Numerical Linear Algebra and Applications in Data Mining

Preliminary version

December 20, 2005

Lars Eldén
Department of Mathematics
Linköping University
Sweden
http://www.mai.liu.se/~laeld/

Preface

This book grew out of a graduate course in *Data mining and applications in science and technology* organized by the Swedish *National Graduate School in Scientific Computing.*

Contents

Part I

Linear Algebra Concepts

Chapter 1

Vectors and Matrices in Data Mining

1.1 Data Mining

In modern society huge amounts of data are collected and stored in computers with the purpose of later extracting useful information.

In modern society huge amounts of data are collected and stored in data bases with the purpose of extracting useful information. Often it is not known at the occasion of collecting the data what information is going to be requested, and therefore the data base is often not designed to distill any particular information, but rather it is to a large extent unstructured. The science of extracting useful information from large data sets is usually referred to as "data mining", sometimes with the addition "knowledge discovery".

There are numerous application areas of data mining, ranging from e-business [9, 62] to bioinformatics [6], from scientific application such as the classification of volcanos on Venus [18] to information retrieval [3] and Internet search engines [10].

Data mining is a truly interdisciplinary science, where techniques from computer science, statistics and data analysis, pattern recognition, linear algebra and optimization are used, often in a rather eclectic manner. Due to the practical importance of the applications, there are now numerous books and surveys in the area. We cite a few here: [21, 22, 27, 31, 41, 42, 43, 45, 96].

It is not an exaggeration to state that everyday life is filled with situations, where we depend critically (and often unknowingly) on advanced mathematical methods for data mining. Linear algebra and data analysis are basic ingredients in many data mining techniques. This book will give some of the mathematical background, applications and new developments.

1.2 Vectors and Matrices in Data Mining

Below we will give examples that illustrate the use of vectors and matrices in data mining. These examples present the main data mining areas discussed in the book, and they will be described in more detail in Part II.

In many applications a matrix is just a rectangular array of data, and the elements are scalar, real numbers,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}.$$

The columns of the matrix will be considered as vectors in $\mathbb{R}^m$.

Example 1.1 [1] *Term-document matrices* are used in *information retrieval*. Consider the following selection of four documents. Key words, which we call *terms*, are marked in boldface[2].

Document 1: The **Google matrix** P is a model of the **Internet.**
Document 2: P_{ij} is nonzero if there is a **link** from **web page** j to i.
Document 3: The **Google matrix** is used to **rank** all **web pages**
Document 4: The **ranking** is done by solving a **matrix eigenvalue** problem. If we
Document 5: **England** dropped out of the top 10 in the **FIFA ranking**.

count the frequency of terms in each document we get the following result.

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5
eigenvalue	0	0	0	1	0
England	0	0	0	0	1
FIFA	0	0	0	0	1
Google	1	0	1	0	0
Internet	1	0	0	0	0
link	0	1	0	0	0
matrix	1	0	1	1	0
page	0	1	1	0	0
rank	0	0	1	1	1
web	0	1	1	0	0

Thus each each document is represented by a vector, or a point, in $\mathbb{R}^8$, and we can

[1] The example is taken from [29].

[2] To avoid making the example too large, we have ignored some words that would normally be considered as terms (key words). Note also that only the stem of the words is significant: "ranking" is considered the same as "rank".

organize them as a term-document matrix,

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Now assume that we want to find all documents that are relevant to the query "**ranking** of **web pages**". This is represented by *query* vector, constructed in an analogous way as the term-document matrix,

$$q = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \in \mathbb{R}^{10}.$$

Thus the query itself is considered as a document. The information retrieval task can now be formulated as a matematical problem: *find the columns of A that are close to the vector q.* To solve this problem we must use some distance measure in $\mathbb{R}^{10}$.

In the information retrieval application it is common that m is large, of the order 10^6, say. Also, as most of the documents only contain a small fraction of the terms, most of the elements in the matrix are equal to zero. Such a matrix is said to be *sparse*.

In some methods for information retrieval one uses linear algebra techniques (e.g. singular value decomposition (SVD)) for data compression and retrieval enhancement. Vector space methods for information retrieval are presented in Chapter 12. ∎

Often, in using a distance measure, it is useful to consider the matrix not just as an array of number but also as a linear operator. Denote the columns of A

$$a_{\cdot j} = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}, \qquad j = 1, 2, \ldots, n,$$

and write

$$A = \begin{pmatrix} a_{\cdot 1} & a_{\cdot 2} & \cdots & a_{\cdot n} \end{pmatrix}.$$

Then the linear transformation given by the matrix A is defined

$$y = Ax = \begin{pmatrix} a_{\cdot 1} & a_{\cdot 2} & \cdots & a_{\cdot n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sum_{j=1}^{n} x_j a_{\cdot j}.$$

Example 1.2 In handwritten digit recognition vectors are used to represent digits. The image of one digit is a 16×16 matrix of numbers, representing grey scale. It can also be represented as a vector in $\mathbb{R}^{256}$, by stacking the columns of the matrix. A set of n digits (handwritten 3's, say) can then be represented by matrix $A \in \mathbb{R}^{256 \times n}$, and the columns of A span a subspace of $\mathbb{R}^{256}$. We can compute an approximate basis of this subspace using the singular value decomposition (SVD) $A = U\Sigma V^T$.

