. This will be the case when k is sufficiently smaller than m.

Experimental results. We aim to investigate the quality of the approximation $\mathbf{C} \approx \mathbf{A}^\top\mathbf{B}$ with $\mathbf{C}$ being the output from Algorithm 1. We define the following residual matrix

$$\mathbf{C}_{\text{res}}^{(\xi)} = \frac{\mathbf{C}-\mathbf{A}^\top\mathbf{B}}{\|\mathbf{A}\|_\xi\,\|\mathbf{B}\|_\xi} = \frac{(\mathbf{SA})^\top(\mathbf{SB})-\mathbf{A}^\top\mathbf{B}}{\|\mathbf{A}\|_\xi\,\|\mathbf{B}\|_\xi} \qquad \text{for } \xi \in \{2, F\}. \tag{3.1}$$

Taking the spectral norm of the residual matrix, $\left\|\mathbf{C}_{\text{res}}^{(2)}\right\|_2$, we obtain an error measure similar to that of Theorem 13 and we will therefore be able to test the relation of the theorem even when the sketching matrix does not necessarily satisfy the property of a $(1\pm\varepsilon)$ subspace embedding. On the other hand, setting $\xi = F$ and considering the Frobenius norm error, $\left\|\mathbf{C}_{\text{res}}^{(F)}\right\|_F$, gives a somewhat better understanding of the entry-wise differences between $\mathbf{C}$ and $\mathbf{A}^\top\mathbf{B}$. Table 3.1 presents results for which $\mathbf{A}$ and $\mathbf{B}$ are chosen to have the same size and type.

Regarding approximation quality, the table shows that the methods perform on par for both problem types, with the exception of leverage score sampling performing significantly better for the `srand` test matrix. An explanation of this might be the construction of the `srand` matrices where the rows are randomly scaled and thus creating variation in the leverage scores. It might also be due to the observation from Section 2.2 that leverage score sampling in many cases is the sketching method requiring the smallest amount of rows in order to provide a subspace embedding. We note that the spectral norm error $\left\|\mathbf{C}_{\text{res}}^{(2)}\right\|_2$ is generally quite large for the ill-

Table 3.1: Approximate matrix multiplication using different sketching matrices for two test matrix types. The matrices $\mathbf{A}$ and $\mathbf{B}$ have $m = 2^{15}$ rows and $n = 2^{10}$ columns and the sketch size parameter is $k = 2n$. The error terms are based on the residual matrix $\mathbf{C}_{\text{res}}^{(\xi)}$ of Equation (3.1).

	`srand`		`cond10`		
Sketching method	$\lVert\mathbf{C}_{\text{res}}^{(2)}\rVert_2$	$\lVert\mathbf{C}_{\text{res}}^{(F)}\rVert_F$	$\lVert\mathbf{C}_{\text{res}}^{(2)}\rVert_2$	$\lVert\mathbf{C}_{\text{res}}^{(F)}\rVert_F$	Gain factor
Leverage score sampling	0.0136	0.0125	0.1584	0.0221	0.27
Gaussian projection	0.0288	0.0228	0.1593	0.0223	0.19
SRHT	0.0245	0.0197	0.1530	0.0213	2.4
Sparse projection	0.0245	0.0199	0.1568	0.0220	7.8

conditioned problem, which is due to the high condition number of the input matrices $\mathbf{A}$ and $\mathbf{B}$, making the product very sensitive to errors.

Interestingly, $k = 2n$ is sufficient for the sketching methods to achieve approximate solutions with a relative error close to 2% as measured by the Frobenius norm error. For the `srand` test matrix, we achieve spectral norm errors below 3%. This is significantly lower than the theoretical expressions of Table 2.1 coupled with Theorem 13, according to which we need 2 billion rows to ensure an approximation error of at most 3%, with probability 0.9, using a sparse projection matrix for the considered problem. The relative spectral norm error is much higher for the `cond10` test matrix, with all the sketching methods yielding errors around 15%. Figure 3.2 shows $\lVert\mathbf{C}_{\text{res}}^{(2)}\rVert_2$ as a function of k on the `cond10` problem with both $\mathbf{A}$ and $\mathbf{B}$ of size $m = 2^{15}$ and $n = 2^{10}$. It is seen that the spectral norm error decreases to around 5%, still using a reasonably sized proportionality constant. In general, we find that choosing k proportional to n often works well.

The potential time advantages of Algorithm 1 are also shown in Table 3.1. Using the sparse projection method, an approximation is almost 8 times faster to compute than the exact matrix product. However, the leverage score sampling and Gaussian projection methods are significantly slower and are therefore not very useful for this problem. For the Gaussian projection, this is due to the large matrix-matrix products involved in the sketching of the input matrices. The leverage scores have to reflect the properties of matrices $\mathbf{A}$ and $\mathbf{B}$ simultaneously and it is therefore necessary to compute the leverage scores based on the matrix $\begin{bmatrix}\mathbf{A} & \mathbf{B}\end{bmatrix}$. Not only does this affect the inner sketching parameters of the leverage score estimation algorithm, the actual concatenation of the matrices in MATLAB also takes time. For the approximate matrix

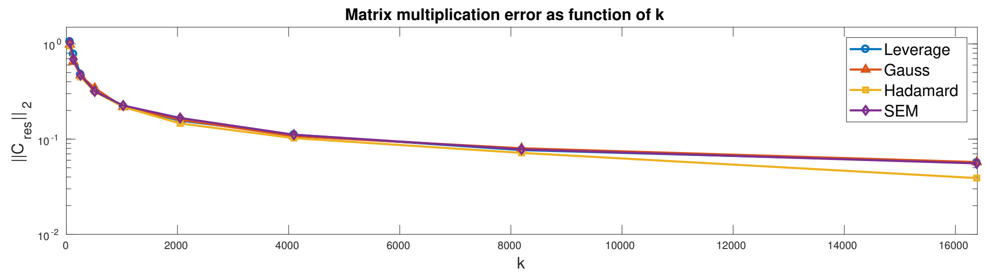


Figure 3.2: Convergence of $\left\|\mathbf{C}_{\text{res}}^{(2)}\right\|_2$ as a function of the sketch size parameter k for the `cond10` matrix type of size $m = 2^{15}$ and $n = 2^{10}$.

multiplication problem, it is common to use importance sampling probabilities based on the Euclidean norms of the rows of each matrix [DKM06; CNW15]. These are much faster to compute than the leverage scores, leading to a considerably faster sketching method for this problem, and could be considered as an alternative method.

Clearly the applicability of Algorithm 1 depends on the error tolerance in the given problem. The errors in Table 3.1 might be tolerable in some settings but unreasonable in others. To get a better understanding of the error magnitudes, we consider the estimation of a sample covariance matrix, which is calculated as the normalised matrix product $\frac{1}{m-1}\left(\mathbf{X} - \bar{\mathbf{X}}\right)\left(\mathbf{X} - \bar{\mathbf{X}}\right)^\top$, where each column of $\mathbf{X} \in \mathbb{R}^{n\times m}$ represents an n-dimensional data point and $\bar{\mathbf{X}}$ consists of the sample means.

We firstly generate an $n \times n$ target covariance matrix $\boldsymbol{\Sigma}$ using a locally periodic kernel function defined as the product of the squared exponential and periodic kernel functions[1]. We choose the parameters of the kernel functions as $\alpha = 1$ and $T_{\text{period}} = \frac{1}{5}$, and draw $m = 2^{15}$ samples from the normal distribution $\mathcal{N}\left(\mathbf{0}, \boldsymbol{\Sigma}\right)$ to form $\mathbf{X}$. Using the sketch parameter $k = 2n$ with the sparse projection method, we obtain the results shown in Figure 3.3. The gain factor was approximately 8, as was also the case for the corresponding result in Table 3.1.

Figure 3.3 shows that with spectral norm error of 0.03, the approximate matrix product captures the essential parts of the sample covariance matrix even though the residual matrix includes non-random artefacts. In applications where the exact

[1]The squared exponential or Gaussian kernel, k_{SE}, and the periodic kernel, k_{PE}, are defined as

$$k_{\text{SE}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \exp\left(-\frac{\left(\mathbf{x}_i - \mathbf{x}_j\right)^2}{2\alpha^2}\right) \quad \text{and} \quad k_{\text{PE}}\left(\mathbf{x}_i, \mathbf{x}_j\right) = \exp\left(-\frac{2}{\alpha^2}\sin^2\left(\frac{\pi\left|\mathbf{x}_i - \mathbf{x}_j\right|}{T_{\text{period}}}\right)\right).$$

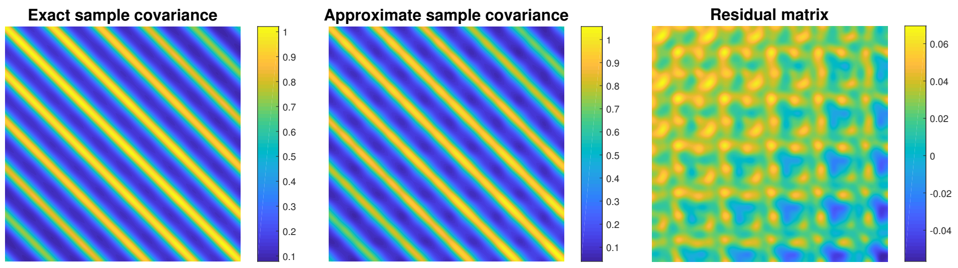


Figure 3.3: Visual representation of the approximation of a sample covariance matrix using Algorithm 1. The sketch size parameter was $k = 2n$, and the data matrix $\mathbf{X}$ was of size $m = 2^{15}$ and $n = 2^{10}$, with each column drawn from a normal distribution with a covariance matrix based on a locally periodic kernel function.

product is not necessary, such an approximation could be very useful. We will see an example of this in Section 3.3.3.

3.1.2 Orthonormalising transformation matrix

Orthonormal matrices are of great interest in numerical linear algebra, particularly for their numerical stability properties: since all eigenvalues of an orthonormal matrix are of magnitude 1, so is the condition number. Given an $m \times n$ matrix $\mathbf{A}$, we can compute a QR decomposition $\mathbf{A} = \mathbf{QR}$, where $\mathbf{Q} \in \mathbb{R}^{m\times n}$ is an orthonormal matrix and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix. This can be done in $O\left(mn^2\right)$ time [GV12], which for large matrices can be prohibitively slow.

As the columns of $\mathbf{Q}$ span the column space of $\mathbf{A}$, $\mathbf{Q}$ can be viewed as an orthonormal basis for the range of $\mathbf{A}$. Correspondingly, $\mathbf{R}$ can be viewed as a matrix whose inverse encodes a change of basis for $\mathbf{A}$ to the orthonormal basis given by $\mathbf{Q}$. Such bases have already played a role in both the properties of ℓ_2-subspace embeddings in Theorem 1 and in the definition of leverage scores, a connection which we will return to later. Indeed, since subspace embeddings approximately preserve distances and orthogonality in the column space of $\mathbf{A}$, we might hope that for an ℓ_2-subspace embedding $\mathbf{S}$, the sketch $\mathbf{SA}$ can be used to calculate an approximate orthonormal basis for the original matrix $\mathbf{A}$.

Simply finding an orthonormal basis for $\mathbf{SA}$ will not work, as the columns of $\mathbf{A}$ are situated in a very different, higher dimensional space to that spanned by the columns of $\mathbf{SA}$. One idea is to view the orthonormal basis as the product $\mathbf{Q} = \mathbf{AR}^{-1}$. Calculating the QR decomposition $\mathbf{SA} = \mathbf{Q_S R_S}$, we use the change of basis matrix $\mathbf{R_S}$ as a substitute for $\mathbf{R}$. This will enable us to approximate an orthonormal basis by $\mathbf{Q} \approx \mathbf{AR_S^{-1}}$.

To give some insight into why $\mathbf{AR_S^{-1}}$ can provide an adequate approximation of an orthonormal basis, we consider the corresponding singular values. For arbitrary $\mathbf{x} \in \mathbb{R}^n$, the property of a $(1 \pm \varepsilon)$ subspace embedding $\mathbf{S}$ implies that

$$(1-\varepsilon)\left\|\mathbf{SAR_S^{-1}x}\right\|_2 \leq \left\|\mathbf{AR_S^{-1}x}\right\|_2 \leq (1+\varepsilon)\left\|\mathbf{SAR_S^{-1}x}\right\|_2,$$

and, using the QR decomposition of $\mathbf{SA}$ and orthonormality of $\mathbf{Q_S}$, we get

$$\left\|\mathbf{SAR_S^{-1}x}\right\|_2 = \|\mathbf{Q_S x}\|_2 = \|\mathbf{x}\|_2 .$$

Combining these results and considering the special case where $\mathbf{x}$ is a left singular vector of $\mathbf{AR_S^{-1}}$, we see that the singular values of $\mathbf{AR_S^{-1}}$ must lie in the interval $[1-\varepsilon, 1+\varepsilon]$.

In summary, our aim is to approximate an orthonormalising transformation matrix using sketching, since an approximation of $\mathbf{Q}$ will then follow by standard matrix calculations. An algorithm for obtaining $\mathbf{R_S}$ is shown in Algorithm 2.

Algorithm 2 Approximation of orthonormal transformation matrix

```
Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$ and sketch parameter $k$
Output: $\mathbf{R_S} \in \mathbb{R}^{n \times n}$
1: Construct sketching matrix $\mathbf{S} \in \mathbb{R}^{k \times m}$
2: Sketch input matrix $\mathbf{A}_{\text{sketch}} = \mathbf{S} \cdot \mathbf{A}$
3: Calculate QR decomposition $\mathbf{A}_{\text{sketch}} = \mathbf{Q_S R_S}$
```

Computationally, we first pay the cost of creating the sketch $\mathbf{SA}$ which depends on the chosen sketching matrix $\mathbf{S} \in \mathbb{R}^{k \times m}$ as shown in Table 2.1. We then perform a QR decomposition of $\mathbf{SA}$ in $O\left(kn^2\right)$ time.

If the explicit orthonormal basis matrix is desired, calculating the inverse of $\mathbf{R_S}$ can be done in $O\left(n^3\right)$ operations and the actual matrix product $\mathbf{AR_S^{-1}}$ takes $O\left(mn^2\right)$ time. However, this is on par with the computational cost of QR decomposing the entire matrix $\mathbf{A}$. One could use sketching to reduce the number of columns in $\mathbf{R_S^{-1}}$, which is the approach used in [Woo14, Theorem 2.13] and [CW13, Theorem 29] in connection with approximating leverage scores as will be discussed later. In many cases, however, we will not need to perform the actual matrix product, e.g. when only the product of $\mathbf{AR}_\mathbf{S}^{-1}$ with a vector is required. We will see such an application when we turn our attention to the solution of the least squares problem and the use of preconditioners.

Experimental results. The experiments of this section aim to test the quality of approximation for an orthonormalising transformation matrix. The orthonormalising property of the transformation matrix $\mathbf{R}$ is of particular interest and thus our tests will focus on the orthonormality of $\mathbf{AR}_\mathbf{S}^{-1}$. For an orthonormal matrix $\mathbf{U}$, it holds by definition that $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$ and all singular values are 1, which suggests the following error measure

$$\frac{\left\|\left(\mathbf{AR}_\mathbf{S}^{-1}\right)^\top \mathbf{AR}_\mathbf{S}^{-1} - \mathbf{I}\right\|_F}{\|\mathbf{I}\|_F} = \frac{\left\|\left(\mathbf{AR}_\mathbf{S}^{-1}\right)^\top \mathbf{AR}_\mathbf{S}^{-1} - \mathbf{I}\right\|_F}{\sqrt{n}},$$

along with the pair of minimum- and maximum singular values of $\mathbf{AR}_\mathbf{S}^{-1}$. In return for acquiring an approximation, Algorithm 2 should be faster than the exact calculation, which in this case is a thin QR decomposition. In MATLAB, this is done by first computing `X = qr(A)` and then `R = triu(X(1:n,:))`.

We have performed multiple experiments on different problem types of varying sizes. Table 3.2 presents a subset of the experiments. The table includes results for `srand` and `cond10` matrices both with dimensions $m = 2^{15}$ and $n = 2^{10}$. As sketching parameter, we used $k = 4n$ for all the different sketching methods.

Using $k = 4n$, the relative Frobenius norm error is at most 0.860 across the four different sketching methods and two problem types. This can be improved further

Table 3.2: Approximation of an orthonormalising transformation matrix for a problem of size $m = 2^{15}$ and $n = 2^{10}$. All sketching methods used a sketching matrix with $k = 4n$ rows. The error terms are based on the residual matrix $\mathbf{Q}_\text{res} = \frac{1}{\sqrt{n}}\left(\left(\mathbf{AR}_\mathbf{S}^{-1}\right)^\top \mathbf{AR}_\mathbf{S}^{-1} - \mathbf{I}\right)$ and the smallest and largest singular values of $\mathbf{AR}_\mathbf{S}^{-1}$, σ_min and σ_max.

	`srand`		`cond10`		Gain
Sketching method	$\|\mathbf{Q}_\text{res}\|_F$	$[\sigma_\text{min}; \sigma_\text{max}]$	$\|\mathbf{Q}_\text{res}\|_F$	$[\sigma_\text{min}; \sigma_\text{max}]$	factor
Leverage score sampling	0.860	$[0.655; 2.026]$	0.849	$[0.663; 2.016]$	2.0
Gaussian projection	0.840	$[0.668; 1.991]$	0.840	$[0.668; 1.990]$	0.31
SRHT	0.767	$[0.691; 1.926]$	0.767	$[0.690; 1.927]$	4.4
Sparse projection	0.842	$[0.665; 1.994]$	0.841	$[0.668; 1.993]$	7.7

by increasing the sketching parameter k, but this would also lower its advantage in speed. In order to visualise a Frobenius norm error of up to 0.841 from unity obtained for the `cond10` matrix with the sparse projection method, the diagonal entries of $\left(\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}\right)^{\top}\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}$ are shown in Figure 3.4. We note that the diagonal entries are generally increasing. Intuitively, this can be understood by noting that both $\mathbf{R}_{\mathbf{S}}$ and $\mathbf{R}_{\mathbf{S}}^{-1}$ are upper triangular matrices and hence the errors of nonzero entries of $\mathbf{R}_{\mathbf{S}}^{-1}$ accumulate along the diagonal part of the product $\left(\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}\right)^{\top}\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}$.

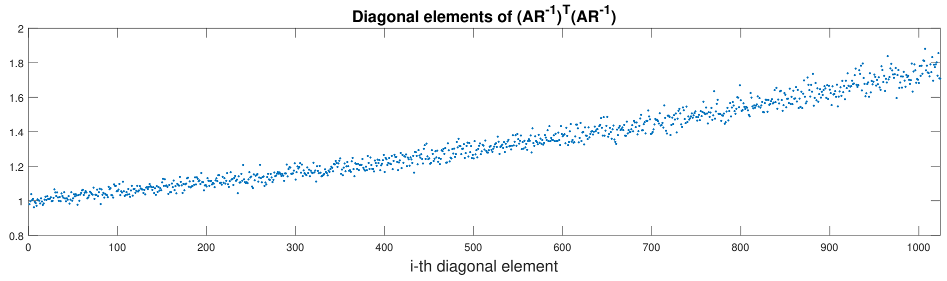


Figure 3.4: Diagonal entries of $\left(\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}\right)^{\top}\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}$ where $\mathbf{A}$ is a `cond10` matrix of size $m = 2^{15}$ and $n = 2^{10}$. $\mathbf{R}_{\mathbf{S}}$ was constructed using sparse projection with sketch size parameter $k = 4n$.

It is seen that the approximate transformation matrix has somewhat stabilised the singular values. This is especially clear for the `cond10` matrix where the condition number of $\mathbf{A}$ is $\kappa(\mathbf{A}) = 10^{10}$, whereas in Table 3.2, it is at most $\kappa\left(\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}\right) = 3.63$. This motivates the use of $\mathbf{R}_{\mathbf{S}}$ or $\mathbf{R}_{\mathbf{S}}^{-1}$ as preconditioners, an application we will investigate later as part of the overdetermined least squares problem. Taking other experiments with other problem types, problem sizes and choices of sketching parameter into account, we found that reasonable results were again obtained with k proportional to n.

Considering the applicability of Algorithm 2, we need k to be less than m in order for the approximation method to be faster. Furthermore, k must be larger than n in order to get a transformation matrix of the right size. Thus the algorithm is best used in situations where a tall and skinny data matrix is considered. This also explains the poor gain factor of the Gaussian projection method, since the matrix product involved in constructing the sketch is of complexity $O(kmn)$, which is greater than $O\left(mn^2\right)$.

As will be seen in Section 3.2.1, the use of our leverage score sampling algorithm for approximating an orthonormalising transformation matrix is somewhat circular. When estimating the leverage scores used to construct the sketch, we use an orthonormalising transformation matrix and therefore apply Algorithm 2. However, as seen from Table 3.2, it is still possible to obtain a gain factor of 2. There might also be scenarios where the leverage scores are known beforehand or have to be estimated for other uses, making leverage score sampling a viable option.

To further show the potential of the algorithm, consider a `cond10` matrix of size $m = 2^{17}$ and $n = 2^{10}$. In this type of problem, the concern is often the conditioning of the data matrix and therefore we seek to make a transformation stabilising the singular values. Using the sparse projection method with $k = 2n$, we are capable of constructing a transformation matrix almost 46 times faster than it would be possible to acquire its exact version. The condition number of the approximation is $\kappa\left(\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}\right) = 5.87$ and though not perfect, it leaves us with a relatively well-conditioned problem.

3.1.3 Low-dimensional orthonormal basis

Assume an $m \times n$ matrix $\mathbf{A}$ has some redundancy in its columns such that the column space of $\mathbf{A}$ is well-described by the span of $k < n$ vectors. Instead of finding an approximate orthonormal basis with n columns as described in Section 3.1.2, it might be preferable to directly find an approximation of a k-dimensional basis.

The sketch of a matrix is a projection onto a lower dimensional space, hence if we sketch the transposed matrix $\mathbf{A}^\top$, we obtain a new matrix with a lower number of rows while approximately preserving distances in the row space of $\mathbf{A}^\top$. More formally, if $\mathbf{S}$ is a $k \times n$ sketching matrix, the k columns of the transposed product $\left(\mathbf{S}\mathbf{A}^\top\right)^\top = \mathbf{A}\mathbf{S}^\top$ should capture most of the column space of $\mathbf{A}$. An approximate k-dimensional orthonormal basis can then be obtained by computing the QR decomposition of $\mathbf{A}\mathbf{S}^\top$, which is both less time and space consuming than calculating the exact orthonormal basis of $\mathbf{A}$. The following theorem will help us assess the quality of the approximation.

Theorem 14 ([HMT11, Theorem 9.1]). *Fix $r \in \mathbb{N}$ and let $\mathbf{A}$ be an $m \times n$ matrix with singular value decomposition*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{U}\begin{bmatrix} \mathbf{\Sigma}_1 & \\ & \mathbf{\Sigma}_2 \end{bmatrix}\begin{bmatrix} \mathbf{V}_1^\top \\ \mathbf{V}_2^\top \end{bmatrix},$$

where $\mathbf{\Sigma}_1$ and $\mathbf{\Sigma}_2$ are diagonal matrices containing the first r and remaining $n-r$ singular values of $\mathbf{A}$, respectively, and $\mathbf{V}_1$ and $\mathbf{V}_2$ contain the corresponding right singular vectors of $\mathbf{A}$. Let $\mathbf{S}$ be a $k \times n$ sketching matrix and let $\mathbf{Q}_\mathbf{S}$ be an orthonormal basis for the sketch $\mathbf{A}\mathbf{S}^\top$. Define $\mathbf{S}_1 = (\mathbf{S}\mathbf{V}_1)^\top$ and $\mathbf{S}_2 = (\mathbf{S}\mathbf{V}_2)^\top$. Assuming $\mathbf{S}_1$ has full rank, the approximation error satisfies

$$\left\|\mathbf{A} - \mathbf{Q}_\mathbf{S}\mathbf{Q}_\mathbf{S}^\top\mathbf{A}\right\|_\xi^2 \leq \|\mathbf{\Sigma}_2\|_\xi^2 + \left\|\mathbf{\Sigma_2}\mathbf{S}_2\mathbf{S}_1^\dagger\right\|_\xi^2 \qquad \text{for } \xi = 2, F,$$

where $\mathbf{S}_1^\dagger$ is the Moore-Penrose pseudoinverse of $\mathbf{S}_1$, i.e. $\mathbf{S}_1^\dagger = \left(\mathbf{S}_1^\top\mathbf{S}_1\right)^{-1}\mathbf{S}_1^\top$.