Three basis vectors of the "3-subspace" are illustrated in Figure 1.1. Let b be a

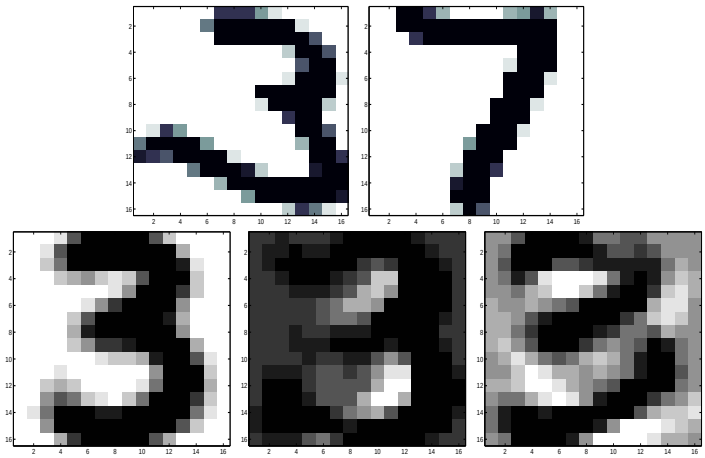


Figure 1.1. *Handwritten digits from the US Postal Service data base [43], and basis vectors for 3's (bottom).*

vector representing an unknown digit. In handwritten digit classification, which is a subarea of *pattern recognition*, one wants to determine (automatically, by computer) which of the digits 0–9 the unknown digit represents. Given a set of basis vectors for 3's, $u_1, u_2, \ldots, u_k$, we can determine whether b is a 3 or not, by checking if there is a linear combination of the basis vectors, $\sum_{j=1}^{k} x_j u_j$, such that

$$b - \sum_{j=1}^{k} x_j u_j$$

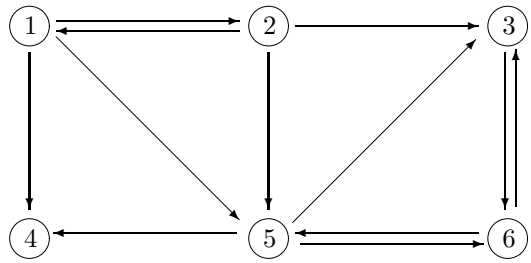
is small. Thus, we here determine the coordinates of b in the basis $\{u_j\}_{j=1}^k$.

In Chapter 11 we discuss methods for classification of handwritten digits. ∎

The very idea of data mining is to extract useful information from large, often unstructured, sets of data. Therefore it is necessary that the methods used are efficient and often specially designed for large problems. In some data mining applications huge matrices occur.

Example 1.3 The task of extracting information from all the web pages available on the Internet, is done by *search engines*. The core of the Google search engine is a matrix computation, probably the largest that is performed routinely [64]. The Google matrix P is of the order billions, i.e. close to the total number of web pages on the Internet. The matrix is constructed based on the link structure of the web, and element P_{ij} is nonzero if there is a link from web page j to i.

The following small link graph illustrates a set of web pages with outlinks and inlinks.



The corresponding matrix becomes

$$P = \begin{pmatrix} 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 & \frac{1}{3} & 0 \end{pmatrix}.$$

For a search engine to be useful, it must somehow use a measure of quality of the web pages. The Google matrix is used to rank all the pages. The ranking is done by solving an eigenvalue problem for P, see Chapter 13. ∎

1.3 Purpose of the book

The present book is not meant to be primarily a textbook in numerical linear algebra, but rather an application-oriented introduction to some techniques in modern linear algebra, with the emphasis on data mining. It depends heavily on the availability of easy-to-use programming environments that implement the algorithms that we will present. Thus, instead of describing in detail the algorithms, we will give enough mathematical theory and numerical background information so that a

student can understand and use the powerful software that is embedded in packages like Matlab [61].

For a more comprehensive presentation of numerical and algorithmic aspects of the matrix decompositions used in this book, we refer the reader to any of the recent textbooks [79, 80, 36, 46, 84, 25]. The solution of linear systems and eigenvalue problems for large and sparse systems are discussed at length in [5, 4]. For those who want to study the detailed implementation of numerical linear algebra algorithms, there is software in Fortran, C, and C++ available for free via the Internet [1].

It will be assumed that the reader has studied introductory courses in linear algebra and scientific computing (numerical analysis). Familiarity with the basics of a matrix-oriented programming language like Matlab probably makes it easier to follow the presentation.

1.4 Programming Environments

In this book we use Matlab [61] to demonstrate the concepts and the algorithms. The codes are not to be considered as software; instead they are intended to demonstrate the basic principles, and we have emphasized simplicity rather than efficiency and robustness. *The codes should only be used for small experiments and never for production domputations.*

Even if we are using Matlab, we want to emphasize that any similar matrix-oriented programming environment, e.g. Mathematica [100], or a statistics package, can be used.

1.5 Floating Point Computations

1.5.1 Flop Counts

The execution times of different algorithms can sometimes be compared by counting the number of *floating point operations*, i.e. arithmetic operations with floating point numbers. In this book we follow the standard procedure [36] and count each operation separately, and we use the term flop for one operation. Thus the statement `y=y+a*x`, where the variables are scalars, counts as 2 flops.

It is customary to count only the highest order term(s). We emphasize that flop counts are often very crude measures of efficiency and computing time, and can even be misleading under certain circumstances. On modern computers, which invariably have memory hierarchies, the data access patterns are very important. Thus there are situations when the execution times of algorithms with the same flop counts can vary by an order of magnitude.

1.5.2 Floating Point Rounding Errors

Error analysis of the algorithms will not be a major part of the book, but we will cite a few results without proofs. We will assume that the computations are done under the IEEE floating point standard [2], and, accordingly, that the following model is valid.

A real number x can, in general, not be represented exactly in a floating point system, but

$$fl[x] = x(1+\epsilon), \tag{1.1}$$

for some ϵ, satisfying $|\epsilon| \leq \mu$, where μ is the *unit roundoff* of the floating point system. From (1.1) we see that the *relative error* in the floating point representation of any real number x satisfies

$$\left| \frac{fl[x] - x}{x} \right| \leq \mu.$$

In IEEE double precision (which is the standard floating point format in Matlab) the unit round off satisfies $\mu \approx 10^{-16}$. In IEEE single precision we have $\mu \approx 10^{-7}$.

Let $fl[x \odot y]$ denote the result of a floating point arithmetic operation.

Theorem 1.4. *Let $\odot$ denote any of the operations $+, -, *$ and $/$. Then, provided that $x \odot y \neq 0$,*

$$\left| \frac{x \odot y - fl[x \odot y]}{x \odot y} \right| \leq \mu, \tag{1.2}$$

or, equivalently,

$$fl[x \odot y] = (x \odot y)(1+\epsilon), \tag{1.3}$$

for some ϵ, satisfying $|\epsilon| \leq \mu$, where μ is the unit roundoff of the floating point system.

When we estimate the error in the *result* of a computation in floating point arithmetic as in (1.2) we can think of it as a *forward error*. Alternatively, we can rewrite (1.3) as

$$fl[x \odot y] = (x+e) \odot (y+f),$$

for some floating point numbers e and f that satisfy

$$|e| \leq \mu|x|, \quad |f| \leq \mu|y|.$$

In other words: $fl[x \odot y]$ is the *exact* result of the operation on *slightly perturbed data*. This is an example of *backward error analysis*.

Example 1.5 (Floating point computations in computer graphics) The detection of a collision between two 3D objects is a standard problem in the application of graphics to computer games, animation and simulation [88]. Earlier fixed point arithmetic was used for computer graphics, but nowadays such computations are routinely done in floating point arithmetic. An important subproblem in this area is the computation of the point on a convex body that is closest to the origin. This problem can be solved by the Gilbert-Johnson-Keerthi algorithm, which is iterative. The algorithm uses the following stopping criterion for the iterations

$$S(v,w) = v^T v - v^T w \leq \epsilon^2,$$

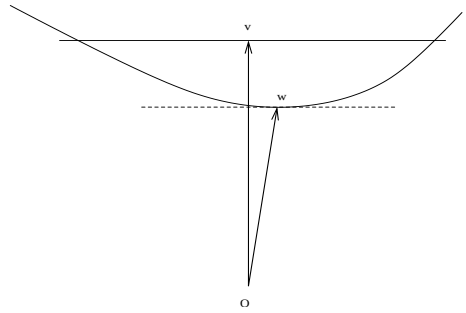


Figure 1.2. *Vectors in the GJK algorithm.*

where the vectors are illustrated in Figure 1.2. As the solution is approached the vectors are very close. In [88][p. 142-145] there is a description of the numerical difficulties that can occur when the computation of $S(v, W)$ is done in floating point arithmetic. We will here give a short explanation in the case of the corresponding computation in the case when v and w are scalar, $s = v^2 - vw$, which exhibits exactly the same problems as in the case of vectors.

Assume that the data are inexact (they are the results of previous computations; in any case they suffer from representation errors (1.1)),

$$\bar{v} = v(1 + \epsilon_v), \quad \bar{w} = w(1 + \epsilon_w),$$

where ϵ_v and ϵ_w are relatively small, often of the order of magnitude of μ. From Theorem 1.4 we see that each arithmetic operation incurs a relative error (1.3), so that

$$\begin{aligned} fl[v^2 - vw] &= (v^2(1 + \epsilon_v)^2(1 + \epsilon_1) - vw(1 + \epsilon_v)(1 + \epsilon_w)(1 + \epsilon_2))(1 + \epsilon_3) \\ &= (v^2 - vw) + v^2(2\epsilon_v + \epsilon_2 + \epsilon_3) - vw(\epsilon_v + \epsilon_w + \epsilon_2 + \epsilon_3) + O(\epsilon^2), \end{aligned}$$

where we have assumed that $|\epsilon_i| \leq \epsilon$. The relative error in the computed quantity can be estimated

$$\left| \frac{fl[v^2 - vw] - (v^2 - vw)}{(v^2 - vw)} \right| \leq \frac{v^2(2|\epsilon_v| + 2\mu) + |vw|(|\epsilon_v| + |\epsilon_w| + 2\mu) + O(\epsilon^2)}{|v^2 - vw|}.$$