We now investigate the implications of this result for sketching matrices that are chosen specifically to be subspace embeddings. If $\mathbf{S}$ is a $(1 \pm \varepsilon)$ subspace embedding for $\mathbf{V}_1$, then all singular values of $\mathbf{S}_1 = \mathbf{V}_1^\top\mathbf{S}^\top$ are within $1 \pm \varepsilon$, and hence $\mathbf{S}_1$ has full rank as long as $\varepsilon < 1$.

The inequality of Theorem 14 is rewritten by use of submultiplicativity and the relationship between the Frobenius and spectral norms[2], such that

$$\left\|\mathbf{A} - \mathbf{Q}_\mathbf{S}\mathbf{Q}_\mathbf{S}^\top\mathbf{A}\right\|_\xi^2 \leq \|\mathbf{\Sigma}_2\|_\xi^2 \left(1 + \|\mathbf{S}_2\|_2^2 \left\|\mathbf{S}_1^\dagger\right\|_2^2\right), \qquad \text{for } \xi = 2, F.$$

The terms $\|\mathbf{\Sigma}_2\|_\xi^2$ represent the smallest obtainable approximation error, and since $\mathbf{\Sigma}_2$ is a diagonal matrix containing the smallest $n-r$ singular values of $\mathbf{A}$, and can be expressed simply as

$$\|\mathbf{\Sigma}_2\|_2^2 = \sigma_{r+1}^2(\mathbf{A}) \qquad \text{and} \qquad \|\mathbf{\Sigma}_2\|_F^2 = \operatorname{trace}\left(\mathbf{\Sigma}_2^\top\mathbf{\Sigma}_2\right) = \sum_{j>r}\sigma_j^2(\mathbf{A}).$$

The factor $\left(1 + \|\mathbf{S}_2\|_2^2 \left\|\mathbf{S}_1^\dagger\right\|_2^2\right)$ therefore represents the loss in accuracy due to the approximation method. Letting $\mathbf{S}_1 = \mathbf{U}_1\mathbf{D}_1\mathbf{W}_1^\top$ denote the singular value decomposition of $\mathbf{S}_1$, the spectral norm of $\mathbf{S}_1^\dagger$ can be bounded by

$$\left\|\mathbf{S}_1^\dagger\right\|_2 = \left\|\mathbf{W}_1\mathbf{D}_1^\dagger\mathbf{U}_1^\top\right\|_2 = \left\|\mathbf{D}_1^\dagger\right\|_2 = \frac{1}{\sigma_{\min}(\mathbf{S}_1)} \leq \frac{1}{1-\varepsilon}.$$

We can rewrite the norm of $\mathbf{S}_2$ in terms of the spectral norm of the sketching matrix $\mathbf{S}$ by

$$\|\mathbf{S}_2\|_2 = \left\|\mathbf{V}_2^\top\mathbf{S}^\top\right\|_2 \leq \left\|\mathbf{V}_2^\top\right\|_2 \|\mathbf{S}^\top\|_2 = \|\mathbf{S}\|_2,$$

[2]In particular, we use that, for conforming matrices $\mathbf{A}$ and $\mathbf{B}$, $\|\mathbf{AB}\|_F \leq \|\mathbf{A}\|_2 \|\mathbf{B}\|_F$. Assuming $\mathbf{B}$ has n columns, this relation holds since

$$\|\mathbf{AB}\|_F^2 = \sum_{j=1}^n \|\mathbf{AB}^{(j)}\|_F^2 = \sum_{j=1}^n \|\mathbf{AB}^{(j)}\|_2^2 \leq \|\mathbf{A}\|_2^2 \sum_{j=1}^n \|\mathbf{B}^{(j)}\|_2^2 = \|\mathbf{A}\|_2^2 \|\mathbf{B}\|_F^2.$$

where we exploit the orthonormality of $\mathbf{V}_2^\top$. For some choices of sketching matrix, the spectral norm of $\mathbf{S}$ is simple to bound or even express directly. For example, if $\mathbf{S}$ represents the subsampled randomised Hadamard transform, the sketching matrix is orthonormal by construction and hence $\|\mathbf{S}\|_2 = 1$. Using the above, we get the following expressions for the approximation in the spectral and Frobenius norm, respectively, when $\mathbf{S}$ is a $(1 \pm \varepsilon)$ subspace embedding for $\mathbf{V}_1$:

$$\left\|\mathbf{A} - \mathbf{Q_S}\mathbf{Q_S}^\top\mathbf{A}\right\|_2^2 \leq \left(1 + \frac{\|\mathbf{S}\|_2^2}{(1-\varepsilon)^2}\right)\sigma_{r+1}^2(\mathbf{A}), \tag{3.2}$$

$$\left\|\mathbf{A} - \mathbf{Q_S}\mathbf{Q_S}^\top\mathbf{A}\right\|_F^2 \leq \left(1 + \frac{\|\mathbf{S}\|_2^2}{(1-\varepsilon)^2}\right)\left(\sum_{j>r}\sigma_j^2(\mathbf{A})\right). \tag{3.3}$$

Note that the quality of the approximation depends explicitly on the decay of the singular values. In particular, the approximation is exact if $\mathbf{A}$ is of rank r or less.

The sketching algorithm for computing an approximate basis for the range of the input matrix is given in Algorithm 3.

The computational cost of computing the approximation $\mathbf{Q_S}$ consists of the time to sketch the original problem and then perform a QR decomposition of the sketch. Hence the computational cost is of order $O\left(mk^2\right)$, whereas a standard QR decomposition of $\mathbf{A}$ takes $O\left(mn^2\right)$ time, and Algorithm 3 can therefore provide a significant saving if n is large. In comparison, the complexity of the method presented in Section 3.1.2 to find an orthonormalising transformation matrix was $O\left(kn^2\right)$.

Algorithm 3 Approximation of k-dimensional orthonormal basis

```
Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$ and sketch parameter $k$
Output: $\mathbf{Q_S} \in \mathbb{R}^{m \times k}$
1: Construct a sketching matrix $\mathbf{S} \in \mathbb{R}^{k \times n}$
2: Sketch the input matrix $\mathbf{A}_{\text{sketch}} = \left(\mathbf{S} \cdot \mathbf{A}^\top\right)^\top$
3: Compute a QR decomposition $\mathbf{A}_{\text{sketch}} = \mathbf{Q_S}\mathbf{R_S}$
```

Experimental results. We investigate the properties and advantages of using sketching to approximate a k-dimensional orthonormal basis by considering a number of test cases. We will focus on two types of input matrices, namely the `polydecay` and `cond10` types. Common for both test matrices is that the spectral decay is relatively fast and we can therefore hope for a k-dimensional basis that approximates the column space of the input matrix well. By construction, the i-th singular value of the `polydecay` and `cond10` matrices depend only on i and not on the specific instantiation of the test matrix.

We test the quality of the approximate k-dimensional orthonormal basis $\mathbf{Q_S}$ by computing the low-rank approximation $\mathbf{Q_S}\mathbf{Q_S}^\top\mathbf{A}$ and considering the residual matrix $\mathbf{A}_{\text{res}} = \mathbf{A} - \mathbf{Q_S}\mathbf{Q_S}^\top\mathbf{A}$, using both the spectral and Frobenius norm. Thus $\|\mathbf{A}_{\text{res}}\|_2$ and $\|\mathbf{A}_{\text{res}}\|_F$ provide error terms for the quality of approximation as is also expressed in Equation (3.2) and (3.3).

The best k-dimensional basis is the matrix consisting of the left singular vectors of the input matrix corresponding to the k largest singular values. In Matlab, this is computed using the built-in function `svd` and then extracting the first k left singular vectors, which is faster than using Matlab's `svds` for moderately sized k. The approximations are computed using our implementation of Algorithm 3 with the results reported in Table 3.3.

The gain factors presented in Table 3.3 show that the approximation of a k-dimensional basis for a given input matrix is relatively fast compared to finding the best k-dimensional basis. Using the sparse projection, our approximation method is 36 times faster than performing the exact calculation and the residual errors are comparable to those of the best k-dimensional basis. Again we see that the leverage score sampling was relatively slow with only a small gain in speed. Observe that for both test matrices, the residual errors of the approximations are small but also relatively far from the optimal error. This difference is greatly reduced when comparing with the best $k/2$-dimensional basis, indicating that using Algorithm 3 with a few more

Table 3.3: Experimental results for the approximate k-dimensional basis for an input matrix of size $m = 2^{10}$, $n = 2^{12}$ and using $k = m/2$ rows in the sketching matrix. The error terms are based on the normalised residual matrix $\mathbf{A}_{\text{res}}^{(\xi)} = \left(\mathbf{A} - \mathbf{Q_S}\mathbf{Q_S}^\top\mathbf{A}\right) / \|\mathbf{A}\|_\xi$.

	`polydecay`		`cond10`		Gain
Sketching method	$\|\mathbf{A}_{\text{res}}^{(2)}\|_2$	$\|\mathbf{A}_{\text{res}}^{(F)}\|_F$	$\|\mathbf{A}_{\text{res}}^{(2)}\|_2$	$\|\mathbf{A}_{\text{res}}^{(F)}\|_F$	factor
Leverage score sampling	0.0056	0.041	$1.3 \cdot 10^{-4}$	$8.9 \cdot 10^{-5}$	1.8
Gaussian projection	0.0053	0.040	$7.3 \cdot 10^{-5}$	$4.8 \cdot 10^{-5}$	9.3
SRHT	0.0053	0.040	$7.5 \cdot 10^{-5}$	$4.8 \cdot 10^{-5}$	15
Sparse projection	0.0053	0.040	$7.2 \cdot 10^{-5}$	$4.8 \cdot 10^{-5}$	36
Optimal k-dimensional basis	0.0019	0.024	$9.9 \cdot 10^{-6}$	$9.9 \cdot 10^{-6}$	1
Optimal $k/2$-dimensional basis	0.0039	0.042	$3.1 \cdot 10^{-3}$	$3.1 \cdot 10^{-3}$	1

columns improves the approximation greatly and is therefore worth considering.

To better understand the quality of approximation offered by Algorithm 3, we consider an image reconstruction problem. We use MATLAB's MRI sample image `mri.tif` of size 128×128 pixels as our test image. Figure 3.5 shows the original image along with the reconstructions generated using the approximated and best k-dimensional orthonormal bases, with $k = 91$. As the plot of the singular values in Figure 3.5 shows, the image is practically of rank 91, with the 37 smallest singular values all below 10^{-10}, and the normalised root mean square error (nrmse) of the best approximation is also negligible. The normalised root mean square error of the sketched approximation, however, is around 5%.

Figure 3.6 shows the evolution of both the spectral and Frobenius norm errors

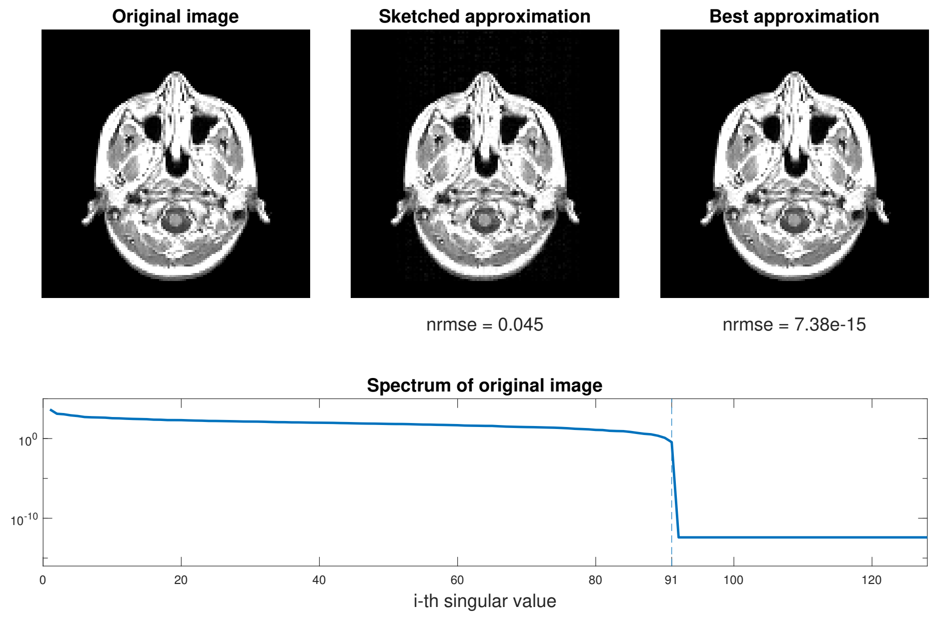


Figure 3.5: Reconstruction of MATLAB's sample `mri.tif` image using $k = 91$ basis vectors and the sketched approximation $\mathbf{A} \approx \mathbf{Q_S}\mathbf{Q_S}^\top\mathbf{A}$, with $\mathbf{Q_S}$ computed as by Algorithm 3, and the best rank k approximation $\mathbf{A} \approx \mathbf{U}_k\mathbf{U}_k^\top\mathbf{A}$, where $\mathbf{U}_k$ is the matrix of the singular vectors of $\mathbf{A}$ corresponding to the k largest singular values.

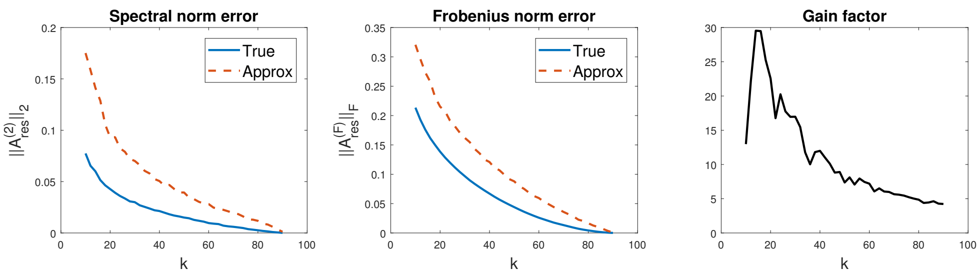


Figure 3.6: The reconstruction error for MATLAB's sample `mri.tif` image as a function of k using Algorithm 3 to approximate a k-dimensional basis and MATLAB's `svd` to calculate the best k-dimensional basis. The rightmost plot shows the achieved gain factor in computation time.

as a function of the sketch parameter k. This shows that the approximation error achieved by Algorithm 3 roughly follows the error expressions from Equations (3.2) and (3.3), since it appears to decay at the same rate as the error of the best rank k approximation. The gain factor shown in the last plot of Figure 3.6 illustrates the potential advantages with respect to computation time.

3.2 Generic linear algebra problems

We will now examine the use of sketching in three generic linear algebra problems, namely leverage score estimation, overdetermined least squares and the singular value decomposition. Our aim is to show how approaches based on the building blocks from the previous section can be used to achieve computational advantages. We present concrete algorithms that are applicable in the most general settings and attempt to give an intuitive understanding of the role of randomisation as a resource with experimental results supporting the analysis.

3.2.1 Leverage score estimation

Leverage score sampling was introduced as a sketching method in Section 2.1.1. The two main results of the section, Theorem 2 and Theorem 3, are already based on leverage score estimation, which we will cover now.

As mentioned previously, computing the exact leverage scores of an $m \times n$ matrix $\mathbf{A}$ requires the computation of an exact orthonormal basis either via QR or singular value decompositions, both of which are of complexity $O\left(mn^2\right)$ [GV12]. To obtain an approximation of the leverage scores, we can instead utilise the orthonormal basis approximation of Section 3.1.2, an approach similar to those of [Woo14] and [CW13]. Algorithm 4 shows in detail how the leverage scores can be approximated.

Algorithm 4 Leverage score approximation

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$ and sketch parameters $k_1$ and $k_2$ (optional)
Output: Approximate leverage scores $\{q_i\}_{i=1}^m$
1: Use Algorithm 2 with sketch parameter $k_1$ to obtain $\mathbf{R_S}$
2: Compute the inverse change of basis matrix $\mathbf{R_S}^{-1}$
3: if second sketch desired then
4:     Construct sketching matrix $\mathbf{S} \in \mathbb{R}^{k_2\times n}$
5:     Sketch the inverse change of basis matrix $\mathbf{R_S}^{-1} = \mathbf{R_S}^{-1}\mathbf{S}^\top$
6: end if
7: Compute approximate orthonormal basis $\mathbf{Q}_{\text{approx}} = \mathbf{A}\mathbf{R_S}^{-1}$
8: Calculate leverage scores $\hat{q}_i = \| (\mathbf{Q}_{\text{approx}})_{(i)} \|_2^2$ for $i = 1, \ldots, m$
9: Normalise the leverage scores by $q_i = \hat{q}_i / \sum_j \hat{q}_j$ for all $i$
```

The algorithm includes an optional second sketch of the inverted change of basis matrix. This is to avoid computing the full matrix product $\mathbf{A}\mathbf{R_S}^{-1}$, which takes $O\left(mn^2\right)$ time. However, if $\mathbf{A}$ is very tall, the algorithm can still yield a speed up in computation time without a second sketch. Naturally, using a second sketch must be expected to reduce the accuracy of the leverage score estimation.

Since $\mathbf{A}$ is generally large, the sketching matrix used to obtain $\mathbf{R_S}$ should be chosen such that the sketch can be computed relatively fast. A sparse projection matrix is used in [Woo14, Theorem 2.13], while [CW13, Theorem 29] opts for a composition of a sparse projection matrix and an SRHT matrix. The main purpose of the sketching matrix $\mathbf{S}_2$ is to reduce the number of rows of the transposed matrix and the sketch time is therefore not as important as keeping the number of rows to a minimum. This makes the Gaussian sketching matrix a viable option, as chosen in both [Woo14] and [CW13].

The computational expense of the above procedure consists of contributions from computing the sketches, the QR decomposition, $O\left(k_1 n^2\right)$, inverting the change of basis matrix, $O\left(n^3\right)$, and the final matrix product, $O\left(k_2mn\right)$ or $O\left(mn^2\right)$. Computing the row norms is negligible. The actual computation time depends on the choice of sketching matrices as this impacts both the sketch time and the number of rows required but it is $O\left(\left(k_1+n\right)n^2+k_2mn\right)$.

Experimental results. In order to test the quality of the leverage score estimation, we consider three different error measurements:

- The relative error between the vector of true leverage scores $\boldsymbol{\ell}=(\ell_1,\ell_2,\dots,\ell_m)$ and estimated leverage scores $\mathbf{q}=(q_1,q_2,\dots,q_m)$ given by $\|\mathbf{q}-\boldsymbol{\ell}\|_2\,/\,\|\boldsymbol{\ell}\|_2$. This measures the accuracy of the approximation relative to the true scores.
- The β-value from Theorem 2 calculated as $\beta=\min\limits_{i=1,\dots,m} q_i/\ell_i$, which shows the maximal underestimation of the approximated leverage scores.
- The significance rate which is the fraction of "significant" true leverage scores that are also estimated to be "significant". We will define significant leverage scores as those scores above $2/m$ as is chosen in for example [HW78].

A low relative error is not necessarily indicative of a good estimation as the high leverage scores might not be captured. Similarly, a low β-value can be due to large relative differences between small leverage scores, which are perhaps not as important as the larger leverage scores.

Table 3.4 shows the results of the leverage score estimation algorithm on the `srand` test matrix. As mentioned above, the first sketching method should be fast, and hence the sparse projection sketching matrix (SPM) is chosen. For the optional second sketch, the emphasis is on accuracy using a low number of rows, and we therefore consider both the Gaussian projection and subsampled randomised Hadamard

Table 3.4: Approximate leverage scores for a $2^{15}\times 2^{10}$ `srand` matrix.

k_1	sketch 1	k_2	sketch 2	$\|\mathbf{q}-\boldsymbol{\ell}\|_2\,/\,\|\boldsymbol{\ell}\|_2$	$\min_i q_i/\ell_i$	Sig. rate	Gain factor
$2n$	SPM	n	–	0.0457	0.8381	0.9532	4.6
$2n$	SPM	$n/2$	Gauss	0.0776	0.7281	0.9181	7.1
$2n$	SPM	$n/2$	SRHT	0.0637	0.7685	0.9330	6.7
$2n$	SPM	$n/5$	Gauss	0.1088	0.6336	0.8816	13
$2n$	SPM	$n/5$	SRHT	0.0993	0.6415	0.8918	12

transform (SRHT). Note that leverage score sampling is clearly unsuitable for this application.

As expected, using only a single sketch gives the most accurate results whereas a second sketch gives rise to faster computation times. Interestingly, using SRHT instead of a Gaussian projection for the second sketch yields more precise estimations at the same computational cost. In general, however, the SRHT and Gaussian projections yield very similar accuracy results. We see that using our leverage score estimation algorithm, we can compute approximate scores that capture almost 90% of the significant leverage scores in only a twelfth of the time.

To get an idea of the performance of the algorithm for a real-world data set, we estimate the leverage scores for the real-life `spambase` data set [DK17]. The data set is based on the analysis of 4601 emails for the frequency of certain words and length of sequences of consecutive capital letters. In total there are 57 features that can be used to classify each item as either spam or not spam. The first 48 features are the frequencies of specific words in the email, the next six features are the frequencies of specific punctuation marks, and the final three features describe lengths and occurrences of consecutive capital letters. We discard the final nine features and consider only the word frequencies. We then remove observations that are all zeros leaving us with 4437 observations.

Algorithm 4 is applied using sparse projection with $k_1 = 4n$ and SRHT with $k_2 = n/2$ as the sketching matrices. Estimating the leverage scores corresponding to each item, we find that 389 emails are above the significance level of $2/m$. The sorted significant leverage scores along with their estimates are shown in Figure 3.7.

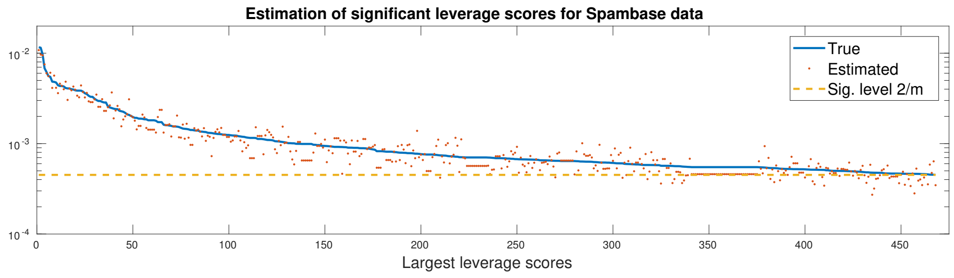


Figure 3.7: Estimation of significant leverage scores for the `spambase` data set, with $m = 4437$ and $n = 48$, using a sparse projection matrix with $k_1 = 4n$ and SRHT with $k_2 = n/2$. The estimation captures 90% of the significant scores.

The relative error for the approximated leverages scores is 0.16, while the β-value is 0.31. Importantly, the significance rate for the estimations is 90%, meaning that very few of the truly significant leverage scores are underestimated to a degree where they are no longer considered significant. If the β-value is calculated only for the significant leverage scores, it increases to 0.51. The approximations are computed about three times as fast as the exact scores.

3.2.2 Overdetermined least squares

Linear systems of equations are ever-present in linear algebra and are at the heart of a large number of problems and algorithms in scientific computing. In matrix form, each row of $\mathbf{A} \in \mathbb{R}^{m \times n}$ and corresponding entry of $\mathbf{b} \in \mathbb{R}^m$ constitute a constraint and we seek a vector $\mathbf{x} \in \mathbb{R}^n$ of variables such that $\mathbf{Ax} = \mathbf{b}$. We will concentrate on the overdetermined version of this problem where the number of constraints is larger than the number of variables, that is $m > n$. We cannot in general hope to obtain equality in $\mathbf{Ax} = \mathbf{b}$, so instead the problem is relaxed to the least squares problem

$$\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{b}\|_2 . \tag{3.4}$$

Two solution methods are the primary subject of this section: solution via the normal equation and iterative solvers such as the MATLAB function `lsqr`.

The maximum likelihood solution to (3.4) is given by the solution to the normal equation

$$\mathbf{A}^\top \mathbf{Ax} = \mathbf{A}^\top \mathbf{b}.$$

Solving the equation in practice can be done in numerous ways, and in MATLAB one would most often use the backslash operator \ or `mldivide`, which relies on different numerical methods depending on the problem but always a QR solver for a rectangular matrix $\mathbf{A}$. In computational considerations, one should pay attention to the necessary matrix-matrix multiplication performed in this method. Computing $\mathbf{A}^\top \mathbf{A}$ can be costly for large $\mathbf{A}$ and avoiding this could lead to major savings in time, for example by using the approximation matrix multiplication method of Section 3.1.1. To see how sketching can be used in the context of overdetermined least squares approximation, let $\mathbf{S} \in \mathbb{R}^{k \times m}$ be a sketching matrix satisfying the property of a $(1 \pm \varepsilon)$ subspace embedding and with $k < m$. It follows directly from the definition of a subspace embedding that

$$(1 - \varepsilon) \|\mathbf{S}(\mathbf{Ax} - \mathbf{b})\|_2 \leq \|\mathbf{Ax} - \mathbf{b}\|_2 \leq (1 + \varepsilon) \|\mathbf{S}(\mathbf{Ax} - \mathbf{b})\|_2 .$$

This implies that by solving the smaller problem $\min_{\mathbf{x}} \|\mathbf{SAx} - \mathbf{Sb}\|_2$, we get an $(1 \pm \varepsilon)$ approximation of the solution to (3.4). Algorithm 5 implements the solution method based on the sketched normal equation. We do not explicitly use the approximate matrix multiplication of Algorithm 1, since the sketching matrix $\mathbf{S}$ used to compute $\mathbf{SA}$ is also needed to construct the sketch $\mathbf{Sb}$, but the idea is the same.

Algorithm 5 Overdetermined least squares (normal equation)