We see that if v and w are large, and close, then the relative error may be large. For instance, with $v = 100$, and $w = 99.999$ we get

$$\left| \frac{fl[v^2 - vw] - (v^2 - vw)}{(v^2 - vw)} \right| \leq 10^5((2|\epsilon_v| + 2\mu) + (|\epsilon_v| + |\epsilon_w| + 2\mu) + O(\epsilon^2)).$$

If the computations are performed in IEEE single precision, which is common in computer graphics applications, then the relative error in $fl[v^2 - vw]$ may be so large that the termination criterion is never satisfied, and the iteration will never terminate. In the GJK algorithm there are also other cases than that described above, when floating point rounding errors can cause the termination criterion to be unreliable, and special care must be taken, see [88]. ∎

The problem that occurs in the preceding example is called *cancellation*: when we subtract two almost equal numbers with errors, the result has fewer significant digits, and the relative error is larger. For more details on the IEEE standard and rounding errors in floating point computations, see e.g. [30, Chapter 2]. Extensive rounding error analyses of linear algebra algorithms are given in [47].

1.6 Notation and conventions

We will consider vectors and matrices with real components. Usually vectors will be denoted by small, roman letters, and matrices by capital roman or greek letters,

$$x \in \mathbb{R}^n, \quad A = (a_{ij}) \in \mathbb{R}^{m \times n}.$$

Tensors, i.e., arrays of real numbers with three or more indices, will be denoted by a calligraphic font. For example,

$$\mathcal{S} = (s_{ijk}) \in \mathbb{R}^{n_1 \times n_2 \times n_3}.$$

We will use $\mathbb{R}^m$ to denote the vector space of dimension m over the real field, and $\mathbb{R}^{m \times n}$ for the vectors space of $m \times n$ matrices.

Chapter 2
Vectors and Matrices

We will assume that the basic notions of linear algebra are known to the reader. For completeness, some will be recapitulated here.

2.1 Matrix-Vector Multiplication

The way basic operations in linear algebra are defined is important, since it influences one's mental images of the abstract notions. Sometimes one is lead to to thinking that the operations should be done in a certain order, when instead the *definition as such* imposes no ordering[3]. Let A be an $m \times n$ matrix. Consider the definition of matrix-vector multiplication,

$$y = Ax, \qquad y_i = \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, \ldots, m. \tag{2.1}$$

Symbolically one can illustrate the definition

$$\begin{pmatrix} \times \\ \times \\ \times \\ \times \end{pmatrix} = \begin{pmatrix} \leftarrow & - & - & \rightarrow \\ \leftarrow & - & - & \rightarrow \\ \leftarrow & - & - & \rightarrow \\ \leftarrow & - & - & \rightarrow \end{pmatrix} \begin{pmatrix} \uparrow \\ | \\ | \\ \downarrow \end{pmatrix}. \tag{2.2}$$

It is obvious that the computation of the different components of the vector y are completely independent of each other, and can be done in any order. However, the definition may lead to think that the matrix should be accessed row-wise, as illustrated in (2.2), and in the following code.

```
for i=1:m
  y(i)=0;
```

[3]It is important to be aware that on modern computers, which invariably have memory hierarchies, the order in which operations are performed is often critical for the performance. However, we will not pursue this aspect here.

```
    for j=1:n
      y(i)=y(i)+a(i,j)*x(j);
    end
  end
```

Alternatively, we can write the operation in the following way. Let $a_{\cdot j}$ be a column vector of A. Then we can write

$$y = Ax = \begin{pmatrix} a_{\cdot 1} & a_{\cdot 2} & \cdots & a_{\cdot n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sum_{j=1}^{n} x_j a_{\cdot j}.$$

This can be illustrated symbolically

$$\begin{pmatrix} \uparrow \\ | \\ | \\ \downarrow \end{pmatrix} = \begin{pmatrix} \uparrow & \uparrow & \uparrow & \uparrow \\ | & | & | & | \\ | & | & | & | \\ \downarrow & \downarrow & \downarrow & \downarrow \end{pmatrix} \begin{pmatrix} \times \\ \times \\ \times \\ \times \end{pmatrix}. \tag{2.3}$$

Here the vectors are accessed column-wise. In Matlab, this version can be written[4]

```
y(1:m)=0;
for j=1:n
     y(1:m)=y(1:m)+a(1:m,j)*x(j);
end
```

Here we have used the vector operations of Matlab: the statement inside the `j`-loop corresponds to an `i`-loop. Thus the two ways of performing the matrix-vector multiplication correspond to changing the order of the loops in the code. This way of writing also emphasizes the view of the column vectors of A as *basis vectors*, and the components of x as *coordinates* with respect to the basis.