```
Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and sketch parameter $k$
Output: $\mathbf{x} \in \mathbb{R}^n$
1: Construct sketching matrix $\mathbf{S} \in \mathbb{R}^{k \times m}$
2: Sketch the input arrays $\mathbf{A}_{\text{sketch}} = \mathbf{S} \cdot \mathbf{A}$ and $\mathbf{b}_{\text{sketch}} = \mathbf{S} \cdot \mathbf{b}$
3: Solve the normal equation $\mathbf{A}_{\text{sketch}}^\top \mathbf{A}_{\text{sketch}} \mathbf{x} = \mathbf{A}_{\text{sketch}}^\top \mathbf{b}_{\text{sketch}}$
```

Note that the time to compute the matrix product $(\mathbf{SA})^\top (\mathbf{SA})$ is $O\left(kn^2\right)$, which is substantially lower than the previous complexity of $O\left(mn^2\right)$ when $k \ll m$. Algorithm 5 is a way of using sketching to obtain $(1 \pm \varepsilon)$ approximations to overdetermined least squares approximation problems, however, as the number of rows k in a $(1 \pm \varepsilon)$ subspace embedding matrix depends on ε^{-2}, this method can only be expected to achieve low-precision solutions in practice.

We have not yet commented on the conditioning of the matrix $\mathbf{A}$ and the impact this has on the solution quality of Algorithm 5. If $\mathbf{A}$ is ill-conditioned, the solution obtained through the normal equation is very sensitive to errors in the vector $\mathbf{b}$. One approach is to perform a QR decomposition of the input matrix such that $\mathbf{A} = \mathbf{QR}$ and solving the corresponding least squares problem

$$\min_{\mathbf{x}} \left\|\mathbf{Rx} - \mathbf{Q}^\top \mathbf{b}\right\|_2 .$$

This is numerical better than working directly with the problem of Equation (3.4) [GV12; Bjo96].

Alternatively, iterative methods present themselves as robust options for solving (3.4) when $\mathbf{A}$ is ill-conditioned or simply when $\mathbf{A}$ is too large to perform matrix-matrix multiplication. The complexities of the iterative methods are dominated by the number of iterations needed to converge. In general, the iterative solvers obtain an ε approximation to (3.4) in $O(\kappa(\mathbf{A}) \ln(1/\varepsilon))$ iterations [Mah+11]. This illustrates that when $\mathbf{A}$ is ill-conditioned, problem (3.4) is harder to solve and in fact the complexity of the algorithm is dominated by the condition number of the input matrix and not the chosen accuracy parameter ε as seen previously.

To boost the performance of the iterative solvers, one can supply a preconditioning matrix $\mathbf{R}$ such that the matrix $\mathbf{AR}^{-1}$ is well-conditioned. The solver then solves the related problem

$$\min_{\mathbf{y}} \left\|\mathbf{AR}^{-1}\mathbf{y} - \mathbf{b}\right\|_2 , \tag{3.5}$$

where $\mathbf{y} = \mathbf{Rx}$ and the solution to (3.4) is found as $\tilde{\mathbf{x}} = \mathbf{R}^{-1}\tilde{\mathbf{y}}$, with $\tilde{\mathbf{y}}$ the minimiser of Equation (3.5). In practice, the preconditioning matrix is given as input to the iterative solver which avoids computing $\mathbf{AR}^{-1}$ explicitly by only computing matrix-vector multiplications. The optimal choice of preconditioner is the upper triangular matrix $\mathbf{R}$ obtained from the QR decomposition of $\mathbf{A}$. As the singular values of the orthonormal matrix $\mathbf{Q}$ are all 1, we have $\kappa\left(\mathbf{AR}^{-1}\right) = \kappa\left(\mathbf{Q}\right) = 1$, resulting in a lower number of required iterations. However, computing the QR decomposition takes $O\left(mn^2\right)$ time and is therefore not very efficient when m and n are large.

Recall that this was the very problem of Section 3.1.2, where we used sketching to compute an approximate orthonormalising transformation matrix. In this context, we construct the sketch $\mathbf{SA}$ and compute the QR decomposition $\mathbf{SA} = \mathbf{Q_S R_S}$. The matrix $\mathbf{R_S}$ is then given as a preconditioning matrix to the iterative solver. This avoids severe ill-conditioning since $\mathbf{AR_S}^{-1}$ has eigenvalues in the range $1 \pm \varepsilon$ and therefore $\kappa\left(\mathbf{AR_S}^{-1}\right) \leq \frac{1+\varepsilon}{1-\varepsilon}$. In Algorithm 6, problem (3.4) is solved using MATLAB's `lsqr` with a preconditioner obtained through sketching.

Algorithm 6 Overdetermined least squares (`lsqr`)

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$ and sketch parameter $k$
Output: $\mathbf{x} \in \mathbb{R}^n$
1: Use Algorithm 2 with sketch parameter $k$ to obtain $\mathbf{R_S}$
2: Using $\mathbf{R_S}$ as a preconditioner, compute $\mathbf{x} = \texttt{lsqr}\left(\mathbf{A}, \mathbf{b}, \texttt{[]}, \texttt{[]}, \mathbf{R_S}\right)$
```

In total, the complexity of the sketching method consists of constructing $\mathbf{R_S}$ in $O\left(kn^2\right)$ time and then the iterative solver stops in $O\left(\frac{1+\varepsilon}{1-\varepsilon}\ln\left(1/\varepsilon\right)\right)$ iterations. When $\mathbf{A}$ is large and ill-conditioned, this is a significant speed-up compared to computing the exact QR decomposition of $\mathbf{A}$ and then performing $O\left(\ln\left(1/\varepsilon\right)\right)$ iterations, or compared to just running the $O\left(\kappa\left(\mathbf{A}\right)\ln\left(1/\varepsilon\right)\right)$ iterations without any preconditioner.

It is worth noting that Algorithm 6 yields near optimal solutions since the iterative algorithm can theoretically solve the problem to arbitrary precision. In the applications of sketching described this far, sketching has implied a trade-off between computational complexity and precision in the final solution. Here the loss of precision is solely in the preconditioner, which is reflected in the number of iterations required but not in the exactness of the final solution.

Experimental results. To test the quality of Algorithm 5 and 6, we create a random matrix $\mathbf{A}$ and an independent uniform random vector $\mathbf{b}$. The focus will be on the minimisation of $\|\mathbf{Ax} - \mathbf{b}\|_2$ which can be compared across different solution methods and we therefore measure the error as

$$\frac{\|\mathbf{A}\tilde{\mathbf{x}} - \mathbf{b}\|_2}{\|\mathbf{b}\|_2},$$

where $\tilde{\mathbf{x}}$ is the solution to (3.4), either exact or approximately.

We split the experiments in two and consider both a well-conditioned and ill-conditioned case. It does not make sense to use Algorithm 6 in a well-conditioned problem since the transformation to a well-conditioned basis would be unnecessary. At the same time, one would not solve an ill-conditioned least squares problem using the normal equation as the result would be unreliable.

For the well-conditioned tests, we let $\mathbf{A}$ be an `srand` test matrix and solve the given problem using Algorithm 5. These results are compared to the Matlab solution `(A'*A)\(A'*b)`, but we also state the solution obtained by solving `A\b` and and using the Matlab function `lsqr`. This will enable us to compare different solution strategies and especially to see the potential benefit from using an iterative method. One instance of such a test is presented in Table 3.5.

Interestingly, Table 3.5 shows that leverage score sampling performs better with respect to approximation error than the three other methods that all lead to similar

Table 3.5: Approximate solution to (3.4) using Algorithm 5 on an `srand` type problem of size $m = 2^{17}$ and $n = 2^9$ and with sketching parameter $k = 4n$.

Sketching method	$\frac{\|\mathbf{A}\tilde{\mathbf{x}}-\mathbf{b}\|_2}{\|\mathbf{b}\|_2}$	Gain factor
Leverage score sampling	0.708	0.75
Gaussian projection	0.765	0.066
SRHT	0.756	1.5
Sparse projection	0.764	8.4
`A\b`	0.660	0.054
`(A'*A)\(A'*b)`	0.660	1
`lsqr(A,b)` in 4 iterations	0.660	1.8

results. Regarding approximation time, sparse projection is by far the fastest method and achieves an approximation with a relative error of around 15%, roughly 8 times faster than the exact computation. Once again the Gaussian projection is extremely slow owing to the large matrix-matrix multiplication involved in forming the sketch. In general it holds, as for the previous experiments, that choosing k proportional to n is enough to get relatively good results. Further evidence of this is obtained using sparse projection with $k = 16n$, which results in a 3% approximation error still only using one third of the time of the exact calculation. Finally, we note that the three direct solution methods differ quite a lot when it comes to speed. `A\b` is much slower than solving the normal equation due to the properties of $\mathbf{A}$ compared to $\mathbf{A}^\top\mathbf{A}$. In contrast, `lsqr` beats the normal equation with a factor of almost two. This is due to `lsqr` avoiding matrix-matrix multiplications and thus we expect even greater savings for larger problems.

For the ill-conditioned tests, we let $\mathbf{A}$ be the `cond10` test matrix and solve the least squares problem using Algorithm 6. We compare the approximation with the solution obtained by the MATLAB computation `lsqr(A,b,[],[],R)`, where $\mathbf{R}$ is the exact transformation matrix obtained in MATLAB by first computing `X = qr(A)` and then extracting `R = triu(X(1:n,:))`. Furthermore, we state the solution obtained without use of preconditioner to see the potential effect on the number of iterations needed. The results are presented in Table 3.6.

As previously mentioned, Algorithm 6 does not produce an approximation of the solution but rather approximates a preconditioning matrix which is then used to solve the given problem to a desired precision. This is clearly reflected in Table 3.6, where all solution methods achieve the same error. On the other hand, the number of

Table 3.6: Approximate solution to (3.4) using Algorithm 6 on a `cond10` type problem of size $m = 2^{17}$ and $n = 2^{10}$ and with sketching parameter $k = 4n$.

Sketching method	$\frac{\|\mathbf{A}\tilde{\mathbf{x}}-\mathbf{b}\|_2}{\|\mathbf{b}\|_2}$	Iterations	Gain factor
Leverage score sampling	0.498	19	1.4
Gaussian projection	0.498	19	0.31
SRHT	0.498	18	2.1
Sparse projection	0.498	19	2.7
`lsqr(A,b,[],[],R)`	0.498	2	1
`lsqr(A,b)`	0.499	2400	0.03

iterations needed to converge depends greatly on the chosen solution method. Using the exact transformation matrix, we need only two iterations whereas at least 18 iterations are performed when we approximate the transformation matrix. However, the computational bottleneck is the QR decomposition rather than the iterations of the `lsqr` function and Algorithm 6 therefore ends up being faster for all but the Gaussian projection. Note that if the matrix $\mathbf{A}$ can be applied quickly to a vector, the gain factors would be larger as additional `lsqr` iterations come at a smaller cost.

3.2.3 Singular value decomposition

The singular value decomposition is of great significance within numerical linear algebra with applications in for example low-rank approximation, computing inverses and pseudoinverses, and solving inverse problems. As for the QR factorisation, the conventional singular value decomposition takes $O\left(mn^2\right)$ time. If the matrix $\mathbf{A}$ is well-approximated by a rank k matrix, or if we are only interested in obtaining k singular vectors that account for most of the variance, we could hope to do this faster. Methods for rank revealing decompositions such as the Lanczos algorithm already exist, for example in MATLAB's `svds` [Lar98], but sketching provides a number of other opportunities.

To gain some insight into why low-rank approximations can be useful, suppose $\mathbf{A}$ consists of measurements such that $\mathbf{A} = \mathbf{B} + \mathbf{E}$, where $\mathbf{B}$ contains the true model values and $\mathbf{E}$ represents noise. Even if the underlying model described by $\mathbf{B}$ has low rank, the matrix $\mathbf{A}$ might have high or even full rank. In such cases, the aim is to find singular vectors and values describing $\mathbf{B}$ rather than $\mathbf{A}$ to minimise the influence of the observed noise.

The main idea behind the randomised singular value decomposition is precisely the assumption of rank-deficiency in the $m \times n$ input matrix $\mathbf{A}$. If this is the case, we can use Algorithm 3 to compute a low-dimensional approximation of an orthonormal matrix $\mathbf{Q_S}$ such that $\mathbf{A} \approx \mathbf{Q_S}\mathbf{Q_S}^\top\mathbf{A}$. An actual SVD is then calculated on the $k \times k$ solution $\mathbf{X}$ to the generalised least squares problem $\mathbf{Q_S}\mathbf{X} = \mathbf{A}$, yielding components that can be used directly to obtain an approximate singular value decomposition of $\mathbf{A}$. The resulting factors can be truncated to achieve a certain target rank for the decomposition. The explained procedure is implemented in Algorithm 7.

The generalised least squares problem in step 2 is easy to solve since $\mathbf{Q_S}^\dagger = \mathbf{Q_S}^\top$ by orthonormality of $\mathbf{Q_S}$, and thus we can simply calculate $\mathbf{X} = \mathbf{Q_S}^\top\mathbf{A}$. However, the cost of this matrix multiplication is $O\left(kmn\right)$, and it requires a further pass over the input

Algorithm 7 Randomised SVD

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, sketch parameter $k$ and target rank $r$
Output: $\mathbf{U}_r \in \mathbb{R}^{m\times r}$, $\mathbf{\Sigma}_r \in \mathbb{R}^{r\times r}$ and $\mathbf{V}_r \in \mathbb{R}^{n\times r}$
1: Use Algorithm 3 with sketch parameter $k$ to obtain $\mathbf{Q}_\mathbf{S}$
2: Solve the generalised least squares problem $\mathbf{A} = \mathbf{Q}_\mathbf{S}\mathbf{X}$
3: Compute the SVD $\mathbf{X} = \mathbf{U}_\mathbf{X}\mathbf{\Sigma}\mathbf{V}^\top$
4: Calculate $\mathbf{U} = \mathbf{Q}_\mathbf{S}\mathbf{U}_\mathbf{X}$
5: Extract the first $r$ singular values and vectors to form $\mathbf{U}_r$, $\mathbf{\Sigma}_r$ and $\mathbf{V}_r$
```

matrix $\mathbf{A}$. Tropp et al. [Tro+17] solve the least squares problem by sketching both $\mathbf{A}$ and $\mathbf{Q}_\mathbf{S}$ using a sketching matrix $\mathbf{S}_2$ such that $\mathbf{X} = (\mathbf{S}_2\mathbf{Q}_\mathbf{S})^\dagger (\mathbf{S}_2\mathbf{A})$. This corresponds to the application of Algorithm 5 to the generalised least squares problem $\mathbf{Q}_\mathbf{S}\mathbf{X} = \mathbf{A}$. This can be faster, however, the primary gain is that it only requires access to the sketches of $\mathbf{A}$ and $\mathbf{Q}_\mathbf{S}$ which implies a significant saving in the space and memory required.

In some cases, taking just a single or two power iterations when computing $\mathbf{AS}^\top$ can lead to better accuracy [HMT11]. Calculating $\left(\mathbf{AA}^\top\right)^q\mathbf{AS}^\top$ for a small integer q drives the lower singular values towards zero and reduces the weight of the corresponding singular vectors relative to the dominant singular vectors. Note that $\left(\mathbf{AA}^\top\right)^q\mathbf{A}$ has the same singular vectors as the input matrix $\mathbf{A}$. The power iteration is particularly useful if the focus is not on saving time but rather on space or storage.

Experimental results. There are several things to consider when assessing the randomised SVD algorithm. The quality of approximation depends largely on the further use of the decomposition. If the goal is a rank r approximation, we can calculate spectral or Frobenius norm errors of the low-rank approximation relative to the best rank r approximation as was done in the investigation of the underlying k-dimensional basis approximation in Section 3.1.3. However, if the aim is approximating the first r singular values of the input matrix, these error terms are not particularly suitable. We choose to include the relative squared error between the r estimated and true singular values but in some cases other error measures might better reflect the approximation quality in respect to the desired goal.

Table 3.7 shows the relative low-rank approximation errors and relative error of the singular values for instances of the `polydecay` and `cond10` test matrices. The target rank is set to $r = 5$ with a sketch size of $k = 2r + 1$ inspired by [Tro+17]

Table 3.7: Experimental results for the randomised SVD for an input matrix of size $m = 2^{10}$ and $n = 2^{12}$ with target rank $r = 5$ and using $k = 2r + 1$ rows in the sketching matrix. The gain factor is relative to MATLAB's `svds(A,r)`. The error terms are based on the residual matrix $\mathbf{A}_{\text{res}}^{(\xi)} = (\mathbf{A} - \mathbf{U}_k \mathbf{\Sigma}_k \mathbf{V}_k) \,/\, \|\mathbf{A} - \mathbf{A}_r\|_\xi$ and the relative error between the r estimated and true singular values, $\hat{\boldsymbol{\sigma}}$ and $\boldsymbol{\sigma}$.

	`polydecay`				`cond10`			
Sketching method	$\lVert\mathbf{A}_{\text{res}}^{(2)}\rVert_2$	$\lVert\mathbf{A}_{\text{res}}^{(F)}\rVert_F$	$\frac{\lVert\hat{\boldsymbol{\sigma}}-\boldsymbol{\sigma}\rVert_2}{\lVert\boldsymbol{\sigma}\rVert_2}$	Gain factor	$\lVert\mathbf{A}_{\text{res}}^{(2)}\rVert_2$	$\lVert\mathbf{A}_{\text{res}}^{(F)}\rVert_F$	$\frac{\lVert\hat{\boldsymbol{\sigma}}-\boldsymbol{\sigma}\rVert_2}{\lVert\boldsymbol{\sigma}\rVert_2}$	Gain factor
Leverage score sampling	1.260	1.196	0.0500	0.20	1.073	1.042	0.188	0.64
Gaussian projection	1.238	1.182	0.0462	8.8	1.086	1.043	0.193	28
SRHT	1.266	1.193	0.0460	2.8	1.079	1.042	0.189	9.1
Sparse projection	1.283	1.187	0.0490	11	1.075	1.043	0.196	35

and the observations from Section 3.1.3. Due to the spectral properties of the test types, we see that the obtained low-rank approximations are very good for the `cond10` test matrix but quite inaccurate for the `polydecay` matrix type. On the other hand, the singular values are more accurately approximated for this test type than for the `cond10` matrix. For the `cond10` matrix, the largest singular value is estimated to be around 0.87 with the true value being 1. The following estimated singular values are also estimated well below the true values, leading to the large observed error.

The gain factor showed in Table 3.7 reveals the relative speed of the randomised SVD method when compared to MATLAB's `svds` method for the mentioned test cases. The `svds` function is significantly slower for the ill-conditioned test problem, leading to the large differences in the gain factor achieved by the method across the two test classes. An interesting observation is that the SRHT method is slower than Gaussian projection, which is in contrast to previously observed results. The reason for this is the implementation of the Hadamard transform, which transforms the entire input matrix before the rows are sampled. In comparison, Gaussian projection involves only a small matrix-matrix multiplication as only a very few rows are used.

As an example of singular value approximation, we return to MATLAB's MRI sample image `mri.tif` from Section 3.1.3. Figure 3.8 shows the first $r = 32$ true and estimated singular values, using sparse projection with $k = 2r + 1$ and $k = r + 5$, respectively. This visually shows the effect of sampling additional columns in the construction of the lower dimensional basis. Furthermore, the figure shows that the estimated singular values follow the true ones very closely and exhibit the same pattern.

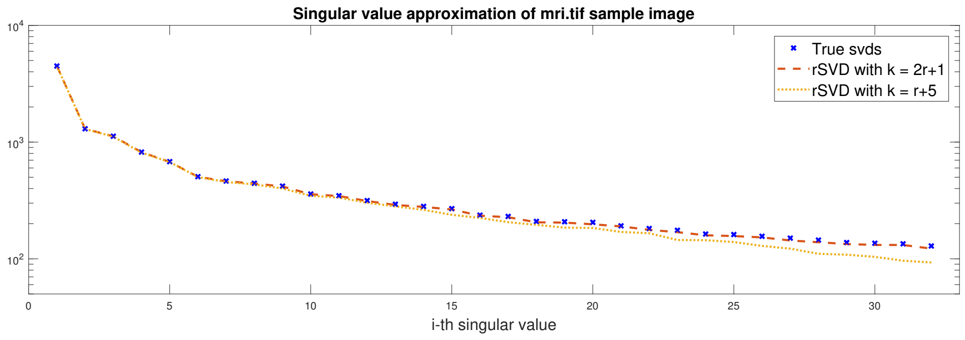


Figure 3.8: Estimation of the largest 32 singular values for MATLAB's `mri.tif` sample image using sparse projection. The relative norm error in the approximation is 0.72% using $k = 2r + 1$ and 2.7% using $k = r + 5$.

3.3 Real-world applications

The sketching algorithms developed for the basic building blocks and the generic linear algebra problems were tested experimentally on mostly synthetic data sets in constructed problems. We will now consider domain specific problems inspired by real-world applications, where the error quantities can be assigned a physical interpretation, allowing us to better assess how the loss of precision due to sketching impacts the applicability of the methods.

We have chosen three applications that show three different advantages of sketching. In matrix completion, sketching can be used to lower the storage cost which in some cases is the bottleneck of existing algorithms. A CUR matrix decomposition can be obtained through use of sketching and can be used to increase the interpretability of the individual decomposition factors. Finally, for the generalised Tikhonov regularisation problem, sketching is used to speed up calculations which would otherwise be very costly.

3.3.1 Matrix completion in recommender systems

In recommender systems, the aim is to predict the rating or preference a user would assign an item based on other user-item ratings. Such systems are employed in many different areas, a commonplace example being the movie recommendations offered by companies like the online streaming service Netflix. Netflix attempts to recommend films to users based on their own rating history and the ratings of other users adjudged to have a similar taste in movies. Each stored rating can be seen as an entry in a user-movie data matrix $\mathbf{D}$, such that $\mathbf{D}_{ij}$ corresponds to the rating user i has given movie j. This matrix is likely to be very sparse as most users have only rated a fraction of all the available films. Predicting the values of the empty entries fills out the matrix and is an example of matrix completion.

Naturally, recreating missing information relies on some underlying assumptions about the correlation between the rating behaviour of different users. It is assumed that a user's preferences are based on a number of descriptive features or characteristics and not on specific movies. These could for example be certain genres, actors or languages. Mathematically, this can be viewed as an assumption that the full, unknown user-movie matrix $\mathbf{X}$ has low rank [Yur+17].

Let E be the index set for the known entries of $\mathbf{D}$ and assume $\mathbf{X}$ has rank at most r. The matrix completion problem can be stated as the constrained optimisation problem

$$\min_{\mathbf{X}} \sum_{(i,j)\in E} f_{\text{loss}}\left(\mathbf{X}_{ij};\mathbf{D}_{ij}\right) \qquad \text{s.t.} \qquad \operatorname{rank}\mathbf{X} \leq r, \tag{3.6}$$

where $f_{\text{loss}}\left(\mathbf{X}_{ij};\mathbf{D}_{ij}\right)$ is an appropriate loss function measuring the cost of predicting $\mathbf{X}_{ij}$ when the true rating is $\mathbf{D}_{ij}$. The loss function could for example be the squared loss or logistic function. Solving this problem is extremely expensive [Yur+17], and we therefore consider the relaxed convex constrained optimisation problem

$$\min_{\mathbf{X}} \sum_{(i,j)\in E} f_{\text{loss}}\left(\mathbf{X}_{ij};\mathbf{D}_{ij}\right) \qquad \text{s.t.} \qquad \|\mathbf{X}\|_{S_1} \leq \alpha, \tag{3.7}$$

where α is a chosen parameter affecting the rank of the solution through the Schatten 1-norm $\|\cdot\|_{S_1}$ corresponding to the sum of the singular values.

We will use the matrix completion problem (3.7) as an example where sketching can be used to gain a computational advantage. More specifically, we will focus on lowering storage costs as recommender systems often deal with massive data sets with large numbers of users and items. Sketching is imposed through a modification of an existing matrix completion algorithm called the Conditional Gradient Method (CGM). The resulting randomised CGM algorithm was introduced in [Yur+17] as `sketchyCGM`. We will first explain the concepts of the standard CGM before formally stating the `sketchyCGM` algorithm.

Let m be the number of users, n the number of items and d the number of ratings such that $d = |E|$. We will use the notation of [Yur+17] and let $\mathcal{A} : \mathbb{R}^{m\times n} \to \mathbb{R}^d$ be the linear operator picking elements corresponding to the entries of the subset E such that, $\mathcal{A}\mathbf{X}$ is a vector with entries $\mathbf{X}_{ij}$ for $(i,j) \in E$. Let $\mathcal{A}^\star : \mathbb{R}^d \to \mathbb{R}^{m\times n}$ be its adjoint operator where, for a vector $\mathbf{z}$, $\mathcal{A}^\star\mathbf{z}$ is the matrix obtained by replacing the known values of $\mathbf{D}$ with the values in $\mathbf{z}$. In particular, we have $\mathcal{A}^\star(\mathcal{A}\mathbf{X}) = \sum_{(i,j)\in E} \mathbf{X}_{ij}\mathbf{e}_i\mathbf{e}_j^\top$ where $\mathbf{e}_i$ is the i-th unit vector. For simplicity we denote the objective function of (3.7) by $f(\mathcal{A}\mathbf{X})$. The specific formulas used to compute the update direction, decision variable update and stopping criterion of the CGM algorithm stem from [Jag13] and are summarised in the following description of the CGM algorithm.

CGM algorithm. Initially create a feasible solution $\mathbf{X}_0 = \mathbf{0} \in \mathbb{R}^{m\times n}$. For each iteration $t = 0, 1, 2, \ldots$, compute a direction for which the decision variable $\mathbf{X}_t$ is to be updated. The update direction $\mathbf{H}_t$ is based on an approximation of a singular value decomposition, for example the MATLAB function `svds`, and computed by

$$(\mathbf{u}_t, \mathbf{v}_t) = \texttt{svds}\left(\mathcal{A}^\star\left(\nabla f\left(\mathcal{A}\mathbf{X}_t\right)\right), 1\right) \qquad \text{and} \qquad \mathbf{H}_t = -\alpha\mathbf{u}_t\mathbf{v}_t^\top. \tag{3.8}$$

The decision variable is updated such that $\mathbf{X}_t$ remains a feasible solution by

$$\mathbf{X}_{t+1} = (1-\eta_t)\mathbf{X}_t + \eta_t\mathbf{H}_t \qquad \text{where} \qquad \eta_t = \frac{2}{t+2}. \tag{3.9}$$

The previous steps are repeated until the following stopping criterion is satisfied

$$(\mathcal{A}\mathbf{X}_t - \mathcal{A}\mathbf{H}_t)^\top \nabla f(\mathcal{A}\mathbf{X}_t) \le \varepsilon. \tag{3.10}$$

The algorithm deserves some further explanation. The stated update minimises a linearisation of the objective function by only performing a rank 1 update using $\mathbf{H}_t$. We refer to [Jag13] to see that $\mathbf{H}_t$ is in fact the minimiser of this linearisation. The predefined step size parameter η_t is constructed such that we quickly move away from the initial solution, and in later iterations, conservatively move towards an acceptable solution. Alternatively, line-search methods can be used to determine an appropriate step size in each iteration. The stopping criterion is a standard choice and can be used for any convex optimisation problem [Jag13]. The intuition is that we should stop when the update direction gives negligible improvements, that is when the minimiser of the linearisation is not sufficiently different from previous iterations.

Our interest will be in the storage costs of this algorithm. Note that $\mathbf{X}_t \in \mathbb{R}^{m\times n}$, which is generally a dense matrix, is stored in every iteration. Thus the algorithm has a storage cost of $O(mn)$. In situations where m and n are large, this can be a challenge.

The idea of the `SketchyCGM` algorithm is to handle the values of $\mathbf{X}$ corresponding to the known and unknown ratings separately. Equations (3.8) and (3.10) depend only on $\mathcal{A}\mathbf{X}$ and hence only on the entries of $\mathbf{X}$ given by the index set E, allowing us to use a change of variables $\mathbf{z} = \mathcal{A}\mathbf{X} \in \mathbb{R}^d$. We perform the updates of $\mathbf{z}$ by applying $\mathcal{A}$ to both sides of Equation (3.9), such that

$$\mathbf{z}_{t+1} = \mathcal{A}\mathbf{X}_{t+1} = (1-\eta_t)\,\mathcal{A}\mathbf{X}_t + \eta_t\mathcal{A}\mathbf{H}_t = (1-\eta_t)\,\mathbf{z}_t + \eta_t\mathcal{A}\mathbf{H}_t.$$

This expression also contains the vector $\mathcal{A}\mathbf{H}_t$. Fortunately, we do not need to store $\mathbf{H}_t$ as a matrix as we can simply store the defining vectors $\mathbf{u}_t$ and $\mathbf{v}_t$. We can compute $\mathcal{A}\left(-\alpha\mathbf{u}_t\mathbf{v}_t^\top\right)$ efficiently by simple multiplication of the relevant entries in $\mathbf{u}_t$ and $\mathbf{v}_t$. The updates of Equation (3.9) also affect the estimation of the unknown ratings. These are handled by initially computing two smaller randomised sketches and performing the updates on these rather than the large matrix $\mathbf{X}$.

To verify that we can use the sketches as a proxy for updating the matrix $\mathbf{X}$, consider the following: Given some matrix $\mathbf{A}_t$ in iteration t, let $\mathbf{Y}_t = \mathbf{A}_t\mathbf{S}_1^\top$ and $\mathbf{W}_t = \mathbf{S}_2\mathbf{A}_t$, where $\mathbf{S}_1$ and $\mathbf{S}_2$ are sketching matrices. Imagine now that $\mathbf{A}_t$ was updated by a matrix $\mathbb{H}$ such that $\mathbf{A}_{t+1} = \theta\mathbf{A}_t + \eta\mathbf{H}$, where θ and η are scalars. Recomputing the sketches of $\mathbf{A}_{t+1}$ would result in

$$\begin{aligned}
\mathbf{Y}_{t+1} &= \mathbf{A}_{t+1}\mathbf{S}_1^\top = (\theta\mathbf{A}_t + \eta\mathbf{H})\,\mathbf{S}_1^\top = \theta\mathbf{A}_t\mathbf{S}_1^\top + \eta\mathbf{H}\mathbf{S}_1^\top = \theta\mathbf{Y}_t + \eta\mathbf{H}\mathbf{S}_1^\top,\\
\mathbf{W}_{t+1} &= \mathbf{S}_2\mathbf{A}_{t+1} \;= \mathbf{S}_2\,(\theta\mathbf{A}_t + \eta\mathbf{H}) \;= \theta\mathbf{S}_2\mathbf{A}_t + \eta\mathbf{S}_2\mathbf{H} \;= \theta\mathbf{W}_t + \eta\mathbf{S}_2\mathbf{H},
\end{aligned}$$

which corresponds to updating the sketches directly with $\mathbf{H}\mathbf{S}_1^\top$ and $\mathbf{H}\mathbf{S}_2$, respectively. When the `sketchyCGM` stopping criterion is satisfied, the algorithm computes a low-rank approximation to the full size matrix based on the updated sketches.

The `sketchyCGM` algorithm is shown in Algorithm 8. The vector of ratings $\mathbf{d}$ taken as input corresponds to $\mathcal{A}\mathbf{D}$ and is used to define the function f, while the index set E defines the operations with functions $\mathcal{A}$ and $\mathcal{A}^\star$. The algorithm uses scaled Gaussian sketching matrices and parameters $k = 2r+1$ and $l = 4r+3$ since this is found to provide a good trade-off between approximation quality and storage savings [Tro+17]. Note that the steps 13-16 closely resemble the procedure of Algorithm 7. A minor difference is the calculation of the orthonormal matrix using MATLAB's `orth` function, which is slower than the function `qr` but automatically truncates the basis to adjust for low rank in the input matrix.

By inspection of the algorithm, we find that most calculations are kept on vector form and the largest stored matrices are $\mathbf{W} \in \mathbb{R}^{l\times n}$ and $\boldsymbol{\Psi} \in \mathbb{R}^{l\times m}$. Furthermore, we also need to store a vector of length d, containing the known entries and as k and l

Algorithm 8 `sketchyCGM` algorithm

Input: Ratings $\mathbf{d} \in \mathbb{R}^d$, index set E, number of users m and movies n, target rank r, parameter α and optimality threshold ε

Output: $\mathbf{U} \in \mathbb{R}^{m\times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$ and $\mathbf{V} \in \mathbb{R}^{n\times r}$

```
 1: $k = 2r + 1$, $l = 4r + 3$                                     ▷ Initialise sketches
 2: $\mathbf{\Omega} = \texttt{randn}\,(n, k)$, $\mathbf{\Psi} = \texttt{randn}\,(l, m)$
 3: $\mathbf{Y} = \mathbf{0} \in \mathbb{R}^{m\times k}$, $\mathbf{W} = \mathbf{0} \in \mathbb{R}^{l\times n}$, $\mathbf{z} = \mathbf{0} \in \mathbb{R}^d$
 4: for $t = 0, 1, 2, \ldots$ do
 5:     $(\mathbf{u}, \mathbf{v}) = \texttt{svds}\,(\mathcal{A}^\star(\nabla f(\mathbf{z})), 1)$      ▷ Compute update direction
 6:     $\mathbf{h} = \mathcal{A}\left(-\alpha \mathbf{u}\mathbf{v}^\top\right)$
 7:     if $(\mathbf{z} - \mathbf{h})^\top \nabla f(\mathbf{z}) \le \varepsilon$ then break; end if    ▷ Check convergence
 8:     $\eta = \frac{2}{t+2}$
 9:     $\mathbf{z} = (1 - \eta)\,\mathbf{z} + \eta\mathbf{h}$                          ▷ Update sketches
10:     $\mathbf{Y} = (1 - \eta)\,\mathbf{Y} - \eta\,\alpha\mathbf{u}\left(\mathbf{v}^\top\mathbf{\Omega}\right)$
11:     $\mathbf{W} = (1 - \eta)\,\mathbf{W} - \eta\,\alpha\,(\mathbf{\Psi}\mathbf{u})\,\mathbf{v}^\top$
12: end for
13: $\mathbf{Q} = \texttt{orth}\,(\mathbf{Y})$                         ▷ Reconstruct low-rank approximation
14: $\mathbf{B} = (\mathbf{\Psi}\mathbf{Q})\,\backslash\mathbf{W}$
15: $[\mathbf{U_B}, \mathbf{\Sigma}, \mathbf{V}] = \texttt{svds}\,(\mathbf{B}, r)$
16: $\mathbf{U} = \mathbf{Q}\mathbf{U_B}$
```

depend linearly on r, we get total storage cost of $O\,(d + r(m + n))$. Observe that the storage savings of the `sketchyCGM` algorithm depend heavily on the target rank of $\mathbf{X}$ as this controls the sketching parameters k and l. When the true rank of $\mathbf{X}$ can be assumed small, excellent approximations can be achieved sacrificing little storage.

Even though the `sketchyCGM` algorithm has been motivated by the matrix completion problem, it should be noted that the method has a much broader range of usage. In fact, it can be used to solve any convex constrained optimisation problem as long as the objective function is smooth and the constraint a Schatten 1-norm constraint [Yur+17].

Experimental results. In order to test the performance of the `sketchyCGM` algorithm, we consider the benchmark MovieLens 100K data set [HK16]. This data set contains 100,000 ratings of 9066 movies by 671 unique users on a scale of 1 to 5 with increments of 0.5. This means that most users have lots of movies left to rate and it is these ratings we will seek to predict. In order to measure the quality of the matrix completion algorithm, the data set is split in two groups, one of which is used to train the algorithm while the other is used for evaluation. The split is made such that we train on 90% of the data and test on the remaining 10%.

To evaluate the algorithm we compute the test error using the squared loss function, resulting in the objective function

$$\sum_{(i,j)\in T} f_{\text{loss}}\left(\mathbf{X}_{ij}\right) = \sum_{(i,j)\in T} \frac{1}{2\,|T|}\left(\mathbf{X}_{ij} - \mathbf{D}_{ij}\right)^2,$$

where T is the index set for the entries in the test set.

Algorithm 8 takes as input the parameter α which makes sure that the minimiser of problem (3.7) has low rank. Choosing an appropriate value for this parameter is found to be quite important for both the `sketchyCGM` and regular CGM algorithm. We therefore perform a 2-fold cross-validation in order to find the optimal α. In the cross-validation, we evaluate the test error of the CGM algorithm after 1000 iterations and choose the value of α giving the lowest test error. For the squared loss function, we found $\alpha = 7500$ to be optimal when sweeping values of α from 3000 to 10000 in steps of 500.

Figure 3.9 shows the test error obtained from both algorithms using the optimal α

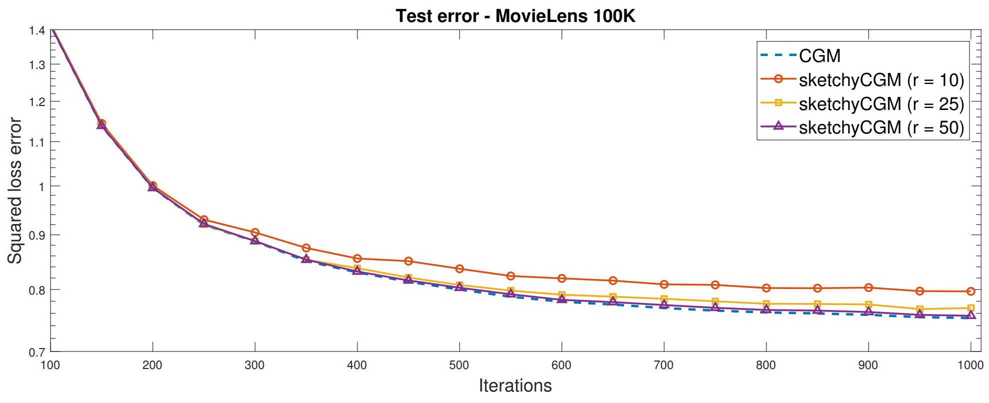


Figure 3.9: `sketchyCGM` and CGM test error on the MovieLens 100K data set with $\alpha = 7500$ and for three different target ranks.

parameter as a function of the number of CGM iterations. The `sketchyCGM` algorithm was run with three different target ranks: $r = 10$, $r = 25$ and $r = 50$.

The figure clearly shows that increasing the target rank also increases the quality of the approximation. With a target rank of $r = 50$, we find that the accuracy of the `sketchyCGM` algorithm is similar to that of the regular CGM algorithm while it only uses half the storage. It is worth noting that the largest matrix stored is three times smaller than $\mathbf{X}$. For $r = 10$ we use only an eighth of the total storage, and the largest matrix is more than 15 times smaller than $\mathbf{X}$.

3.3.2 CUR matrix decomposition

A natural way to encode information about m objects described by n features is using an $m \times n$ data matrix $\mathbf{A}$. An initial step for further investigation is often to construct a compressed representation of $\mathbf{A}$ that is easier to analyse, which is commonly achieved using a singular value decomposition [MD09]. In cases where the data matrix $\mathbf{A}$ is expected to have low rank, the SVD can be truncated to contain only the information pertaining to the r first singular values and provably provides the best rank r approximation to $\mathbf{A}$. This set of orthogonal basis vectors accounts for as much of the variance in the data as possible which is a very useful property exploited in for example explorative data analysis, feature extraction, classification and data visualisation [Bis06]. A related and frequently applied method is the Principal Component Analysis (PCA), where the first r principal directions are the eigenvectors corresponding to the r largest eigenvalues of the data covariance matrix.

One of the disadvantages of using the SVD to generate a basis becomes apparent when trying to analyse and understand the underlying physical factors of the basis vectors. The singular vectors are linear combinations of up to all the objects or features and thereby mathematical abstractions rather than explanatory variables with domain specific interpretations [MD09]. This section introduces an alternative matrix factorisation method called the CUR matrix decomposition. Here we decompose $\mathbf{A} = \mathbf{CUR}$, where $\mathbf{C}$ and $\mathbf{R}$ are matrices consisting of a small number of actual columns and rows of the data matrix, respectively. The matrices $\mathbf{C}$ and $\mathbf{R}$ can be seen as collections of basis vectors and allow for direct interpretation in the context of the data. The structure of the CUR matrix decomposition also entails other advantages, such as the preservation of input sparsity in $\mathbf{C}$ and $\mathbf{R}$, which can be a helpful property when it comes to ease of interpretation but certainly also in terms of storage costs. Note that the name CUR decomposition can be misleading, since the matrix product $\mathbf{CUR}$ in most cases will provide a low-rank approximation of $\mathbf{A}$ and not an exact decomposition.

Focusing initially on the matrix $\mathbf{C}$, the aim is to choose a small subset of columns from the $m \times n$ matrix $\mathbf{A}$ that approximately span the entire column space of $\mathbf{A}$. For a suitable matrix norm, we can view this as the minimisation problem

$$\min_{\mathbf{C},\mathbf{X}} \|\mathbf{A} - \mathbf{CX}\|\,,$$

which is known as the Column Subset Selection Problem (CSSP). The CSSP is provably NP-complete [Shi17], and a combinatorial approach where all possible subsets of n_c columns are generated takes $O\left(n^{n_c}\right)$ time [BMD09].

The complexity of the problem has inspired randomised approaches where columns are sampled with respect to some carefully chosen probability distribution, and this is were sketching makes its entrance. In [DK03], Drineas and Kannan provided an early algorithm for CUR decomposition where the probability distribution was based on the Euclidean lengths of the matrix columns and rows leading to additive error approximation guarantees. By using probabilities proportional to the leverage scores of the input matrix, relative error guarantees were achieved in [DMM08]. As described in Section 3.2.1, the leverage scores of $\mathbf{A}$ indicate to which part of the m-dimensional column space the singular value information of $\mathbf{A}$ is being dispersed. Similarly, the leverage scores of $\mathbf{A}^\top$ describe the influence of the columns of $\mathbf{A}$ on the best low-rank approximation. The main result of [DMM08] is the following.

Theorem 15 ([DMM08, Theorem 2])**.** *Given a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *and an integer* $r \ll \min\{m,n\}$*, there exists a randomised algorithm choosing in expectation* $n_c = O\left(r \ln r \ln(1/\delta)\,\varepsilon^{-2}\right)$ *columns and* $n_r = O\left(n_c \ln n_c \ln(1/\delta)\,\varepsilon^{-2}\right)$ *rows of* $\mathbf{A}$ *to construct* $\mathbf{C}$ *and* $\mathbf{R}$*, respectively, such that*

$$\|\mathbf{A} - \mathbf{CUR}\|_F \le (1+\varepsilon)\,\|\mathbf{A} - \mathbf{A}_r\|_F$$

for a suitable matrix $\mathbf{U}$*, with probability at least* $1-\delta$*. Here,* $\mathbf{A}_r$ *is the best rank* r *approximation to* $\mathbf{A}$*.*

Our implementation of a CUR decomposition algorithm is based on a slightly weaker result discussed in a later article by Mahoney [MD09]. Here, an algorithm is presented for which the guarantee of Theorem 15 holds with a factor of $(2+\varepsilon)$ instead of the stated $(1+\varepsilon)$. We have chosen to implement this version of a CUR decomposition algorithm as it is computationally simpler and because the matrices $\mathbf{C}$ and $\mathbf{R}$ consist of actual columns and rows of $\mathbf{A}$, unlike the algorithm corresponding to Theorem 15. This gives us an uncomplicated algorithm with results that are easy to interpret and understand in the original context. Furthermore, it should be noted that the speed and error guarantees of our implementation do not compare with recent algorithms, such as the one presented in [BW17] where the dependence of the number of rows and columns on the error parameter ε is optimal.

The implemented CUR matrix decomposition algorithm is seen in Algorithm 9. Note that the number of rows and columns are chosen in expectation, and the dimension of output matrices are therefore not guaranteed to match the input choices n_c and n_r. The outputs specified in the algorithm reflect the expected dimensions of the decomposition factors

Algorithm 9 CUR matrix decomposition

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, rank parameter $r$ and number of columns $n_c$ and rows $n_r$ to be
    sampled in expectation
Output: $\mathbf{C} \in \mathbb{R}^{m\times n_c}$, $\mathbf{U} \in \mathbb{R}^{n_c\times n_r}$ and $\mathbf{R} \in \mathbb{R}^{n_r\times n}$
1: Compute a rank $r$ SVD of $\mathbf{A}$ to obtain $\mathbf{A} = \mathbf{U\Sigma V}^\top$
2: For $j = 1, \dots, n$ and $i = 1, \dots, m$, compute the leverage scores
       $$q_j^c = \frac{1}{r}\left\|\mathbf{V}_{(j)}\right\|_2^2 \qquad \text{and} \qquad q_i^r = \frac{1}{r}\left\|\mathbf{U}_{(i)}\right\|_2^2$$
3: Construct $\mathbf{C}$ and $\mathbf{R}$ by keeping the $j$-th column of $\mathbf{A}$, with probability
    $\min\left(1, n_c q_j^c\right)$, and $i$-th row of $\mathbf{A}$, with probability $\min\left(1, n_r q_i^r\right)$
4: Calculate $\mathbf{U} = \mathbf{C}^\dagger \mathbf{A} \mathbf{R}^\dagger$
```

In Algorithm 9, sketching primarily appears in the sampling of the carefully chosen columns and rows of the input matrix. This means that the sketching methods based on projection are not applicable. The leverage score sampling method is not used directly for two reasons. First of all, we assume that the input matrix is approximately low-rank, and we can therefore approximate an orthonormal basis using only few singular vectors. Secondly, a singular value decomposition is efficient when the leverage scores for both $\mathbf{A}$ and $\mathbf{A}^\top$ are required, since we obtain both the left and right singular vectors of the matrix. In regards to the complexity of the algorithm, the computational bottleneck is in the first step where a singular value decomposition is performed. As a way of speeding up the CUR decomposition, one could approximate the leverage scores for example by direct application of the randomised SVD of Algorithm 7.