2.2 Matrix-Matrix Multiplication

Matrix multiplication can be done in several ways, each representing a different access pattern for the matrices. Let $A \in \mathbb{R}^{m\times k}$, and $B \in \mathbb{R}^{k\times n}$. The definition of matrix multiplication is

$$\begin{aligned} \mathbb{R}^{m\times n} \ni C = AB = (c_{ij}), \\ c_{ij} = \sum_{s=1}^{k} a_{is} b_{sj}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n. \end{aligned} \tag{2.4}$$

If we compare to the definition of matrix-vector multiplication (2.1), we see that in matrix multiplication *each column vector in B is multiplied by A.*

We can formulate (2.4) as a matrix multiplication code

[4] In the terminology of LAPACK [1] this is the SAXPY version of matrix-vector multiplication. SAXPY is an acronym from the BLAS library.

```
for i=1:m
  for j=1:n
    for s=1:k
      c(i,j)=c(i,j)+a(i,s)*b(s,j)
    end
  end
end
```

This is an inner product version of matrix multiplication, which is emphasized in the following equivalent code:

```
for i=1:m
  for j=1:n
    c(i,j)=a(i,1:k)*b(1:k,j)
  end
end
```

It is immediately seen that the the loop variables can be permuted in $3! = 6$ different ways, and we can write a *generic matrix multiplication code*.

```
for ...
  for ...
    for ...
      c(i,j)=c(i,j)+a(i,s)*b(s,j)
    end
  end
end
```

A column-oriented (or SAXPY) version is given in

```
for j=1:n
  for s=1:k
    c(1:m,j)=c(1:m,j)+a(1:m,s)*b(s,j)
  end
end
```

The matrix A is accessed by columns and B by scalars. This access pattern can be illustrated

$$\begin{pmatrix} & \uparrow & \\ & | & \\ & | & \\ & \downarrow & \end{pmatrix} = \begin{pmatrix} \uparrow & \uparrow & \uparrow & \uparrow \\ | & | & | & | \\ | & | & | & | \\ \downarrow & \downarrow & \downarrow & \downarrow \end{pmatrix} \begin{pmatrix} & \times & \\ & \times & \\ & \times & \\ & \times & \end{pmatrix}$$

In another permutation we let the `s`-loop be the outermost:

```
for s=1:k
  for j=1:n
    c(1:m,j)=c(1:m,j)+a(1:m,s)*b(s,j)
  end
end
```

This can be illustrated as follows: Let $a_{\cdot k}$ denote the column vectors of A and $b_{k\cdot}^T$ the row vectors of B. Then matrix multiplication can be written

$$C = AB = \begin{pmatrix} a_{\cdot 1} & a_{\cdot 2} & \dots & a_{\cdot k} \end{pmatrix} \begin{pmatrix} b_{1\cdot}^T \\ b_{2\cdot}^T \\ \vdots \\ b_{k\cdot}^T \end{pmatrix} = \sum_{s=1}^{k} a_{\cdot s} b_{s\cdot}^T,$$

This is the *outer product* form of matrix multiplication. We remind that the outer product follows the standard definition of matrix multiplication: Let x and y be column vectors in $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively. Then

$$xy^T = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix} = \begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & & \vdots \\ x_my_1 & x_my_2 & \cdots & x_my_n \end{pmatrix}$$

$$= \begin{pmatrix} y_1x & y_2x & \cdots & y_nx \end{pmatrix} = \begin{pmatrix} x_1y^T \\ x_2y^T \\ \vdots \\ x_my^T \end{pmatrix}.$$

2.3 Scalar Product and Vector Norms

In this section we will discuss briefly how to measure the "size" of a vector. The most common vector norms are

$$\| x \|_1 = \sum_{i=1}^{n} |x_i|, \quad \text{1-norm}$$

$$\| x \|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}, \quad \text{Euclidean norm,}$$

$$\| x \|_\infty = \max_{1 \le i \le n} |x_i|, \quad \text{max-norm.}$$

The Euclidean vector norm is the generalization of the standard Euclidean distance in $\mathbb{R}^3$ to $\mathbb{R}^n$. All three norms defined here are special cases of the p-norm:

$$\| x \|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

Associated with the Euclidean vector norm is the *inner product* between two vectors x and y in $\mathbb{R}^n$ is defined

$$(x, y) = x^T y.$$

Generally, a *vector norm* is a mapping $\mathbb{R}^n \to \mathbb{R}$, with the properties

$$\begin{aligned} \| x \| &\geq 0, \quad \text{for all } x, \\ \| x \| &= 0, \text{ if and only if } x = 0, \\ \| \alpha x \| &= |\alpha| \, \| x \|, \ \alpha \in \mathbb{R}, \\ \| x + y \| &\leq \| x \| + \| y \|, \quad \text{the triangle inequality.} \end{aligned}$$

With norms we can introduce the concepts of continuity and error in approximations of vectors. Let $\bar{x}$ be an approximation of the vector x. The for any given vector norm, we define the *absolute error*

$$\| \delta x \| = \| \bar{x} - x \|,$$

and the *relative error* (assuming that $x \neq 0$)

$$\frac{\| \delta x \|}{\| x \|} = \frac{\| \bar{x} - x \|}{\| x \|}.$$

In a finite dimensional vector space all vector norms are equivalent in the sense that for any two norms $\| \cdot \|_\alpha$ and $\| \cdot \|_\beta$ there exist constants m and M, such that

$$m \| x \|_\alpha \leq \| x \|_\beta \leq M \| x \|_\alpha, \tag{2.5}$$

where m and M do not depend on x. For example, with $x \in \mathbb{R}^n$,

$$\| x \|_2 \leq \| x \|_1 \leq \sqrt{n} \, \| x \|_2.$$

This equivalence implies that if a sequence of vectors $(x_i)_{i=1}^{\infty}$ converges to x^* in one norm,

$$\lim_{i \to \infty} \| x_i - x^* \| = 0,$$

then it converges to the same limit in all norms.