Experimental results. The main goal of this section is to test the advantage gained in terms of interpretability when considering factors of a CUR decomposition rather than a regular SVD or PCA. For the interpretation of the CUR decomposition to be meaningful, the product of the factors should provide a good approximation of the original matrix relative to the best rank r approximation of that matrix. We therefore consider error measures based on the spectral and Frobenius norm of the residual matrix

$$\mathbf{A}_{\text{res}}^{(\xi)} = \frac{\mathbf{A} - \mathbf{CUR}}{\|\mathbf{A} - \mathbf{A}_r\|_\xi} \qquad \text{for} \quad \xi \in \{2, F\}.$$

As the time complexity of Algorithm 9 is not our primary focus, the computation times are not included in the reported results.

We first analyse the performance of Algorithm 9 with respect to low rank approximation for synthetic data sets where an assumption of low rank is reasonable. Table 3.8 displays some of our findings for different numbers of rows and columns included in the decomposition.

Table 3.8 shows that it is possible to obtain approximations that are similar in accuracy to the best rank r approximation using a decomposition consisting of a relatively small number of actual columns and rows. Note that choosing the input parameters n_c and n_r such that the rank of the decomposition is somewhat larger than r, improves the approximation quality significantly. In fact, a CUR decomposition with $4r+1$ columns and rows approximates the `polydecay` matrix better than the best rank r approximation for the chosen problem size.

To investigate the interpretability of the CUR matrix decomposition compared to

Table 3.8: Experimental results for CUR matrix decomposition with an input matrix of size $m = n = 2^{11}$, and target rank $r = 5$. The parameters n_c and n_r are the expected number of sampled columns and rows used for constructing $\mathbf{C}$ and $\mathbf{R}$, respectively. The error terms are based on the residual matrix $\mathbf{A}_{\text{res}}^{(\xi)} = (\mathbf{A} - \mathbf{CUR}) \,/\, \|\mathbf{A} - \mathbf{A}_k\|_\xi$.

		`polydecay`		`cond10`	
n_c	n_r	$\|\mathbf{A}_{\text{res}}^{(2)}\|_2$	$\|\mathbf{A}_{\text{res}}^{(F)}\|_F$	$\|\mathbf{A}_{\text{res}}^{(2)}\|_2$	$\|\mathbf{A}_{\text{res}}^{(F)}\|_F$
$r+1$	$r+1$	2.2825	1.6766	1.0584	1.0548
$2r+1$	$2r+1$	1.2617	1.2346	1.0568	1.0476
$4r+1$	$2r+1$	1.1891	1.1248	1.0518	1.0391
$4r+1$	$4r+1$	0.7579	0.9369	1.0378	1.0239

a PCA, consider the `spambase` data set as introduced in Section 3.2.1. The rows of the data matrix $\mathbf{A}$ represent the collection of emails, the columns a set of recorded words and each matrix entry the frequency of the word in the given email. It is tempting to seek explanations or causalities in terms of the domain from which the data is drawn when computing a decomposition, such as relating certain words to a specific structure or pattern in the set of emails. As will become apparent, the CUR decomposition is a handy tool for exploring connections in the data set and suggesting interesting relationships, something which is hard to extract from a PCA as mentioned earlier.

We initially compute a PCA of the matrix $\mathbf{A}$, obtaining a set of eigenvalues and corresponding principal directions that constitute a transformation of $\mathbf{A}$ to a basis maximising the variance of the data. The top plot in Figure 3.10 shows the first five principal directions. Since these are made up by linear combinations of the features, we cannot relate the maximisation of variance with specific word frequencies.

A CUR decomposition of $\mathbf{A}$ is now computed. As the columns contained in $\mathbf{C}$ can be used as basis vectors to approximate the entire data matrix, they must also describe a significant part of the variance in the data. The selection of columns is based on the column leverage scores shown in the bottom plot of Figure 3.10. These provide an overview of columns expected to make up the matrix $\mathbf{C}$.

The shown leverage scores indicate that five features are far more influential than

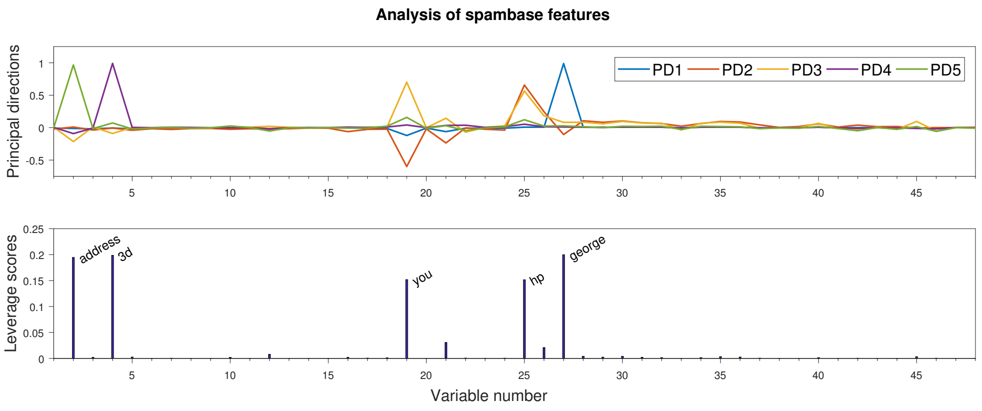


Figure 3.10: *Top plot*: First 5 principal directions of the `spambase` data set with $m = 4437$ observations and $n = 48$ features. *Bottom plot*: Leverage scores of the $n = 48$ features with the top five labelled explicitly.

the others. These features correspond directly to the words "address", "3d", "you", "hp" and "george". The first principal direction may also have a seemingly significant spike for the word "george", but the importance of all the smaller values that are spread out across the rest of the words in defining the principal direction cannot be immediately determined.

To understand why "george" explains some of the variance in the data, we must consider the context of the data set and how it was collected. The `spambase` data set is based on a donation of emails from George Forman, a former analyst at Hewlett-Packard (HP) Labs, and the considered emails include both non-spam emails, such as personal correspondence and filed work, and spam emails such as advertisements and other junk mail [DK17]. Coupling this background information with the observed leverage scores suggests that the words "george" and "hp" might be indicators of personal, non-spam emails.

As we are in possession of the class labels for each data point, we can plot the frequencies of the word "george" and "hp" based on the classes spam and non-spam as displayed in Figure 3.11. In total, there are 1798 spam emails and 2639 non-spam emails. We can also calculate the information gain statistic for each feature i as $|f_i^s - f_i^n|$, where f_i^s and f_i^n are the mean frequencies of word i in emails of class spam and non-spam, respectively. This supervised metric measures the discrepancy

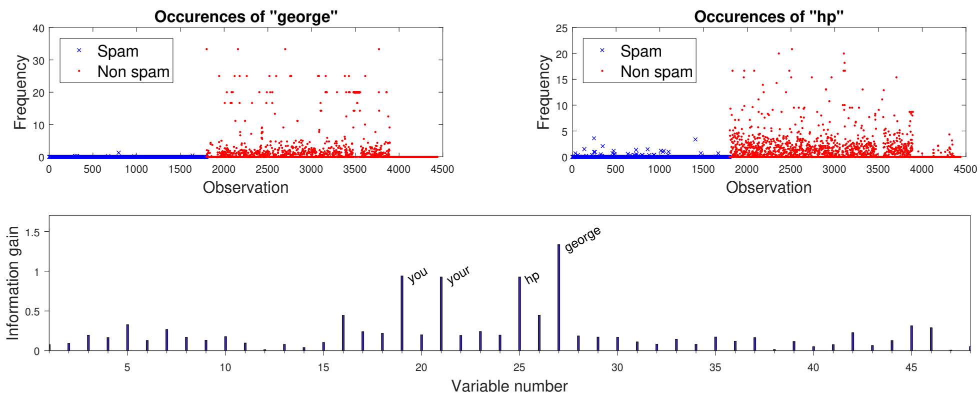


Figure 3.11: *Top plots*: the frequency of the words "george" and "hp" for each of the 4437 emails separated into spam and non-spam classes. *Bottom plot*: information gain for each of the 48 feature words with the top four labelled explicitly.

between the word frequencies of each class and is shown for the 48 words in Figure 3.11. The information gain is not normalised according to the number of emails in which each word appears, and hence words that are rarely used in the emails are unlikely to have a large information gain, even if they are used significantly more in a certain class of emails.

The information gain for “george”, “hp” and “you” match the high leverage scores of Figure 3.10, whereas “address” and “3d” have low information gains. “george” and “hp” both appear significantly more in non-spam emails while “you” is found more frequently in spam emails. The word “3d” only appears in 47 mails in total, with only 8 of these in non-spam emails, while “address” is found in 625 spam emails compared to 273 non-spam emails. The low information gain of “3d” is largely due to the rarity of the word, while the low information gain of “address” suggests that this term is perhaps not indicative of spam or non-spam emails, but rather reflects some other underlying properties of the data set.

3.3.3 Optimisation with generalised Tikhonov regularisation

The Tikhonov regularised least squares problem appears in many fields of applied mathematics under many different names: in statistics it is usually referred to as ridge or prior regression while in machine learning it often features as weight decay or parameter shrinkage. For an $m \times n$ model matrix $\mathbf{A}$ and response vector $\mathbf{b}$, the simplest form of the Tikhonov regularised problem is

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda^2 \|\mathbf{x}\|_2^2 . \tag{3.11}$$

This is an ordinary least squares problem augmented with a simple regularisation term to avoid overfitting the solution to the data observations by penalising solutions with large norms. This is especially useful in ill-conditioned inverse problems where the solution is very sensitive to errors in the data. If a priori assumptions about the noise in the measurements $\mathbf{b}$ or about the solution $\mathbf{x}$ are available, these can be incorporated in the least squares problem through the corresponding covariance matrices. This leads to the generalised Tikhonov regularisation problem

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_{\mathbf{\Sigma}_{\mathbf{b}}^{-1}}^2 + \|\mathbf{x}\|_{\mathbf{\Sigma}_{\mathbf{x}}^{-1}}^2 , \tag{3.12}$$

where $\mathbf{\Sigma}_{\mathbf{b}}$ and $\mathbf{\Sigma}_{\mathbf{x}}$ are the covariance matrices of $\mathbf{b}$ and $\mathbf{x}$, respectively. This corresponds to a priori assumptions that $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_{\mathbf{x}})$ and $\mathbf{b} \sim \mathcal{N}(\mathbf{A}\mathbf{x}, \mathbf{\Sigma}_{\mathbf{b}})$, with $\mathbf{b}$

conditioned on $\mathbf{A}$ and $\mathbf{x}$. The optimal solution to the problem is given by the closed formula [Han98, p. 101]

$$\hat{\mathbf{x}} = \left(\mathbf{A}^\top \boldsymbol{\Sigma}_\mathbf{b}^{-1} \mathbf{A} + \boldsymbol{\Sigma}_\mathbf{x}^{-1}\right)^{-1} \mathbf{A}^\top \boldsymbol{\Sigma}_\mathbf{b}^{-1} \mathbf{b}. \tag{3.13}$$

Calculating this solution involves computing the inverses of the $m \times m$ matrix $\boldsymbol{\Sigma}_\mathbf{b}$ and the $n \times n$ matrices $\boldsymbol{\Sigma}_\mathbf{x}$ and $\left(\mathbf{A}^\top \boldsymbol{\Sigma}_\mathbf{b}^{-1} \mathbf{A} + \boldsymbol{\Sigma}_\mathbf{x}^{-1}\right)$, as well as performing the matrix-matrix multiplications in $\mathbf{A}^\top \boldsymbol{\Sigma}_\mathbf{b}^{-1} \mathbf{A}$. This makes the calculation very expensive when m and n are large.

A clever way of avoiding the computation of $\boldsymbol{\Sigma}_\mathbf{x}^{-1}$ is to use the Woodbury matrix inversion lemma[3] to rewrite

$$\left(\mathbf{A}^\top \boldsymbol{\Sigma}_\mathbf{b}^{-1} \mathbf{A} + \boldsymbol{\Sigma}_\mathbf{x}^{-1}\right)^{-1} = \boldsymbol{\Sigma}_\mathbf{x} - \boldsymbol{\Sigma}_\mathbf{x} \mathbf{A}^\top \left(\boldsymbol{\Sigma}_\mathbf{b} + \mathbf{A} \boldsymbol{\Sigma}_\mathbf{x} \mathbf{A}^\top\right)^{-1} \mathbf{A} \boldsymbol{\Sigma}_\mathbf{x}. \tag{3.14}$$

However, bypassing the $n \times n$ inverse comes at the expense of a new $m \times m$ inverse. The gain or loss of this is obviously dependent on the problem size.

Now assume the symmetric covariance matrix $\boldsymbol{\Sigma}_\mathbf{x}$ is well-approximated by a matrix of rank $r \ll n$, i.e. that $\boldsymbol{\Sigma}_\mathbf{x} \approx \mathbf{U}\mathbf{U}^\top$ for some $\mathbf{U} \in \mathbb{R}^{n \times r}$. This implies that $\boldsymbol{\Sigma}_\mathbf{x}$ is close to rank-deficient, and hence the calculation of the inverse $\boldsymbol{\Sigma}_\mathbf{x}$ is prone to large numerical errors. If $\boldsymbol{\Sigma}_\mathbf{x}$ is truly rank-deficient, then problem (3.12) is not well-defined. Consider instead the change of variable $\mathbf{x} = \mathbf{U}\mathbf{y}$, where $\mathbf{y} \in \mathbb{R}^r$ and $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. First, we observe that the properties of $\mathbf{x}$ are indeed captured by this change of variable since

$$\begin{aligned}
&\mathbb{E}[\mathbf{x}] = \mathbb{E}[\mathbf{U}\mathbf{y}] = \mathbf{U}\mathbb{E}[\mathbf{y}] = \mathbf{0} \\
&\mathbb{E}\left[\mathbf{x}\mathbf{x}^\top\right] = \mathbb{E}\left[\mathbf{U}\mathbf{y}\mathbf{y}^\top \mathbf{U}^\top\right] = \mathbf{U}\mathbb{E}\left[\mathbf{y}\mathbf{y}^\top\right]\mathbf{U}^\top = \mathbf{U}\mathbf{U}^\top.
\end{aligned}$$

Using this, we can transform problem (3.12) to a problem in the variable $\mathbf{y}$ as

$$\min_{\mathbf{y}} \|\mathbf{A}\mathbf{U}\mathbf{y} - \mathbf{b}\|_{\boldsymbol{\Sigma}_\mathbf{b}^{-1}}^2 + \|\mathbf{y}\|^2, \tag{3.15}$$

and then recover an approximate solution to (3.12) by $\hat{\mathbf{x}} = \mathbf{U}\hat{\mathbf{y}}$, where $\hat{\mathbf{y}}$ is the minimiser of (3.15). The main advantage of this approach is identified by looking at the closed form solution formula

$$\hat{\mathbf{y}} = \left(\mathbf{U}^\top \mathbf{A}^\top \boldsymbol{\Sigma}_\mathbf{b}^{-1} \mathbf{A}\mathbf{U} + \mathbf{I}\right)^{-1} \mathbf{A}^\top \boldsymbol{\Sigma}_\mathbf{b}^{-1} \mathbf{b}. \tag{3.16}$$

[3]For conforming matrices $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{V}$ and $\mathbf{Z}$, the Woodbury matrix identity states that

$$(\mathbf{X} + \mathbf{V}\mathbf{Y}\mathbf{Z})^{-1} = \mathbf{X}^{-1} - \mathbf{X}^{-1}\mathbf{V}\left(\mathbf{Y}^{-1} + \mathbf{Z}\mathbf{X}^{-1}\mathbf{V}\right)^{-1}\mathbf{Z}\mathbf{X}^{-1}.$$

Besides the necessary inversion of $\mathbf{\Sigma_b}$, the expression only contains an $r \times r$ inverse and matrix multiplications between matrices, with at least one dimension of size r. This represents a significant reduction in computational complexity.

Naturally, for this method to provide decent results, the covariance matrix $\mathbf{\Sigma_x}$ should be well-approximated by a low-rank matrix. In addition, for the method to be advantageous in practice, we need a fast way of computing a suitable decomposition of $\mathbf{\Sigma_x}$. Here, we can utilise the randomised singular value decomposition introduced in Section 3.2.3 for a fast approximation of the singular values and vectors, which can then be combined to form the matrix $\mathbf{U}$ used above.

Establishing exact expressions for the covariance matrices $\mathbf{\Sigma_b}$ and $\mathbf{\Sigma_x}$ requires very intimate knowledge of the problem, variables and noise, and these matrices are therefore often estimated in some way. Given a number of samples from the considered distributions, the sample covariance matrix can be calculated as in the example of Section 3.1.1. If $\mathbf{X}$ is a matrix with N columns each corresponding to a data point, the unbiased sample covariance matrix is given by $\frac{1}{N-1}\left(\mathbf{X}-\bar{\mathbf{X}}\right)\left(\mathbf{X}-\bar{\mathbf{X}}\right)^\top$, where $\bar{\mathbf{X}}$ contains the sample means. The sample covariance matrix is sensitive to outliers and a lot of samples can be necessary to obtain a good approximation of the true underlying covariance matrix. Information about the underlying distribution can be used to ensure that the estimated covariance matrix reflects important properties of this distribution. This could for example be sparsity, as considered in [VKO00], or a low-rank assumption as discussed here.

Algorithm 10 shows how to approximate a solution to (3.12) using sketching as a computational tool. If the true covariance matrix is not provided, the approximate matrix multiplication of Algorithm 1 is applied to estimate the sample covariance matrix. The matrix $\mathbf{U}$ is obtained through use of the randomised singular value decomposition of Algorithm 7.

Experimental results. To investigate the computational advantages of the described sketching method in a practical setting, we construct a synthetic example.

Initially, we form a random model matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ with each entry drawn from the standard normal distribution. We then generate a covariance matrix $\mathbf{\Sigma_x}$ using the locally periodic kernel function introduced in Section 3.1.1, again with parameters $\alpha = 1$ and $T_{\text{period}} = \frac{1}{5}$. Such a choice of $\mathbf{\Sigma_x}$ is often numerically rank-deficient, which poses a problem for the formulation (3.12). To avoid large numerical errors when computing the inverse of $\mathbf{\Sigma_x}$, we adjust the covariance matrix by adding small values of γ to the diagonal, and use the matrix $\mathbf{\Sigma_x} + \gamma\mathbf{I}$ as the underlying covariance matrix.