In data-mining application it is common to use the *cosine of the angle* between two vectors as a distance measure:

$$\cos \theta(x, y) = \frac{x^T y}{\| x \|_2 \, \| y \|_2}.$$

With this measure two vectors are close if the cosine is close to one. Similarly, x and y are *orthogonal* if the angle between them is $\pi/2$, i.e. $x^T y = 0$.

2.4 Matrix Norms

For any vector norm we can define a corresponding *operator norm* : Let $\| \cdot \|$ be a vector norm. The corresponding *matrix norm* is defined

$$\| A \| = \sup_{x \neq 0} \frac{\| Ax \|}{\| x \|}.$$

One can show that such a matrix norm satisfies (for $\alpha \in \mathbb{R}$):

$$\begin{aligned} \| A \| &\geq 0 \quad \text{for all } A, \\ \| A \| &= 0 \text{ if and only if } A = 0, \\ \| \alpha A \| &= |\alpha| \, \| A \|, \alpha \in \mathbb{R} \\ \| A + B \| &\leq \| A \| + \| B \|, \quad \text{the triangle inequality.} \end{aligned}$$

For a matrix norm defined as above the following fundamental inequalities hold:

Proposition 2.1. *Let $\| \cdot \|$ denote a vector norm and the corresponding matrix norm. Then*

$$\begin{aligned} \| Ax \| &\leq \| A \| \, \| x \|, \\ \| AB \| &\leq \| A \| \, \| B \|. \end{aligned}$$

Proof. From the definition we have

$$\frac{\| Ax \|}{\| x \|} \leq \| A \|,$$

for all $x \neq 0$. The second inequality is obtained by using the first twice for $\| ABx \|$. □

One can show that the *2-norm* satisfies

$$\| A \|_2 = (\max_{1 \leq i \leq n} \lambda_i (A^T A))^{1/2},$$

i.e. the square root of the largest eigenvalue of the matrix $A^T A$. Thus it is a comparatively heavy computation to obtain $\| A \|_2$ for a given matrix (of medium or large dimensions). It is considerably easier to compute the *matrix maximum norm* (for $A \in \mathbb{R}^{m \times n}$)

$$\| A \|_\infty = \max_{1 \leq i \leq m} \sum_{j=1}^{n} |a_{ij}|,$$

and the *matrix 1-norm*

$$\| A \|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|.$$

In Section 6.1 we will see that the 2-norm of a matrix has an explicit expression in terms of the singular values of A.

Let $A \in \mathbb{R}^{m \times n}$. In some cases we will not treat the matrix as a linear operator, but rather as a point in a space of dimension mn, i.e. $\mathbb{R}^{mn}$. Then we can use the *Frobenius* matrix norm, which is defined

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}. \tag{2.6}$$

Sometimes it is practical to write the Frobenius norm in the following, equivalent way.

$$\| A \|_F^2 = \mathrm{tr}(A^T A), \tag{2.7}$$

where the *trace* of a matrix $A \in \mathbb{R}^{n\times n}$ is given by

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii}.$$

The Frobenius norm does not correspond to a vector norm, so it is not an operator norm in that sense. This norm has the advantage that it is easier to compute than the 2-norm. The Frobenius *matrix norm* is actually closely related to the Euclidean *vector norm* in the sense that it is the Euclidean vector norm on the (linear space) of matrices $\mathbb{R}^{m\times n}$, when the matrices are identified with elements in $\mathbb{R}^{mn}$.

2.5 Linear Independence – Bases

Given a set of vectors $(v_j)_{j=1}^n$ in $\mathbb{R}^m$, $m \geq n$, and consider the set of linear combinations

$$\mathrm{span}(v_1, v_2, \ldots, v_n) = \{y \mid y = \sum_{j=1}^{n} \alpha_j v_j\}$$

for arbitrary coefficients α_j. The vectors $(v_j)_{j=1}^n$ are called *linearly independent* if

$$\textstyle\sum_{j=1}^n \alpha_j v_j = 0 \;\text{ if and only if }\; \alpha_j = 0, \text{ for } j = 1, 2, \ldots, n.$$

A set of m linearly independent vectors in $\mathbb{R}^m$ are called a *basis* in $\mathbb{R}^m$: any vector in $\mathbb{R}^m$ can be expressed as a linear combination of the basis vectors.