Algorithm 10 Generalised Tikhonov regularisation

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^{m}$, $\boldsymbol{\Sigma}_\mathbf{b} \in \mathbb{R}^{m\times m}$, covariance matrix $\boldsymbol{\Sigma}_\mathbf{x} \in \mathbb{R}^{n\times n}$ or set of
    samples $\mathbf{X} \in \mathbb{R}^{n\times N}$, sketching parameters $k_{\text{mult}}$ and $k_{\text{rsvd}}$, and target rank $r$
Output: Approximate solution $\hat{\mathbf{x}}$ to problem (3.12)
1: if covariance matrix not provided then
2:     Estimate the sample covariance $\boldsymbol{\Sigma}_\mathbf{x}$ using Algorithm 1 with parameter $k_{\text{mult}}$
3: end if
4: Use Algorithm 7 with $k_{\text{rsvd}}$ to approximate $\boldsymbol{\Sigma}_\mathbf{x} \approx \mathbf{U}_r\mathbf{D}_r\mathbf{V}_r^\top$
5: Form $\mathbf{U} = \mathbf{U}_r\mathbf{D}_r^{1/2}$, where $\left(\mathbf{D}_r^{1/2}\right)_{ij} = \sqrt{(\mathbf{D}_r)_{ij}}$
6: Calculate $\hat{\mathbf{y}}$ using Equation (3.16) and compute $\hat{\mathbf{x}} = \mathbf{U}\hat{\mathbf{y}}$
```

This simply corresponds to added penalisation of large values in $\mathbf{x}$.

The noise covariance matrix is chosen as a scaled unit matrix such that $\boldsymbol{\Sigma}_\mathbf{b} = \sigma_\mathbf{b}^2\mathbf{I}$, corresponding to an assumption of the noise being uncorrelated and sharing the standard deviation $\sigma_\mathbf{b}$. A target solution vector $\mathbf{x}_{\text{target}}$ is then drawn from the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\mathbf{x})$ and a measurement vector $\mathbf{b}$ is drawn from $\mathcal{N}(\mathbf{A}\mathbf{x}, \boldsymbol{\Sigma}_\mathbf{b})$. We will assume that the covariance matrix $\boldsymbol{\Sigma}_\mathbf{x}$ is not known explicitly, but that we are in possession of N samples from the distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\mathbf{x})$. The samples are collected in the matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]$, which is used to calculate the sample covariance matrix $\widehat{\boldsymbol{\Sigma}}_\mathbf{x}$.

The problem is now to approximate $\mathbf{x}_{\text{target}}$ given $\mathbf{A}$, $\mathbf{b}$, $\sigma_\mathbf{b}$ and $\mathbf{X}$. We calculate the following five approximate solutions:

$\mathbf{x}_{\text{direct}}$ is as in Equation (3.13) with the sample covariance matrix $\widehat{\boldsymbol{\Sigma}}_\mathbf{x}$ instead of the true covariance matrix.

$\mathbf{x}_{\text{woodbury}}$ is obtained by inserting Equation (3.14) in Equation (3.13), again using the sample covariance matrix.

$\mathbf{x}_{\text{tikhonov}}$ is the solution to the simple Tikhonov regularised problem (3.11). Note that this method uses a regularisation parameter λ instead of a covariance matrix. For this problem, $\lambda = 1$ appears to be an appropriate choice.

$\mathbf{x}_{\text{svd}}$ is based on the assumption of $\boldsymbol{\Sigma}_\mathbf{x}$ having low rank, with the solution calculated as in Equation (3.16) but with the decomposition $\widehat{\boldsymbol{\Sigma}}_\mathbf{x} = \mathbf{U}\mathbf{U}^\top$ found using MATLAB's `svd` function.

$\mathbf{x}_{\text{sketch}}$ corresponds to the use of Algorithm 10 with approximate matrix multiplication using the sketch parameter $k_{\text{mult}} = n$, and a randomised singular value decomposition with parameter $k_{\text{rsvd}} = 2r + 1$. In both of these cases, a sparse projection matrix is used to construct the sketches.

Table 3.9 shows the results obtained for the special case where the model problem is of size $m = 6000$ and $n = 2000$. We assume that we are given $N = 4n$ samples and that the standard deviation of the measurements is $\sigma_{\mathbf{b}} = 0.1$. As previously mentioned, $\mathbf{\Sigma_x}$ is numerically rank-deficient with only the largest 104 singular values above $2 \cdot 10^{-10}$, causing MATLAB's `rank` function to claim that the rank is 104. Consequently, we set our rank regularisation parameter $\gamma = 10^{-10}$ in order to avoid singularity issues. This also motivates setting the rank parameter used in both $\mathbf{x}_{\text{svd}}$ and $\mathbf{x}_{\text{sketch}}$ to $r = 100$, as this should be sufficient for a good low-rank approximation.

Two sample covariance matrices are constructed: $\widehat{\mathbf{\Sigma}}_{\mathbf{x}}$ is the actual sample covariance matrix, while $\widehat{\mathbf{\Sigma}}_{\mathbf{S}}$ is a sketched version, used in the computation of $\mathbf{x}_{\text{sketch}}$. With $N = 4n$, we find that the relative spectral norm error between $\widehat{\mathbf{\Sigma}}_{\mathbf{x}}$ and the underlying true covariance matrix, $\mathbf{\Sigma_x}$, is 0.0265. Similarly, the error between $\widehat{\mathbf{\Sigma}}_{\mathbf{S}}$ and $\mathbf{\Sigma_x}$ is 0.0548. The relative error between $\widehat{\mathbf{\Sigma}}_{\mathbf{S}}$ and $\widehat{\mathbf{\Sigma}}_{\mathbf{x}}$ is 0.0362. These errors are similar in size to those obtained in Section 3.1.1, and the performance of the methods will help put the magnitude of the approximation errors in perspective.

The results of Table 3.9 show that all but the simple Tikhonov regularised solution yield low solution and relative errors, but the objective values and gain factors reflect significant differences in the approaches.

The solution $\mathbf{x}_{\text{woodbury}}$ is theoretically the same solution as $\mathbf{x}_{\text{direct}}$, however, we

Table 3.9: Generalised Tikhonov regularisation problem for a model of size $m = 6000$ and $n = 2000$, and with parameters $\sigma_{\mathbf{b}} = 0.1$ and $r = 100$. The objective value is given by the term minimised in (3.12) using the true covariance matrix, while the solution error and relative error are with respect to the target solution $\mathbf{x}_{\text{target}}$, used to generate $\mathbf{b}$.

Solution type	Objective value	Solution error	Relative error	Gain factor
$\mathbf{x}_{\text{direct}}$	5975.62	$1.03 \cdot 10^{-3}$	$2.33 \cdot 10^{-4}$	1
$\mathbf{x}_{\text{woodbury}}$	5982.55	$1.03 \cdot 10^{-3}$	$2.33 \cdot 10^{-4}$	0.26
$\mathbf{x}_{\text{tikhonov}}$	-	$6.75 \cdot 10^{-2}$	$1.53 \cdot 10^{-3}$	1.78
$\mathbf{x}_{\text{svd}}$	5957.56	$1.03 \cdot 10^{-3}$	$2.33 \cdot 10^{-4}$	0.71
$\mathbf{x}_{\text{sketch}}$	6040.40	$1.03 \cdot 10^{-3}$	$2.33 \cdot 10^{-4}$	5.81

see that this is not quite the case in practice, with the objective value slightly larger for the $\mathbf{x}_{\text{woodbury}}$ solution. This difference is due to issues with numerical cancellation and accuracy especially present when $\sigma_{\mathbf{b}}$ is small. To see this, consider the term $\left(\boldsymbol{\Sigma}_{\mathbf{b}} + \mathbf{A}\boldsymbol{\Sigma}_{\mathbf{x}}\mathbf{A}^\top\right)^{-1}$. When $\sigma_{\mathbf{b}}$ is small and $\mathbf{A}$ a tall matrix, this inversion will be numerically rank-deficient and thus numerical precision problems arise. Furthermore, we note that $\mathbf{x}_{\text{woodbury}}$ is slower to compute, since the required $m \times m$ inverse dominates the computation time, when m is significantly larger than n.

The relatively large error of $\mathbf{x}_{\text{tikhonov}}$ shows that neglecting the information of the underlying covariance matrix $\boldsymbol{\Sigma}_{\mathbf{x}}$ leads to poorer performance. Since $\mathbf{x}_{\text{tikhonov}}$ is the solution to problem (3.11) and not (3.12), the objective value is not stated. The method is faster than the direct method, since the inverse of $\boldsymbol{\Sigma}_{\mathbf{x}}$ is not calculated and hence for problems where the covariance of $\mathbf{x}$ is not as important, the simple Tikhonov regularised solution might be a suitable alternative.

Interestingly, the low-rank approximation of the sample covariance matrix, as used in the solution $\mathbf{x}_{\text{svd}}$, leads to a lower objective value than the direct solution method. This is in part due to the fact that the underlying covariance matrix is approximately of rank $r = 100$, and hence in the calculation of $\mathbf{x}_{\text{svd}}$, we have incorporated knowledge of the true covariance matrix instead of relying entirely on the sample covariance. Despite the savings obtained in the matrix multiplications and smaller inverses, the method is slightly slower than the direct method due to the singular value decomposition of $\widehat{\boldsymbol{\Sigma}}_{\mathbf{x}}$.

The solution $\mathbf{x}_{\text{sketch}}$ is a faster alternative, as it uses sketching to speed up both the calculation of the sample covariance matrix and the singular value decomposition. In fact, Table 3.9 shows that it is almost 6 times faster than the direct method. The solution error and relative error are on par with the methods that perform best, even though the objective value is slightly higher. Since the sample covariance matrix is already an estimate of the true covariance matrix, approximating the sample covariance is not expected to have much impact on the solution quality, which is also reflected by obtained results.

If a covariance matrix is provided when using Algorithm 10, the gain factor increases even further, since the gain factor of the randomised SVD of Algorithm 7 is much higher than the gain factor of the approximate matrix multiplication in Algorithm 1. In the above example, a gain factor of around ten is obtained, with an error almost identical to that of the solution $\mathbf{x}_{\text{svd}}$ based on the true singular value decomposition of $\widehat{\boldsymbol{\Sigma}}_{\mathbf{x}}$.

CHAPTER 4

Assessment of applicability

The applications of Chapter 3 showed how sketching can be used in various situations from basic linear algebra operations to more advanced optimisation problems. In the current chapter, we assess the use of sketching more generally by extracting and analysing the key findings of the previous sections. First, we discuss the computational aspects involved in implementing sketching and relevant to the experiments performed in this report. We then focus on the accessibility of sketching as a resource across different areas of applied mathematics and computer science. Finally, we consider additional applications of sketching showing the potential advantages and pitfalls of the method and suggest topics for further research.

4.1 Computational aspects

Chapter 3 showed that sketching can be a useful resource leading to notable advantages like reduced complexity, lower storage costs and increased interpretability as illustrated by the reported experiments. However, it is also clear that there are many computational aspects to consider when implementing sketching and analysing the benefits of its use. The quality of solutions, efficiency of algorithms and ease of implementation all play a role in determining the performance of sketching in a given problem setting.

The choice of the sketching parameter k controls both the quality of approximation and the amount of resources used in the computation. As the MRI image reconstruction problem of Section 3.1.3 and approximation of leverage scores in Section 3.2.1 illustrate, when k is large we obtain good approximations but save little in terms of computation time, while when k is small, we achieve significant savings in time or storage at the price of poorer approximation quality. The sketching parameter therefore represents a trade-off between approximation quality and potential computational advantages. For a problem involving a matrix of size $m \times n$, we found that setting k proportional to the lower dimension n with a small proportionality constant often seemed to offer a reasonable balance between the two properties for all of the sketching methods.

Comparing with the bounds summarised in Section 2.2, this is an important observation, as this choice of k rarely suffices to theoretically ensure any guarantees for the quality of approximation. This also means that one needs to use sketching with caution and possibly take measures to validate results such as repeated trials and using knowledge and experience within the field of application.

Our experiments show that the performance of the various sketching methods depends heavily on the problem at hand. Here, performance is referred to as the approximation quality together with the respective gain factor. Often the quality of approximation was comparable between sketching methods but the gain factor very different. Generally, Gaussian projection was slow since the construction of the sketch is based on a large matrix-matrix multiplication. Similarly, the leverage score sampling method rarely provided significant savings in time despite the approximation rather than exact computation of the leverage scores. The subsampled randomised Hadamard transform often achieved decent gain factors, whilst we found the sparse projection method to be fastest in almost any application. Theoretically, this method needs a larger k than its counterparts to guarantee a similar approximation error but in practice this seemed negligible, and the fact that it can be applied in input sparsity is also a significant advantage.

The leverage score sampling method differs from the other presented sketching methods in that it is both a sampling method and input-dependent. Basing the sampling probabilities on leverage scores leads to many nice theoretical properties as described in Section 2.1.1, but approximating these in a fast manner is not easily done for all input sizes. When the input matrix is almost square, using Algorithm 4 is only slightly faster than calculating the exact leverage scores as reflected by the small gain factor obtained in the experiments for approximating a k-dimensional basis of a $2^{10} \times 2^{12}$ matrix in Section 3.1.3. One might therefore consider other sampling probabilities that do not require a complete or approximate QR factorisation of the input matrix. Probabilities based on the Euclidean length of rows or columns have been used previously, for example in [DKM06] for approximate matrix multiplication, an application for which the leverage score sampling method did not provide a computational advantage in Section 3.1.1.

The simplest sampling scheme is to select rows or columns according to a uniform distribution. This is no longer an importance sampling as the method is now independent of the input matrix and does not rely on any underlying assumptions. Naturally, sampling uniformly is very fast and can in a number of cases yield decent results as the motivating example in Chapter 1 demonstrates. In many of the synthetic experiments considered throughout Chapter 3, a uniform sampling sketching method gave similar accuracy results as the four tested sketching methods.

It is important to stress, however, that simply sampling uniformly can fail heavily for certain inputs. In similar fashion to Chapter 1, [AMT10] demonstrated the potential pitfalls for a matrix with one very large leverage score when an orthonormalising transformation matrix is created through uniform sampling and used as a preconditioner in the `lsqr` function, as in Algorithm 6. The system involving the sketched preconditioner becomes so ill-conditioned that `lsqr` fails to run. The lack of theoretical guarantees attainable for uniform sampling as a sketching method is the reason for not including the method explicitly in our experiments.

Throughout the analysis, we have used the number of flops to quantify the speed of the numerical algorithms. This often serves as a good starting point for comparisons with respect to computational complexity, however, it can also be misleading and should in some cases be combined with other performance metrics. In the following, we present two cases where the number of flops does not fully reflect the complexity and advantages of the introduced algorithms.

First, some algorithms admit for better exploitation of computer architecture than others. Consider for example algorithms based on the Gaussian projection method. For an $m \times n$ matrix $\mathbf{A}$ and $k \times m$ matrix $\mathbf{S}$, constructing the sketch $\mathbf{SA}$ is an operation of complexity $O(kmn)$. However, since the i-th column of the sketch depends only on the i-th column of $\mathbf{A}$, we can split the calculations across multiple processors. Each processor is assigned a number of columns of $\mathbf{A}$ and executes the sketching operation for these, after which the outputs from all the used processors are collected to form $\mathbf{SA}$. The processors do not necessarily need to access the fully formed sketching matrix $\mathbf{S}$, but can be passed a suitable seed for a random number generator so that a copy of $\mathbf{S}$ can be created locally.

Though far from being a unique property of randomised algorithms, it is a general feature of such. In [HMT11], the authors go so far as to call randomised algorithms "embarrassingly parallelisable". Parallelisation is often performed implicitly in MATLAB, e.g. in the standard matrix multiplication operation which is based on LAPACK's level 3 Basic Linear Algebra Subprograms (BLAS). Our implementations are not optimised to take advantage of parallel computing in all aspects, making it hard to compare the flops and computation time with that of built-in MATLAB functions.

Secondly, in problems where the data does not fit in fast memory, the computational costs are often dominated by the passes over the data. Pass-efficiency, which is a measure of the number of times data passes through memory, can therefore also be an important metric when assessing performance of algorithms. As an example where the randomised algorithms can be used to optimise pass-efficiency, consider the randomised SVD in Algorithm 7. The classical SVD requires a number of passes over the data matrix $\mathbf{A}$ corresponding to the rank of $\mathbf{A}$ [HMT11], whereas the randomised SVD requires only two passes over the data. The algorithm can even be modified to yield a single-pass algorithm as described in Section 3.2.3, which simply involves a second sketch of the input matrix and approximately solving the generalised least squares problem of step 2 in Algorithm 7. Further details can be found in [Tro+17].

The ideas of the pass-efficient algorithms can also be extended to situations where the data is revealed column by column or in a stream of updates, as is the case in the `sketchyCGM` algorithm of Section 3.3.1. Here the sketches are updated in each iteration according to the calculated update direction. Sketching can therefore also allow us to perform computations on data that would otherwise be unmanageable or inconvenient to process.

The benefits and optimisation possibilities of sketching algorithms depend on the given situation, and the experiments presented in this report provide only a basic idea of the performance achievable in a general setting. However, even these simplistic implementations show how sketching algorithms can beat existing methods in certain computational aspects and the examples should serve as inspiration to what sketching can further achieve and provide motivation to develop the algorithms further.

We wish to emphasise that our MATLAB implementations of the sketching matrices and sketching algorithms are in most cases very simple. As described in the following section, the simplicity of the basic algorithms makes sketching accessible as a computational resource and the algorithms can then be extended, modified or improved to work optimally in a given setting. This also means that the gain factors presented in this report are merely indicators of the results that can be achieved.

The following overview indicates how the algorithms from the three stages in

Chapter 3 rely only on few standard MATLAB functions:

Basic building block algorithms consist only of basic matrix operations such as addition and multiplication with Algorithms 2 and 3 also using the built-in QR decomposition function `qr`.

Generic linear algebra algorithms are made up of the basic building block algorithms in combination with the built-in functions `svd`, `lsqr` and the `\` operator.

Real-world algorithms are either small modifications of well-known algorithms or a direct application of a generic linear algebra algorithm such as the randomised SVD of Algorithm 7.

The only exception from a simple implementation is within the SRHT sketching method. As described in Section 2.1.3, the multiplication of the Hadamard matrix can be implemented very efficiently. In order to exploit this, we used an implementation, `ffht`, by Johnson and Püschel presented in [JP00]. A Fast Walsh-Hadamard transform is available in MATLAB's Signal Processing Toolbox as the function `fwht`, however, we found `ffht` to be much faster than `fwht`. For example, for a $2^{15} \times 2^{10}$ matrix, the application of `ffht` was 80 times faster than the corresponding computation using `fwht`. MATLAB's `fwht` is therefore not suitable as an indicative implementation of the subsampled randomised Hadamard transform.

4.2 Accessibility

Sketching is at its most useful when coupled with domain-specific knowledge to either interpret the possibly approximate output or impose assumptions that can be used to derive tighter bounds and better guarantees. Much of the literature therefore deals with the use of sketching in narrow fields of application on the basis of which it can be hard to extrapolate to more general settings. Another direction commonly found is a theoretical computer science perspective where the focus is often on algorithmic properties, error bounds and performance guarantees. For example, the introductory survey [Woo14] highlights advances in algorithms for numerical linear algebra through theoretical results, relevant historic context and some sample pseudocode but contains very few concrete examples and practical results.

The article [Tro+17] has a declared goal of "developing simple, practical algorithms that can serve as reliable modules in other applications" with a focus on portraying the links between theoretical and practical aspects of sketching. However, the article explicitly concerns sketching for low-rank matrix approximation and the presented algorithms are designed specifically for settings where access to the data is very limited, such that only a single pass is possible or the data matrix is revealed through a stream of linear updates. The applicability of sketching in more general settings and for the purpose of reducing computation time is not discussed.

Mahoney's monograph [Mah+11] also has a greater emphasis on the ideas underlying sketching, particularly the connection with statistical leverage scores, and on distinguishing between theoretical and practical performance. However, the connections and interplay between the different applications of sketching can be hard to grasp and the presented algorithms are based on specific choices of sketching matrices rather than general subspace embeddings. Since the article was published in 2011, the sparse projection method has become popular in the sketching community and, as we saw in Section 2.1.4, a sparse embedding matrix does not satisfy the Johnson-Lindenstrauss Lemma directly but can still provide a subspace embedding through the Johnson-Lindenstrauss moment property. From the description of random projections and specific algorithms in [Mah+11], it is difficult to determine whether alternative sketching methods such as as the sparse projection can be used to the same effect as the presented sketching techniques.

As the above examples highlight, the underlying methods and the interdependence between applications are often obscured and overshadowed. In the worst case, this hinders access to the methods for those unfamiliar with the field seeking an understanding of sketching and its applicability to their scientific areas.

One of the main contributions of this report is the identification of three elementary building blocks and the structuring of applications in layers based on these primitives. The basic building blocks of Section 3.1 are the fundamental components in the generic linear algebra applications of Section 3.2, which in turn provide the basis for the more specialised applications in Section 3.3. As soon as one recognises one of the basic building blocks in a given setting, sketching can potentially be used to gain an advantage. This allows for very easy identification of prospective usage across a vast area of scientific and mathematical domains and makes sketching accessible to a large audience.

Each application is accompanied by an algorithm, all of which are easily implemented in most high-level programming languages. The pseudocode is presented with the use of built-in MATLAB functions but these are all widely available linear algebra tools. The generality and simplicity of the algorithms is also aimed at easing the understanding of functionality and how the randomisation is incorporated as modifications or supplements to standard linear algebra routines. The algorithms and applications can be used as springboards and tailored to more specialised usages where domain knowledge can be leveraged to further improve performance and guarantees.