Proposition 2.2. *Assume that the vectors $(v_j)_{j=1}^n$ are linearly dependent. Then some v_k can be written as linear combinations of the rest, $v_k = \sum_{j\neq k} \beta_j v_j$.*

Proof. There exists coefficients α_j, with some $\alpha_k \neq 0$, such that

$$\sum_{j=1}^{n} \alpha_j v_j = 0.$$

Take a $\alpha_k \neq 0$ and write

$$\alpha_k v_k = \sum_{j\neq k} -\alpha_j v_j,$$

which is the same as

$$v_k = \sum_{j\neq k} \beta_j v_j, \qquad \beta_j = -\frac{\alpha_j}{\alpha_k}.$$

□

If we have a set of linearly dependent vectors, then we can keep a linearly independent subset, and express the rest in terms of the linearly independent ones.

Thus we can consider the number of linearly independent columns as a measure of the information contents of the set, and compress the set accordingly: Take the linearly independent vectors as representatives (basis vectors) for the set, and compute the coordinates of the rest in terms of the basis. However, in real applications we seldom have *exactly linearly dependent vectors* but rather *almost linearly dependent vectors.* It turns out that for such a data reduction procedure to be practical and numerically stable, we need the basis vectors not only to be linearly independent, but orthogonal. We will come back to this in Chapter 4.

2.6 The Rank of a Matrix

The *rank* of a matrix is defined as the maximum number of linearly independent column vectors.

It is a standard result in linear algebra that the number of linearly independent column vectors is equal to the number of linearly independent row vectors.

Proposition 2.3. *An outer product matrix xy^T, where x and y are vectors in $\mathbb{R}^n$, has rank 1.*

Proof.

$$xy^T = \begin{pmatrix} y_1 x & y_2 x & \cdots & y_n x \end{pmatrix} = \begin{pmatrix} x_1 y^T \\ x_2 y^T \\ \vdots \\ x_n y^T \end{pmatrix}$$

Thus, all the columns (rows) of xy^T are linearly dependent. □

A square matrix $A \in \mathbb{R}^{n \times n}$ with rank n is called *nonsingular*, and has an *inverse* A^{-1} satisfying

$$AA^{-1} = A^{-1}A = I.$$

If we multiply linearly independent vectors by a nonsingular matrix, then the vectors remain linearly independent.

Proposition 2.4. *Assume that the vectors $v_1, \ldots, v_p$ are linearly independent. Then for any nonsingular matrix T, the vectors $Tv_1, \ldots, Tv_p$ are linearly independent.*

Proof. Obviously $\sum_{j=1}^{p} \alpha_j v_j = 0$ if and only if $\sum_{j=1}^{p} \alpha_j T v_j = 0$ (since we can multiply any of the equations by T or T^{-1}). Therefore the statement follows. □

Exercises

2.1. Draw a sketch of the unit circle $\| x \| = 1$ in $\mathbb{R}^2$ and $\mathbb{R}^3$ for $\| \cdot \|_1$, $\| \cdot \|_2$, and $\| \cdot \|_\infty$?

2.2. Prove that $\lim_{p\to\infty} \| x \|_p = \lim_{p\to\infty} \left(\sum_{i=1}^n |x_i|^p\right)^{1/p} = \| x \|_\infty$.

2.3. Prove that for $D = \mathrm{diag}(d_1, \cdots, d_n)$

$$\|D\|_p = \max_{1\le i\le n} |d_{ii}|.$$

Chapter 3

Linear Systems and Least Squares

In this chapter we will briefly review some facts about the solution of linear systems of equations

$$Ax = b, \tag{3.1}$$

where $A \in \mathbb{R}^{n\times n}$ is square and nonsingular. We will also consider *overdetermined linear systems*, where the matrix $A \in \mathbb{R}^{m\times n}$ is *rectangular* with $m > n$, and their solution using the least squares method.

The linear system (3.1) can be solved using *Gaussian elimination* with *partial pivoting*, which is equivalent to factorizing the matrix as a product of triangular matrices. As we are only giving these results as a background, we mostly state them without proofs. For thorough presentations of the theory of matrix decompositions for solving linear systems of equations, we refer, e.g., to [36, 79].

Before discussing matrix decompositions, we state the basic result concerning conditions for the existence of a unique solution of (3.1).

Proposition 3.1. *Let $A \in \mathbb{R}^{n\times n}$ and assume that A is nonsingular. Then for any right hand side b, the linear system $Ax = b$ has a unique solution.*

Proof. The result is an immediate consequence of the fact that the column vectors of a nonsingular matrix are linearly independent. □

3.1 LU Decomposition

Gaussian elimination can be conveniently described using *Gauss transformations*, and these transformations are the key elements in the equivalence between Gaussian elimination and LU decomposition. More details on Gauss transformations can be found in any textbook in numerical linear algebra, see e.g. [36, p. 94]. In Gaussian elimination with partial pivoting the reordering of the rows are accomplished by *permutation matrices*, which are identity matrices with the rows reordered, see e.g. [36, Section 3.4.1].