Finally, we note that the mathematical foundations of Chapter 2 introduce the crucial concept of subspace embeddings before presenting four specific methods of constructing sketching matrices. These are by no means the only available sketching methods but represent the most common instances appearing in the sketching literature by covering both sampling and various types of projections. Importantly, the practical applications are not based on any one sketching method but stated for general subspace embeddings. The numerical experiments conducted for each basic application serves as a comparison of the four sketching methods, supporting the theoretical comparison of Section 2.2 and aiding the choice of an appropriate method for a given problem.

4.3 Further applications

We will now highlight two applications of sketching that have received considerable interest within the numerical linear algebra community. The `blendenpik` algorithm is a fast least squares solver while `Newton-Sketch` is a modification of Newton's method used in optimisation problems. Both are based on the desire to reduce time complexity, and the underlying sketching principles can be traced directly to the building blocks of Section 3.1.

The section is rounded off by a discussion of topics for further investigation based on the presented theory, observed results and assessment of applicability provided in this report.

4.3.1 The `blendenpik` algorithm

In [AMT10], Avron, Maymounkov and Toledo introduce the `blendenpik` algorithm, a least squares solver useful for dense, highly overdetermined systems of equations. The authors find the `blendenpik` algorithm to perform extremely well compared to existing state-of-the-art algorithms, with the vanilla implementation consistently beating LAPACK's direct QR factorisation-based solvers. As is apparent from the following short description, the `blendenpik` algorithm is simply an instance of Algorithm 6 from Section 3.2.2.

`blendenpik`: Use Algorithm 2 with the SRHT sketching method (or similar) in order to construct $\mathbf{R_S}$. Estimate the condition number of $\mathbf{R_S}$ and compare with a certain threshold. If acceptable, use the iterative method `lsqr` with $\mathbf{R_S}$ as preconditoner, otherwise use LAPACK.

The success of the `blendenpik` algorithm shows that a simple algorithm for solving overdetermined least squares problems with the `lsqr` method is easily extended to outperform state-of-the-art least squares algorithms with only small modifications. In general, the experiments of [AMT10] showed that `blendenpik` achieves high-precision solutions much faster than LAPACK in all their investigations which includes test matrices of different sizes and properties, and both well- and ill-conditioned matrices. The only test cases where LAPACK was found to be faster, involved tiny, nearly square or sparse matrices and this was not consistently. The article also compared the `blendenpik` algorithm to an unpreconditioned version of `lsqr`, with similar results to those described in the experiments of Section 3.2.2.

[AMT10] concludes that randomised projection methods for solving overdetermined least squares problems like `blendenpik` should be incorporated into future version of LAPACK.

4.3.2 The `Newton-Sketch` algorithm

An algorithm named `Newton-Sketch` was introduced by Mert Pilanci and Martin Wainwright in the paper [PW15]. The method is a modification of the ubiquitous Newton's method used for solving various optimisation problems. `Newton-Sketch` is much like the original Newton's method with the exception of taking approximate Newton steps using an approximation of the Hessian. The classical Newton's method is very widely used and thus when Pilanci and Wainwright claimed that Newton's sketch has substantially lower complexity than Newton's method, sketching was once more the centre of attention.

The classical Newton's method is used in optimisation to find the roots of a twice-differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ by solving the linear equation

$$\left[\nabla^2 f(\mathbf{x}_t)\right] \mathbf{p}_t = -\nabla f(\mathbf{x}_t) \tag{4.1}$$

for the direction $\mathbf{p}_t$ in each iteration and performing the update $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{p}_t$. The cost of an iteration can be split into the following three factors [BBN17]:

- Computing the gradient $\nabla f(\mathbf{x}) \in \mathbb{R}^n$,

- Forming the Hessian $\nabla^2 f(\mathbf{x}) \in \mathbb{R}^{n\times n}$,
- Solving the linear system (4.1).

The idea behind `Newton-Sketch` is to use sketching to approximate $\nabla^2 f(\mathbf{x})$. To do this, it is initially assumed that the Hessian is decomposed as $\mathbf{X}^\top\mathbf{X} = \nabla^2 f(\mathbf{x})$ and that the factor $\mathbf{X}$ is readily available. Note that this factor, which is referred to as a Hessian matrix square root in [PW15], need not be square.

Consider a finite-sum problem where f is of the form $f(\mathbf{x}) = \sum_{i=1}^m g_i\left(\mathbf{A}_{(i)}\mathbf{x}\right)$ for some $\mathbf{A} \in \mathbb{R}^{m\times n}$ and functions $g_i : \mathbb{R}^m \to \mathbb{R}$, $i = 1, \dots, m$. Defining the diagonal matrix $\mathbf{D} \in \mathbb{R}^{m\times m}$ by $\mathbf{D}_{ii} = \sqrt{\nabla^2 g_i\left(\mathbf{A}_{(i)}\mathbf{x}\right)}$, the Hessian of f is given by $\nabla^2 f = \mathbf{A}^\top\mathbf{D}\mathbf{D}\mathbf{A}$ and $\mathbf{X} = \mathbf{D}\mathbf{A} \in \mathbb{R}^{m\times n}$ is a Hessian matrix square root. In this setting, forming the Hessian exactly is a matrix product of complexity $O\left(mn^2\right)$. Utilising the decomposition of the Hessian, we construct a sketching matrix $\mathbf{S}_t \in \mathbb{R}^{k\times m}$ in each iteration and approximate the Hessian by

$$\nabla^2 f(\mathbf{x}_t) \approx \left(\mathbf{S}_t\mathbf{X}_t\right)^\top\left(\mathbf{S}_t\mathbf{X}_t\right), \tag{4.2}$$

corresponding directly to the use of Algorithm 1. Assuming $\left(\mathbf{S}_t\mathbf{X}_t\right)^\top\left(\mathbf{S}_t\mathbf{X}_t\right)$ is invertible, each step in `Newton-Sketch` is given by

$$\mathbf{x}_t = \mathbf{x}_t - \left(\left(\mathbf{S}_t\mathbf{X}_t\right)^\top\left(\mathbf{S}_t\mathbf{X}_t\right)\right)^{-1}\nabla f(\mathbf{x}_t).$$

This method still requires the computation of the gradient, and the linear system in Equation (4.2) is of the same size as the original linear system (4.1). The computational gain of `Newton-Sketch` as presented in [PW15] is therefore entirely in approximating the Hessian. This approximation often comes at the price of extra iterations when compared to the classical Newton's method.

The numerical results of [PW15] show `Newton-Sketch` to convincingly outperform Newton's method when it comes to wall-clock time. However, our own investigation for the case of unconstrained logistic regression showed that the classical Newton method was considerably faster than shown in [PW15], and the cost of the extra iterations in `Newton-Sketch` therefore resulted in little or no gain in wall-clock time. This suggests that the computation of the exact Hessian in [PW15] was implemented in an inefficient manner.

A possible way of speeding up the method is to note that the factorisation of the Hessian allows for the use of iterative solvers which avoid constructing the full approximate Hessian. This is the approach taken in [BBN17], where `Newton-Sketch` is compared to subsampled Newton methods. In their experiments, they use a matrix-free conjugate gradient method to solve the system (4.2) approximately. Since the linear system is already based on an approximate Hessian, this second approximation is not expected to have much impact on the number of iterations required. [BBN17] report the "most optimistic computational results for `Newton-Sketch`" in that they ignore the cost of forming the Hessian square root and sketching this matrix, which they note can be time consuming operations. These extra operations were part of the reason why the added iterations proved too expensive when we considered `Newton-Sketch` in the logistic regression problem.

Alternatives to the classical Newton's method that provide similar convergence properties but at a smaller computational cost, are of great interest in optimisation. `Newton-Sketch` is one such method and is proven to provide advantages in certain settings. However, the need for a Hessian square root impedes the applicability of the method in more general applications and often requires expensive operations.

4.3.3 Topics for further investigation

In Chapter 3, we describe and test nine different applications of sketching with four different sketching methods. As the aim of the report is to convey the underlying ideas and interdependencies between sketching applications, a considerable amount of application-specific theory and testing has been omitted. A natural direction of further investigation is therefore the consideration of each sketching application in finer detail. The following is a review of relevant questions and suggested topics to explore for each of the applications in the first two stages: basic building blocks and generic linear algebra problems. This is followed by some topics that are more general in nature.

Matrix multiplication: The experiments of Section 3.1.1 showed that leverage score sampling produced significantly better approximation results for the `srand` test matrix when compared to the other sketching methods. However, in the same example the method was slower than the exact calculation of the matrix product due to the approximation of leverage scores. It is therefore of interest to analyse other importance sampling methods to see whether they can replicate or even better the approximation quality of matrix multiplication at a lower computational cost, something which is discussed in [DKM06].

Orthonormalising transformation matrix: Using Algorithm 2 to obtain $\mathbf{R_S}$ for an $m \times n$ input matrix requires that $k \geq n$ for $\mathbf{R_S}$ to be square and $k < m$ in order to yield a saving in computation time. Thus, the algorithm is subject to certain input restrictions and an extension to handle other cases, for example when the input matrix is almost square, would be useful.

Figure 3.4 showed that the diagonal entries of $\left(\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}\right)^{\top}\mathbf{A}\mathbf{R}_{\mathbf{S}}^{-1}$ were increasing, an effect for which we have no full explanation. A further investigation might reveal that it is possible to correct for this within the solution method, that it is caused directly by the used sketching methods, or that it is in fact an unavoidable consequence of the applied procedure.

Finally, our approach was based on use of the QR decomposition to find $\mathbf{R}_{\mathbf{S}}$. One could also have used an SVD and constructed $\mathbf{R}_{\mathbf{S}}$ as the product $\mathbf{\Sigma}\mathbf{V}^{\top}$ as suggested in [Dri+12]. This factorisation makes the inverse $\mathbf{R}_{\mathbf{S}}^{-1}$ easy to calculate, which is desirable for many downstream applications, for example the approximation of leverage scores as presented in Algorithm 4.

k-dimensional orthonormal basis: Theorem 14 provides the theoretical basis for the presented method and bounds the approximation error of projecting onto the k-dimensional basis by a term including the spectral norm of the sketching matrix. It would be interesting to investigate the impact of this term in practice as the experimental results shown in Table 3.3 do not suggest big differences between the sketching matrices, apart from the leverage score sampling performing considerably worse for the `cond10` test matrix.

Furthermore, Table 3.3 compares the sketched k-dimensional basis with both the optimal k- and $k/2$-dimensional bases and showed that much greater accuracy is gained by including a few additional columns in the approximation. A more complete analysis of this effect would be of value, particularly since this idea is utilised in Section 3.2.3 where the sketch size is set larger than the target rank as suggested by [Tro+17].

An intriguing direction of research within low-rank approximation based on the approaches of Section 3.1.3, is the preservation of matrix structure such as symmetry or positive-semidefiniteness. Algorithm 3 cannot be used directly in this regard, however, [Tro+17] provides algorithms that handle each of these cases separately, outputting low-rank approximations with the desired property.

Leverage score estimation: The obvious consideration regarding leverage score estimation is lowering the computational complexity. As seen throughout Chapter 3, the leverage score sampling method rarely provided significant computational gains in terms of time. Faster methods of approximating the leverage scores is therefore not only attractive as a stand-alone application but also in the context of improving the leverage score sampling method.

There are also interesting alternatives or extensions of leverage scores that provide similar functionality but improved properties in certain settings. One such development is the ridge leverage scores presented in [CMM17], where a new algorithm for finding a near optimal low-rank approximation of a matrix $\mathbf{A}$ in $O(\mathrm{nnz}(\mathbf{A}))$ time is presented. The ridge leverage scores are adapted to low-rank approximation through regularisation with a parameter based on the best rank k approximation error.

Overdetermined least squares: When using the normal equation for the least squares problem $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|$, one has a choice of whether to sketch both sides of the equation or simply the matrix product $\mathbf{A}^\top\mathbf{A}$ on the left hand side. The significance of this choice is not quite clear and the literature contains examples of both approaches. For example, [Mah+11] considers the fully sketched problem while [PW16] uses the second approach in sketching only the Hessian of a problem in a similar fashion to that of the `Newton-Sketch` algorithm.

Algorithm 6 was found to give decent results for ill-conditioned problems, however, when the problem is well-conditioned, solving the normal equation is often faster as shown in Table 3.5. To construct an algorithm that handles arbitrary input and chooses a solution method accordingly, it would therefore be advantageous to estimate the condition number of the input matrix. There are relatively fast routines for this, for example LAPACK's `dtrcon` as mentioned in [AMT10], but it would be interesting to see whether sketching could be used in this type of application. The performance of the randomised SVD of Algorithm 7 gives hope that this could indeed be possible.

Singular value decomposition: The implemented randomised SVD gave good results in terms of gain in computation time, especially for the `cond10` test matrix as shown in Table 3.7. The accuracy of the output also seemed reasonable, however, the error measures were mostly based on low-rank approximation and other metrics could also be relevant to consider. If the method is used to find a number of singular vectors for example, it might be desirable that these closely approximate the exact singular vectors rather than just provide a good low-rank approximation.

Extending the functionality of Algorithm 7 to handle single-pass scenarios and incorporating a power iteration, as mentioned below the algorithm, would significantly increase the applicability of the method in cases where access to the input matrix is limited and storage rather than computation time is the main concern. [HMT11] explains some of these considerations, however, testing the impact of these and the increase in computational complexity as compared with the basic Algorithm 7 would shed further light on the performance gains.

On the basis of our experiments, it often seems sufficient to choose the sketching parameter k proportional to the lower dimension of the input matrix to obtain a reasonable balance between approximation quality and gained advantages. However, this observation is not substantially tested, and it would therefore be interesting to investigate whether general guidelines for the choice of sketching parameters could be established. Such guidelines would increase the practicality of sketching and help provide empirically-based recommendations that would complement the theoretical bounds.

This approach is taken in the paper [Tro+17] on practical sketching algorithms for low-rank approximation, which includes a section on "Theoretical guidance on sketching sizes". However, the presented guidance is limited to the use of the Gaussian sketching matrix in low-rank approximation problems and thus not very general. As an alternative suggestion for creating a general guideline for the sketch size parameter, it might be useful to consider the basic building blocks of this report since these provide the foundation for most other sketching algorithms. If it is possible to establish general rules for sketching method selection and size specification in these three basic problems, we could hope to extrapolate this to other sketching applications and thereby create a more general set of recommendations.

CHAPTER 5

Conclusion

Sketching is a very general concept covering the reduction in size of a large range of problems from basic operations such as matrix multiplication, over standard optimisation methods like least squares, to low-rank approximation and matrix completion. In much of the literature, the simplicity of the underlying idea is often overshadowed by theoretical guarantees that do not seem compatible with practical uses, overly specialised methods where sketching is hidden implicitly in algorithms, and issues concerning implementation of particular sketching methods in certain environments. The aim of this project has been to provide a practical introduction to sketching and establish a framework for analysing the use of sketching in algorithms, paving the way for incorporating sketching in the solution to a host of different problems.

Our main contribution is the identification of three basic building blocks representing the most elementary uses of sketching, the value of which is two-fold. Firstly, they can be used to pinpoint the exact use of sketching in randomised methods, easing analysis of the contribution and merits of sketching in a given application, as illustrated by the generic linear algebra problems of Section 3.2. Secondly, they can be employed as components in modifications of existing algorithms or as a way of constructing novel algorithms in a modular fashion, which the matrix completion in recommender systems and the generalised Tikhonov regularised optimisation problem of Section 3.3 both demonstrate.

In extension of the above, an objective has been to clarify the relationship between theoretical and practical results. Chapter 2 presented the fundamental theory of subspace embeddings and four different sketching methods, while Chapter 3 studied the use of sketching in nine different applications. This allowed for the comparison of theoretical properties and guarantees with the results obtainable in various simple problem settings. A key observation is that in many cases significant advantages can be achieved using sketching matrices with a number of rows far below that required to theoretically ensure accurate approximations. It is notable that these computational gains are from simple algorithms that are easily implemented, since it is naturally hard for novel approaches to compete with well-established computational methods.

The results obtained promise well for the future development of faster sketching methods as the theoretical complexities are more closely reflected in implementations. However, more efficient sketching methods is also a necessity for some applications, especially if small approximation errors are required, as the results of this report also indicate. For example, the Gaussian projection and leverage score sampling methods rarely provided time reductions of note, and for very small errors, the number of rows required in the sketching matrices would render most of the methods ineffective.

This also highlights an important facet of sketching in practice that users must consider: how much accuracy can be sacrificed? The results clearly exemplify the trade-off between computational gains and approximation precision that sketching often implies. Furthermore, bypassing the theoretical number of rows required in practice means that a form of supervision of the obtained results is required. In many of the presented applications, an error tolerance cannot be guaranteed or built into an algorithm without at least some knowledge of the true solution, unless an excessive number of rows is used in the sketching process.

The above considerations further substantiate our claim that the best results are obtained when sketching is used in algorithms purpose-built for specific applications and incorporating domain knowledge. This enables a tighter analysis through realistic assumptions and algorithmic choices, and can facilitate the inclusion of error supervision or other control measures. The building block structure presented in this report accommodate the construction of specialised algorithms by providing simple modules that can be leveraged directly. The introduced framework makes sketching more accessible as an algorithmic tool and hopefully serves to further motivate the use of randomisation as a computational resource.

APPENDIX A

Proofs from Chapter 2

A.1 Proof of Theorem 3

The proof follows the idea of [Woo14, Theorem 2.11], but uses [IW14, Theorem 3.8] to reduce the minimum number of rows k needed.

In the proof of Theorem 3 we will need the following Chernoff bound for random matrices.

Lemma 16 ([Mag10, Theorem 13]). *Let $\mathbf{X}_1, \dots, \mathbf{X}_k$ be independent copies of a symmetric random $n \times n$ matrix $\mathbf{X}$ with $\mathbb{E}[\mathbf{X}] = \mathbf{0}$, $\|\mathbf{X}\|_2 \le \gamma$ and $\left\|\mathbb{E}\left[\mathbf{X}^\top \mathbf{X}\right]\right\|_2 \le \sigma^2$. Letting $\mathbf{W} = \frac{1}{k}\sum_{j=1}^{k} \mathbf{X}_j$, then*

$$\Pr\left[\|\mathbf{W}\|_2 > \varepsilon\right] \le 2ne^{-\frac{3}{2}\frac{k\varepsilon^2}{3\sigma^2+\gamma\varepsilon}} \qquad \textit{for any } \varepsilon > 0.$$

We are now in a position to prove that a distribution of sampling matrices based on approximate leverage score probabilities gives rise to a subspace embedding.

Proof of Theorem 3. For any $0 < \delta < 1$, let $\mathbf{\Pi}^q_{\text{LEV}}$ be a distribution on $k \times m$ sampling matrices based on approximate leverage score probabilities $\mathbf{q} = (q_1, q_2, \dots, q_m)$ of some input matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $\operatorname{rank}(\mathbf{A}) = n$ and orthonormal basis $\mathbf{U}$. Let $\mathbf{S} \sim \mathbf{\Pi}^q_{\text{LEV}}$ and define the series of symmetric matrices $\mathbf{X}_i \in \mathbb{R}^{n\times n}$ for $i = 1, \dots, k$, given by

$$\mathbf{X}_i = \mathbf{I} - \frac{(\mathbf{SU})^\top_{(i)}(\mathbf{SU})_{(i)}}{(\mathbf{Sq})_i}.$$

These are in fact independent copies of a random matrix variable that we call $\mathbf{X}$. We note that both terms in the expression for $\mathbf{X}_i$ are symmetric and positive semi-definite. For general such matrices, $\mathbf{M}_1$ and $\mathbf{M}_2$, the following inequality holds

$$\begin{aligned}\|\mathbf{M}_1 - \mathbf{M}_2\|_2 &= \max_{\|x\|=1} \mathbf{x}^\top(\mathbf{M}_1 - \mathbf{M}_2)\,\mathbf{x} = \max_{\|x\|=1}\left(\mathbf{x}^\top\mathbf{M}_1\mathbf{x} - \mathbf{x}^\top\mathbf{M}_2\mathbf{x}\right) \\ &\le \max\left\{\|\mathbf{M}_1\|_2, \|\mathbf{M}_2\|_2\right\}. \end{aligned} \tag{A.1}$$

In order to use the matrix Chernoff bound presented above we need to investigate the properties of this random variable. Taking the expectation of the i-th sampled row we see that $\mathbb{E}\left[(\mathbf{SU})_{(i)}\right] = \sum_{j=1}^{m} q_j \mathbf{U}_{(j)}$. This is used to show that $\mathbf{X}$ has mean zero as follows

$$\mathbb{E}\left[\mathbf{X}_i\right] = \mathbf{I} - \mathbb{E}\left[\frac{(\mathbf{SU})_{(i)}^\top (\mathbf{SU})_{(i)}}{(\mathbf{Sq})_i}\right] = \mathbf{I} - \sum_{j=1}^{m} q_j \left(\frac{\mathbf{U}_{(j)}^\top \mathbf{U}_{(j)}}{q_j}\right) = \mathbf{I} - \mathbf{U}^\top \mathbf{U} = \mathbf{0}.$$

Next, it is shown that $\|\mathbf{X}\|_2$ is bounded by using the inequality (A.1) and the relation between the q_i's and the leverage scores, yielding

$$\begin{aligned}\|\mathbf{X}_i\|_2 &\leq \max\left\{\|\mathbf{I}\|_2, \left\|\frac{(\mathbf{SU})_{(i)}^\top (\mathbf{SU})_{(i)}}{(\mathbf{Sq})_i}\right\|_2\right\} \leq \max\left\{1, \max_j \frac{\left\|\mathbf{U}_{(j)}^\top \mathbf{U}_{(j)}\right\|_2}{q_j}\right\} \\ &\leq \max\left\{1, \max_j \frac{\ell_j}{q_j}\right\} = \frac{n}{\beta}.\end{aligned}$$

Finally, the expected variance is also bounded which is shown using the same tricks as earlier, as

$$\begin{aligned}\mathbb{E}\left[\mathbf{X}_i^\top \mathbf{X}_i\right] &= \mathbb{E}\left[\mathbf{I}\right] - 2\,\mathbb{E}\left[\frac{(\mathbf{SU})_{(i)}^\top (\mathbf{SU})_{(i)}}{(\mathbf{Sq})_i}\right] + \mathbb{E}\left[\frac{(\mathbf{SU})_{(i)}^\top (\mathbf{SU})_{(i)} (\mathbf{SU})_{(i)}^\top (\mathbf{SU})_{(i)}}{(\mathbf{Sq})_i^2}\right] \\ &= \mathbf{I} - 2\sum_{j=1}^{m} q_j \left(\frac{\mathbf{U}_{(j)}^\top \mathbf{U}_{(j)}}{q_j}\right) + \sum_{j=1}^{m} q_j \left(\frac{\mathbf{U}_{(j)}^\top \mathbf{U}_{(j)} \mathbf{U}_{(j)}^\top \mathbf{U}_{(j)}}{q_j^2}\right) \\ &= \mathbf{I} - 2\mathbf{U}^\top \mathbf{U} + \sum_{j=1}^{m} \frac{\left\|\mathbf{U}_{(j)}\right\|_2^2}{q_j} \mathbf{U}_{(j)}^\top \mathbf{U}_{(j)} \\ &= \sum_{j=1}^{m} \frac{\ell_j}{q_j} \mathbf{U}_{(j)}^\top \mathbf{U}_{(j)} - \mathbf{I}.\end{aligned}$$

Using inequality (A.1) we then have

$$\left\|\mathbb{E}\left[\mathbf{X}^\top \mathbf{X}\right]\right\|_2 \leq \max\left\{\left\|\sum_{j=1}^{m} \frac{\ell_j}{q_j} \mathbf{U}_{(j)}^\top \mathbf{U}_{(j)}\right\|_2, \|\mathbf{I}\|_2\right\} \leq \max\left\{\frac{n}{\beta}\left\|\mathbf{U}^\top \mathbf{U}\right\|_2, 1\right\} \leq \frac{n}{\beta}.$$

Define $\mathbf{W} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{X}_i$ and observe that $\mathbf{W} = \mathbf{I} - \mathbf{U}^\top \mathbf{S}^\top \mathbf{S} \mathbf{U}$. Hence, we are in a position where the matrix Chernoff bound can be applied with $\sigma^2 = n/\beta$ and $\gamma = n/\beta$ to give

$$\Pr\left[\left\|\mathbf{I} - \mathbf{U}_{\mathbf{A}}^\top \mathbf{S}^\top \mathbf{S} \mathbf{U}_{\mathbf{A}}\right\|_2 > \varepsilon\right] \leq 2ne^{-\frac{3}{2}\frac{k\varepsilon^2}{3\sigma^2 + \gamma\varepsilon}} \leq 2ne^{-\frac{3}{2}\frac{k\varepsilon^2}{3n/\beta + n\varepsilon/\beta}} = 2ne^{-\frac{3}{2}\frac{k\beta\varepsilon^2}{n(3+\varepsilon)}}$$

This implies that $\mathbf{S}$ is a subspace embedding for $\mathbf{A}$ except, with probability at most $2ne^{-\frac{3}{2}\frac{k\beta\varepsilon^2}{n(3+\varepsilon)}}$. To match the theorem, we would like to be able to choose this probability as a value δ that is independent of β, ε and n. This imposes a restriction on the minimum number of rows k needed. We therefore seek k such that

$$\delta \geq 2ne^{-\frac{3}{2}\frac{k\beta\varepsilon^2}{n(3+\varepsilon)}}.$$

Since $0 < \varepsilon < 1$, this is satisfied when

$$k \geq \left(2 + \frac{2}{3}\varepsilon\right) n\beta^{-1}\varepsilon^{-2} \ln\left(\frac{2n}{\delta}\right).$$

In the theorem the constant $2+\frac{2}{3}\varepsilon$ is replaced with the larger value $\frac{8}{3}$ for simplicity. □

A.2 Proof of Theorem 5

This proof follows the approach in [Woo14, Theorem 2.1 and Lemma 2.2] but provides a tighter analysis by using [Ver10, Lemma 5.2].

In the proof of the theorem, we will use an ε-net as a particular choice of finite subset on which to exploit the Johnson–Lindenstrauss property. For a subset X of a metric space M, a set $\mathcal{N} \subset M$ is an ε-net for X, if for every point $x \in X$, there is a point of $\mathcal{N}$ at a distance from x less than ε. We firstly prove the existence of small finite nets on the unit sphere $S^{n-1} = \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\|_2 = 1\}$.

Lemma 17 ([Ver10, Lemma 5.2])**.** *For any* $0 < \gamma < 1$ *and* $n \in \mathbb{N}$*, there exists a* γ*-net* $\mathcal{N}$ *on* S^{n-1} *for which* $|\mathcal{N}| \leq \left(1 + \frac{2}{\gamma}\right)^n$.

Proof. Take $\mathcal{N}$ as the maximal γ-separated subset of S^{n-1}, i.e. such that $\|\mathbf{x}-\mathbf{y}\|_2 > \gamma$ for all $\mathbf{x}, \mathbf{y} \in \mathcal{N}$, $\mathbf{x} \neq \mathbf{y}$. Then $\mathcal{N}$ is in fact a γ-net for S^{n-1}, since otherwise there would exist $\mathbf{z} \in S^{n-1}$ such that $\|\mathbf{z} - \mathbf{x}\|_2 > \gamma$ for all $\mathbf{x} \in \mathcal{N}$. But then $\mathcal{N} \cup \{\mathbf{z}\}$ is a γ-separated subset of S^{n-1} for any such $\mathbf{z}$, contradicting the maximality of $\mathcal{N}$.

By the separation property, the balls of radii $\frac{\gamma}{2}$ centred at the points of $\mathcal{N}$ are disjoint. Furthermore, these are all contained in the ball of radius $1 + \frac{\gamma}{2}$ centred at the origin. Denoting the volume of the ball of radius r by $\mathrm{Vol}\,(B_r)$ and comparing volumes, we have $|\mathcal{N}| \cdot \mathrm{Vol}\left(B_{\frac{\gamma}{2}}\right) \leq \mathrm{Vol}\left(B_{1+\frac{\gamma}{2}}\right)$ and hence

$$|\mathcal{N}| \leq \frac{\mathrm{Vol}\left(B_{1+\frac{\gamma}{2}}\right)}{\mathrm{Vol}\left(B_{\frac{\gamma}{2}}\right)} = \frac{\left(1+\frac{\gamma}{2}\right)^n}{\left(\frac{\gamma}{2}\right)^n} = \left(1 + \frac{2}{\gamma}\right)^n,$$

giving the desired bound on the number of elements in $\mathcal{N}$. $\square$

The following corollary gives a useful characterisation of the Johnson–Lindenstrauss property in terms of inner products.

Corollary 18. *Let $k, m \in \mathbb{N}$, $\mathbf{S} \in \mathbb{R}^{k\times m}$, $\mathbf{u}, \mathbf{v} \in \mathbb{R}^m$ and $\varepsilon \in \mathbb{R}$. If $\mathbf{S}$ satisfies the Johnson–Lindenstrauss property, i.e.*

$$(1-\varepsilon)\|\mathbf{u}-\mathbf{v}\|_2 \leq \|\mathbf{S}(\mathbf{u}-\mathbf{v})\|_2 \leq (1+\varepsilon)\|\mathbf{u}-\mathbf{v}\|_2,$$

then $\mathbf{S}$ also satisfies

$$|\langle \mathbf{Su}, \mathbf{Sv}\rangle - \langle \mathbf{u}, \mathbf{v}\rangle| \leq \varepsilon \|\mathbf{u}\|_2 \|\mathbf{v}\|_2.$$

Proof. We first prove the result assuming $\mathbf{u}$ and $\mathbf{v}$ are unit vectors, by considering the action of $\mathbf{S}$ on $\mathbf{u}+\mathbf{v}$ and $\mathbf{u}-\mathbf{v}$. By the parallelogram law and property of $\mathbf{S}$

$$\begin{aligned} 4\langle \mathbf{Su}, \mathbf{Sv}\rangle &= \|\mathbf{Su}+\mathbf{Sv}\|_2^2 - \|\mathbf{Su}-\mathbf{Sv}\|_2^2 \\ &\geq (1-\varepsilon)\|\mathbf{u}+\mathbf{v}\|_2^2 - (1+\varepsilon)\|\mathbf{u}-\mathbf{v}\|_2^2 \\ &= \left(\|\mathbf{u}+\mathbf{v}\|_2^2 - \|\mathbf{u}-\mathbf{v}\|_2^2\right) - \varepsilon\left(\|\mathbf{u}+\mathbf{v}\|_2^2 + \|\mathbf{u}-\mathbf{v}\|_2^2\right) \\ &= 4\langle \mathbf{u}, \mathbf{v}\rangle - 2\varepsilon\left(\|\mathbf{u}\|_2^2 + \|\mathbf{v}\|_2^2\right) \\ &= 4\langle \mathbf{u}, \mathbf{v}\rangle - 4\varepsilon. \end{aligned}$$

The opposite inequality is similar, thus yielding $|\langle \mathbf{Sv}, \mathbf{Sv}\rangle - \langle \mathbf{u}, \mathbf{v}\rangle| \leq \varepsilon$ for all unit vectors $\mathbf{u}$ and $\mathbf{v}$. The extension to vectors of arbitrary length follows trivially by linearity of $\mathbf{S}$. $\square$

Proof of Theorem 5. Let $\mathbf{\Pi}$ be a $\left(\frac{\varepsilon}{4}, \delta, 5^n\right)$ Johnson–Lindenstrauss transform on $k\times m$ matrices, draw $\mathbf{S}$ from $\mathbf{\Pi}$ and let $\mathbf{A} \in \mathbb{R}^{m\times n}$ with normalised column space

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{y} \in \mathbb{R}^m \,|\, \mathbf{y} = \mathbf{Ax} \text{ for some } \mathbf{x} \in \mathbb{R}^n \text{ and } \|\mathbf{y}\|_2 = 1\}.$$

We will prove the theorem by taking a suitably sized finite net of $\mathcal{R}(\mathbf{A})$, and showing that since $\mathbf{S}$ satisfies the Johnson–Lindenstrauss property for elements in the chosen net, with probability $1-\delta$, it actually satisfies the Johnson–Lindenstrauss property for the entire subspace $\mathcal{R}(\mathbf{A})$, with probability $1-\delta$.

Initially, we note that for $r = \operatorname{rank}(\mathbf{A}) \leq n$, and by considering an orthonormal matrix $\mathbf{U}$ with the same column space as $\mathbf{A}$, we can express $\mathcal{R}(\mathbf{A})$ as

$$\mathcal{R}(\mathbf{A}) = \{\mathbf{y} \in \mathbb{R}^m \,|\, \mathbf{y} = \mathbf{Ux} \text{ for some } \mathbf{x} \in \mathbb{R}^r \text{ with } \|\mathbf{x}\|_2 = 1\}.$$

Using Lemma 17, we now choose $\mathcal{N}'$ to be a $\frac{1}{2}$-net of size $|\mathcal{N}'| \leq 5^n$ for the sphere S^{r-1}, and define

$$\mathcal{N} = \{\mathbf{y} \in \mathbb{R}^m \mid \mathbf{y} = \mathbf{U}\mathbf{x} \text{ for some } \mathbf{x} \in \mathcal{N}'\} .$$

We now claim that $\mathcal{N}$ is a $\frac{1}{2}$-net for $\mathcal{R}(\mathbf{A})$. To see this, assume the contrary and take $\mathbf{U}\mathbf{x} \in \mathcal{R}(\mathbf{A})$ such that $\|\mathbf{y} - \mathbf{U}\mathbf{x}\|_2 > \frac{1}{2}$ for all $\mathbf{y} \in \mathcal{N}$ or equivalently $\|\mathbf{U}\mathbf{z} - \mathbf{U}\mathbf{x}\|_2 > \frac{1}{2}$ for all $\mathbf{z} \in \mathcal{N}'$. By orthogonality of $\mathbf{U}$, this implies $\|\mathbf{z} - \mathbf{x}\|_2 > \frac{1}{2}$ for all $\mathbf{z} \in \mathcal{N}'$, which contradicts $\mathcal{N}'$ being a $\frac{1}{2}$-net on S^{r-1}.

As $\mathbf{\Pi}$ is an $\left(\frac{\varepsilon}{4}, \delta, 5^n\right)$ Johnson–Lindenstrauss transform, then in particular the Johnson–Lindenstrauss property of $\mathbf{S}$ holds for all elements in $\mathcal{N}$, with probability $1 - \delta$. The key to expanding the Johnson–Lindenstrauss property to the entire subspace, is that any $\mathbf{y} \in \mathcal{R}(\mathbf{A})$ can be written as

$$\mathbf{y} = \mathbf{y}_0 + \alpha_1 \mathbf{y}_1 + \alpha_2 \mathbf{y}_2 + \cdots ,$$

where for all $i \in \mathbb{N}$ we have $\mathbf{y}_i \in \mathcal{N}$, and α_i is defined recursively as

$$\alpha_i = \left\| \mathbf{y} - \sum_{j=0}^{i-1} \alpha_j \mathbf{y}_j \right\|_2 .$$

We will show by induction that $\alpha_i \leq \frac{1}{2^i}$ and hence that the above linear combination indeed converges to $\mathbf{y}$. As induction basis we simply note that $\alpha_0 = \|\mathbf{y}\|_2 = 1$. Now assume $\alpha_t \leq \frac{1}{2^t}$ for some $t \in \mathbb{N}$. Then

$$\alpha_{t+1} = \left\| \mathbf{y} - \sum_{j=0}^{t} \alpha_j \mathbf{y}_j \right\|_2 = \alpha_t \left\| \frac{1}{\alpha_t} \left(\mathbf{y} - \sum_{j=0}^{t-1} \alpha_j \mathbf{y}_j \right) - \mathbf{y}_t \right\|_2 \leq \alpha_t \cdot \frac{1}{2} \leq \frac{1}{2^{t+1}} .$$

We can now examine the distortion of $\mathbf{S}$ on an arbitrary $\mathbf{y} \in \mathcal{R}(\mathbf{A})$:

$$\|\mathbf{S}\mathbf{y}\|_2^2 = \left\| \mathbf{S} \left(\sum_{i=0}^{\infty} \alpha_i \mathbf{y}_i \right) \right\|_2^2 = \left\langle \sum_{i=0}^{\infty} \alpha_i \mathbf{S}\mathbf{y}_i, \sum_{j=0}^{\infty} \alpha_j \mathbf{S}\mathbf{y}_j \right\rangle = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \alpha_i \alpha_j \left\langle \mathbf{S}\mathbf{y}_i, \mathbf{S}\mathbf{y}_j \right\rangle .$$

Using the Johnson–Lindenstrauss property of $\mathbf{S}$ on elements of $\mathcal{N}$ as given by Corollary 18, we have that, with probability $1 - \delta$, this is bounded from above by

$$\begin{aligned} \|\mathbf{S}\mathbf{y}\|_2^2 &\leq \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \alpha_i \alpha_j \left(\langle \mathbf{y}_i, \mathbf{y}_j \rangle + \frac{\varepsilon}{4} \right) \leq \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \langle \mathbf{y}_i, \mathbf{y}_j \rangle + \frac{\varepsilon}{4} \sum_{i=0}^{\infty} \alpha_i \sum_{j=0}^{\infty} \alpha_j \\ &\leq \|\mathbf{y}\|_2^2 + \varepsilon = 1 + \varepsilon. \end{aligned}$$

The bound from below is shown in the same way, and hence since $\mathbf{A}$ was chosen arbitrarily, we have shown that $\mathbf{\Pi}$ is an (ε, δ, n) oblivious subspace embedding. $\square$

A.3 Proof of Lemma 10

This proof is based on the proof in [KN14, Theorem 21].

Proof of Lemma 10. Let $\mathbf{\Pi}$ be a distribution as in Lemma 10 and let $\mathbf{A}$ and $\mathbf{B}$ be matrices with m rows. We aim to prove

$$\Pr_{\mathbf{S}\sim\mathbf{\Pi}}\left[\left\|(\mathbf{SA})^\top(\mathbf{SB}) - \mathbf{A}^\top\mathbf{B}\right\|_F > 3\varepsilon \|\mathbf{A}\|_F \|\mathbf{B}\|_F\right] \leq \delta,$$

which we will do in a three step procedure. Firstly, we define a norm on random scalars X by

$$\|X\|_p = \left(\mathbb{E}\left[|X|^p\right]\right)^{1/p}.$$

To see that this is indeed a norm, note that it is positive definite by the non-degeneracy of the expectation $\mathbb{E}$, it has uniform scaling by linearity of $\mathbb{E}$, and Minkowski's inequality ensures that it satisfies the triangle inequality.

We will show that for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$ and $\mathbf{S} \sim \mathbf{\Pi}$, we have

$$\|\langle \mathbf{Sx}, \mathbf{Sy}\rangle - \langle \mathbf{x}, \mathbf{y}\rangle\|_l \leq 3\varepsilon\delta^{1/l} \|\mathbf{x}\|_2 \|\mathbf{y}\|_2. \tag{A.2}$$

To show this, we initially note that

$$2\langle \mathbf{x}, \mathbf{y}\rangle = \|\mathbf{x}\|_2^2 + \|\mathbf{y}\|_2^2 - \|\mathbf{x}-\mathbf{y}\|_2^2.$$

Using this and assuming $\|\mathbf{x}\|_2 = \|\mathbf{y}\|_2$, we get the following via the triangle inequality

$$\begin{aligned} 2\|\langle \mathbf{Sx}, \mathbf{Sy}\rangle - \langle \mathbf{x}, \mathbf{y}\rangle\|_l &= \left\|\|\mathbf{Sx}\|_2^2 + \|\mathbf{Sy}\|_2^2 - \|\mathbf{Sx}-\mathbf{Sy}\|_2^2 - \|\mathbf{x}\|_2^2 - \|\mathbf{y}\|_2^2 - \|\mathbf{x}-\mathbf{y}\|_2^2\right\|_l \\ &\leq \left\|\|\mathbf{Sx}\|_2^2 - 1\right\|_l + \left\|\|\mathbf{Sy}\|_2^2 - 1\right\|_l + \left\|\|\mathbf{S}(\mathbf{x}-\mathbf{y})\|_2^2 - \|\mathbf{x}-\mathbf{y}\|_2^2\right\|_l. \end{aligned}$$

By the (ε, δ, l) Johnson–Lindenstrauss moment property of $\mathbf{\Pi}$

$$\left\|\|\mathbf{Sx}\|_2^2 - 1\right\|_l^l = \mathop{\mathbb{E}}_{\mathbf{S}\sim\mathbf{\Pi}}\left[\left|\|\mathbf{Sx}\|_2^2 - 1\right|^l\right] \leq \varepsilon^l\delta,$$

we get

$$\begin{aligned} 2\|\langle \mathbf{Sx}, \mathbf{Sy}\rangle - \langle \mathbf{x}, \mathbf{y}\rangle\|_l &\leq \left(\varepsilon^l\delta\right)^{1/l} + \left(\varepsilon^l\delta\right)^{1/l} + \left(\varepsilon^l\delta\|\mathbf{x}-\mathbf{y}\|_2^{2l}\right)^{1/l} \\ &\leq \varepsilon\delta^{1/l} + \varepsilon\delta^{1/l} + \varepsilon\delta^{1/l}\left(\|\mathbf{x}\|_2 + \|\mathbf{y}\|_2\right)^2 \\ &\leq 6\varepsilon\delta^{1/l}. \end{aligned}$$

By linearity, the above inequality can be extended to arbitrary $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, whereby the desired result is achieved.

We now use the result in (A.2) along with the triangle inequality and Minkowski's inequality to calculate

$$\begin{aligned}
\left\| \left\| (\mathbf{SA})^\top (\mathbf{SB}) - \mathbf{A}^\top \mathbf{B} \right\|_F^2 \right\|_{l/2} &= \left\| \sum_{i,j} \left(\left\langle \mathbf{SA}^{(i)}, \mathbf{SB}^{(j)} \right\rangle - \left\langle \mathbf{A}^{(i)}, \mathbf{B}^{(j)} \right\rangle \right)^2 \right\|_{l/2} \\
&\le \sum_{i,j} \left\| \left(\left\langle \mathbf{SA}^{(i)}, \mathbf{SB}^{(j)} \right\rangle - \left\langle \mathbf{A}^{(i)}, \mathbf{B}^{(j)} \right\rangle \right)^2 \right\|_{l/2} \\
&\le \sum_{i,j} \left\| \left\langle \mathbf{SA}^{(i)}, \mathbf{SB}^{(j)} \right\rangle - \left\langle \mathbf{A}^{(i)}, \mathbf{B}^{(j)} \right\rangle \right\|_l^2 \\
&\le \sum_{i,j} \left(3\varepsilon\delta^{1/l} \left\| \mathbf{A}^{(i)} \right\|_2 \left\| \mathbf{B}^{(j)} \right\|_2 \right)^2 \\
&= \left(3\varepsilon\delta^{1/l} \right)^2 \|\mathbf{A}\|_F^2 \|\mathbf{B}\|_F^2 .
\end{aligned}$$

The theorem follows by Markov's inequality, as

$$\begin{aligned}
\Pr_{\mathbf{S}\sim\mathbf{\Pi}} \left[\frac{\left\| (\mathbf{SA})^\top (\mathbf{SB}) - \mathbf{A}^\top \mathbf{B} \right\|_F}{\|\mathbf{A}\|_F \|\mathbf{B}\|_F} > 3\varepsilon \right] &\le \frac{\mathbb{E}_{\mathbf{S}\sim\mathbf{\Pi}} \left[\left\| (\mathbf{SA})^\top (\mathbf{SB}) - \mathbf{A}^\top \mathbf{B} \right\|_F \right]}{3\varepsilon \|\mathbf{A}\|_F \|\mathbf{B}\|_F} \\
&\le \frac{\mathbb{E}_{\mathbf{S}\sim\mathbf{\Pi}} \left[\left\| (\mathbf{SA})^\top (\mathbf{SB}) - \mathbf{A}^\top \mathbf{B} \right\|_F^l \right]}{(3\varepsilon)^l \|\mathbf{A}\|_F^l \|\mathbf{B}\|_F^l} \\
&= \frac{\left\| \left\| (\mathbf{SA})^\top (\mathbf{SB}) - \mathbf{A}^\top \mathbf{B} \right\|_F^2 \right\|_{\frac{l}{2}}^{\frac{l}{2}}}{(3\varepsilon)^l \|\mathbf{A}\|_F^l \|\mathbf{B}\|_F^l} \\
&\le \frac{\left(\left(3\varepsilon\delta^{1/l} \right)^2 \|\mathbf{A}\|_F^2 \|\mathbf{B}\|_F^2 \right)^{\frac{l}{2}}}{(3\varepsilon)^l \|\mathbf{A}\|_F^l \|\mathbf{B}\|_F^l} \\
&= \delta.
\end{aligned}$$

□

Bibliography

[AC06] Nir Ailon and Bernard Chazelle. “Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform”. In: *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing.* ACM. 2006, pages 557–563.

[Ach03] Dimitris Achlioptas. “Database-friendly random projections: Johnson–Lindenstrauss with binary coins”. In: *Journal of Computer and System Sciences* 66.4 (2003), pages 671–687.

[Alo03] Noga Alon. “Problems and results in extremal combinatorics—I”. In: *Discrete Mathematics* 273.1-3 (2003), pages 31–53.

[AMT10] Haim Avron, Petar Maymounkov, and Sivan Toledo. “Blendenpik: Supercharging LAPACK’s least-squares solver”. In: *SIAM Journal on Scientific Computing* 32.3 (2010), pages 1217–1236.

[BBN17] Albert S. Berahas, Raghu Bollapragada, and Jorge Nocedal. “An Investigation of Newton-Sketch and Subsampled Newton Methods”. In: *arXiv preprint arXiv:1705.06211* (2017).

[BG13] Christos Boutsidis and Alex Gittens. “Improved matrix algorithms via the Subsampled randomized Hadamard transform”. In: *SIAM Journal on Matrix Analysis and Applications* 34.3 (2013), pages 1301–1340.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[Bjo96] Åke Bjorck. *Numerical methods for least squares problems.* Society for Industrial and Applied Mathematics, 1996.

[BMD09] Christos Boutsidis, Michael W. Mahoney, and Petros Drineas. “An improved approximation algorithm for the column subset selection problem”. In: *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. Siam. 2009, pages 968–977.

[BW17] Christos Boutsidis and David P. Woodruff. “Optimal CUR matrix decompositions”. In: *SIAM Journal on Computing* 46.2 (2017), pages 543–589.

[CCF02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. “Finding frequent items in data streams”. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2002, pages 693–703.

[CMM17] Michael B. Cohen, Cameron Musco, and Christopher Musco. “Input sparsity time low-rank approximation via ridge leverage score sampling”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pages 1758–1777.

[CNW15] Michael B. Cohen, Jelani Nelson, and David P. Woodruff. “Optimal approximate matrix product in terms of stable rank”. In: *arXiv preprint arXiv:1507.02268* (2015).

[CW13] Kenneth L. Clarkson and David P. Woodruff. “Low rank approximation and regression in input sparsity time”. In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM. 2013, pp. 81–90.

[DG03] Sanjoy Dasgupta and Anupam Gupta. “An elementary proof of a theorem of Johnson and Lindenstrauss”. In: *Random Structures & Algorithms* 22.1 (2003), pages 60–65.

[DK03] Petros Drineas and Ravi Kannan. “Pass efficient algorithms for approximating large matrices”. In: *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. 2003, pages 223–232.

[DK17] Dua Dheeru and Efi Karra Taniskidou. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.

[DKM06] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. “Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication”. In: *SIAM Journal on Computing* 36.1 (2006), pages 132–157.

[DMM08] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. “Relative-error CUR matrix decompositions”. In: *SIAM Journal on Matrix Analysis and Applications* 30.2 (2008), pages 844–881.

[Dri+11] Petros Drineas et al. “Faster least squares approximation”. In: *Numerische mathematik* 117.2 (2011), pages 219–249.

[Dri+12] Petros Drineas et al. “Fast approximation of matrix coherence and statistical leverage”. In: *Journal of Machine Learning Research* 13.Dec (2012), pages 3475–3506.

[GV12] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. JHU Press, 2012.

[H+78] Samad Hedayat, Walter Dennis Wallis, et al. “Hadamard matrices and their applications”. In: *The Annals of Statistics* 6.6 (1978), pages 1184–1238.

[Han98] Per Christian Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. Siam, 1998.

[HK16] F. Maxwell Harper and Joseph A. Konstan. “The movielens datasets: History and context”. In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5.4 (2016), page 19.

[HMT11] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions”. In: *SIAM review* 53.2 (2011), pages 217–288.

[HW78] David C. Hoaglin and Roy E. Welsch. “The hat matrix in regression and ANOVA”. In: *The American Statistician* 32.1 (1978), pages 17–22.

[IN07] Piotr Indyk and Assaf Naor. “Nearest neighbor preserving embeddings”. In: *ACM Transactions on Algorithms (TALG)* 3.3 (2007), page 31.

[IW14] Ilse C. Ipsen and Thomas Wentworth. “The effect of coherence on sampling from matrices with orthonormal columns, and preconditioned least squares problems”. In: *SIAM Journal on Matrix Analysis and Applications* 35.4 (2014), pages 1490–1520.

[Jag13] Martin Jaggi. “Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization.” In: *ICML (1)*. 2013, pages 427–435.

[JL84] William B. Johnson and Joram Lindenstrauss. “Extensions of Lipschitz mappings into a Hilbert space”. In: *Contemporary mathematics* 26.189–206 (1984), page 1.

[JP00] Jeremy Johnson and Markus Püschel. “In search of the optimal Walsh-Hadamard transform”. In: *Acoustics, Speech, and Signal Processing, 2000. ICASSP’00. Proceedings. 2000 IEEE International Conference on.* Volume 6. IEEE. 2000, pages 3347–3350.

[KN14] Daniel M. Kane and Jelani Nelson. “Sparser Johnson–Lindenstrauss transforms”. In: *Journal of the ACM (JACM)* 61.1 (2014), page 4.

[Kre78] Erwin Kreyszig. *Introductory functional analysis with applications.* Volume 1. Wiley New York, 1978.

[Lar98] Rasmus Munk Larsen. “Lanczos bidiagonalization with partial reorthogonalization”. In: *DAIMI Report Series* 27.537 (1998).

[Lia+14] Yingyu Liang et al. “Improved distributed principal component analysis”. In: *Advances in Neural Information Processing Systems.* 2014, pages 3113–3121.

[LN16] Kasper Green Larsen and Jelani Nelson. “Optimality of the Johnson-Lindenstrauss lemma”. In: *arXiv preprint arXiv:1609.02094* (2016).

[Mag10] Malik Magdon-Ismail. “Row sampling for matrix algorithms via a non-commutative Bernstein bound”. In: *arXiv preprint arXiv:1008.0587* (2010).

[Mah+11] Michael W. Mahoney et al. “Randomized algorithms for matrices and data”. In: *Foundations and Trends® in Machine Learning* 3.2 (2011), pages 123–224.

[Mat08] Jiří Matoušek. “On variants of the Johnson–Lindenstrauss lemma”. In: *Random Structures & Algorithms* 33.2 (2008), pages 142–156.

[MD09] Michael W. Mahoney and Petros Drineas. “CUR matrix decompositions for improved data analysis”. In: *Proceedings of the National Academy of Sciences* 106.3 (2009), pages 697–702.

[NDT09] Nam H. Nguyen, Thong T. Do, and Trac D. Tran. “A fast and efficient algorithm for low-rank approximation of a matrix”. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing.* ACM. 2009, pages 215–224.

[NN13] Jelani Nelson and Huy L. Nguyên. “OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings”. In: *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on.* IEEE. 2013, pages 117–126.

[PW15] Mert Pilanci and Martin J. Wainwright. “Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence”. In: *arXiv preprint arXiv:1505.02250* (2015).

[PW16] Mert Pilanci and Martin J. Wainwright. “Iterative Hessian sketch: Fast and accurate solution approximation for constrained least-squares”. In: *The Journal of Machine Learning Research* 17.1 (2016), pages 1842–1879.

[Sar06] Tamás Sarlós. “Improved approximation algorithms for large matrices via random projections”. In: *Foundations of Computer Science, 2006. FOCS’06. 47th Annual IEEE Symposium on*. IEEE. 2006, pages 143–152.

[Shi17] Yaroslav Shitov. “Column subset selection is NP-complete”. In: *arXiv preprint arXiv:1701.02764* (2017).

[Ste13] Willi-Hans Steeb. *Hilbert spaces, wavelets, generalised functions and modern quantum mechanics*. Volume 451. Springer Science & Business Media, 2013.

[Tro+17] Joel A. Tropp et al. “Practical sketching algorithms for low-rank matrix approximation”. In: *SIAM Journal on Matrix Analysis and Applications* 38.4 (2017), pages 1454–1485.

[Tro11] Joel A. Tropp. “Improved analysis of the Subsampled Randomized Hadamard Transform”. In: *Advances in Adaptive Data Analysis* 3 (2011), pages 115–126.

[Ver10] Roman Vershynin. “Introduction to the non-asymptotic analysis of random matrices”. In: *arXiv preprint arXiv:1011.3027* (2010).

[VKO00] Samuli Visuri, Visa Koivunen, and Hannu Oja. “Sign and rank covariance matrices”. In: *Journal of Statistical Planning and Inference* 91.2 (2000), pages 557–575.

[Woo14] David P. Woodruff. “Sketching as a tool for numerical linear algebra”. In: *Foundations and Trends in Theoretical Computer Science* 10.1–2 (2014), pages 1–157.

[Yur+17] Alp Yurtsever et al. “Sketchy decisions: Convex low-rank matrix optimization with optimal storage”. In: *arXiv preprint arXiv:1702.06838* (2017).