Consider an $N \times N$ matrix A. In the first step of Gaussian elimination with partial pivoting, we reorder the rows of the matrix so that the element of largest magnitude in the first column is moved to the (1,1) position. This is equivalent to multiplying A from the left by a permutation matrix P_1. The elimination, i.e., the zeroing of the elements in the first column below the diagonal, is then performed by multiplying

$$A^{(1)} := L_1^{-1} P_1 A, \tag{3.2}$$

where L_1 is a Gauss transformation

$$L_1 = \begin{pmatrix} 1 & 0 \\ m_1 & I \end{pmatrix}, \qquad m_1 = \begin{pmatrix} m_{21} \\ m_{31} \\ \vdots \\ m_{N1} \end{pmatrix}.$$

The result of the first step of Gaussian elimination with partial pivoting is

$$A^{(1)} = \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1N} \\ 0 & a_{22}^{(1)} & \dots & a_{2N}^{(1)} \\ \vdots & & & \\ 0 & a_{N2}^{(1)} & \dots & a_{NN}^{(1)} \end{pmatrix}.$$

The Gaussian elimination algorithm then proceeds by zeroing the elements of the second column below the main diagonal (after moving the largest element to the diagonal position), and so on.

From (3.2) we see that the first step of Gaussian elimination with partial pivoting can be expressed as a matrix factorization. This is also true of the complete procedure.

Theorem 3.2 (LU decomposition.). *Any nonsingular $n \times n$ matrix A can be decomposed into*

$$PA = LU,$$

where P is a permutation matrix, L is a lower triangular matrix with ones on the main diagonal, and U is an upper triangular matrix.

Proof. (Sketch). The theorem can be proved by induction. From (3.2) we have

$$P_1 A = L_1 A^{(1)}.$$

Define the $(N-1) \times (N-1)$ matrix

$$B = \begin{pmatrix} a_{22}^{(1)} & \dots & a_{2N}^{(1)} \\ \vdots & & \\ a_{N2}^{(1)} & \dots & a_{NN}^{(1)} \end{pmatrix}.$$

By an induction assumption, B can be decomposed into

$$P_B B = L_B U_B,$$

and we then see that $PA = LU$, where

$$U = \begin{pmatrix} a'_{11} & a_2^T \\ 0 & U_B \end{pmatrix}, \qquad L = \begin{pmatrix} 1 & 0 \\ P_B m_1 & L_B \end{pmatrix}, \qquad P = \begin{pmatrix} 1 & 0 \\ 0 & P_B \end{pmatrix} P_1,$$

and $a_2^T = (a'_{12}\, a'_{13}\, \ldots\, a'_{1N})$. □

It is easy to show that the amount of work for computing the LU decomposition is $2n^3/3$ flops, approximately: In the k'th step of Gaussian elimination one operates on an $(n-k+1) \times (n-k+1)$ submatrix, and for each element in that submatrix one multiplication and one addition is performed. Thus the total number of flops is

$$2\sum_{k=1}^{n-1}(n-k+1)^2 \approx \frac{2n^3}{3},$$

approximately.

3.2 Symmetric, Positive Definite Matrices

The LU decomposition of a symmetric, positive definite matrix A can always be computed without pivoting. In addition, it is possible to take advantage of symmetry so that the decomposition becomes symmetric, too, and requires half as much work as in the general case.

Theorem 3.3 (LDL^T decomposition). *Any symmetric, positive definite matrix A has a decomposition*

$$A = LDL^T,$$

where L is lower triangular with ones on the main diagonal, and D is a diagonal matrix with positive diagonal elements.

Example 3.4 The positive definite matrix

$$A = \begin{pmatrix} 8 & 4 & 2 \\ 4 & 6 & 0 \\ 2 & 0 & 3 \end{pmatrix},$$

has the LU decomposition

$$A = LU = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.25 & -0.25 & 1 \end{pmatrix} \begin{pmatrix} 8 & 4 & 2 \\ 0 & 4 & -1 \\ 0 & 0 & 2.25 \end{pmatrix},$$

and the LDL^T decomposition

$$A = LDL^T, \quad D = \begin{pmatrix} 8 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2.25 \end{pmatrix}.$$

■

The diagonal elements in D are positive, and therefore we can put

$$D^{1/2} = \begin{pmatrix} \sqrt{d_1} & & & \\ & \sqrt{d_2} & & \\ & & \ddots & \\ & & & \sqrt{d_n} \end{pmatrix},$$

and then we get

$$A = LDL^T = (LD^{1/2})(D^{1/2}L^T) = U^T U,$$

where U is an upper triangular matrix. This variant of the LDL^T decomposition is called the *Cholesky decomposition*.

Since A is symmetric, it is only necessary to store the main diagonal and the elements above it, $n(n+1)/2$ matrix elements in all. Exactly the same amount of storage is needed for the LDL^T and the Cholesky decompositions. It is also seen that since only half as many elements as in the ordinary LU decomposition need be computed, the amount of work is also halved, approximately $n^3/6$ operations. When the LDL^T decomposition is computed, it is not necessary to first compute the LU decomposition, but the elements in L and D can be computed directly.

3.3 Perturbation Theory and Condition Number

The *condition number* of a nonsingular matrix A is defined as

$$\kappa(A) = \|A\| \, \|A^{-1}\|,$$

where $\|\cdot\|$ denotes any operator norm. If we use a particular matrix norm, e.g. the 2-norm, then we write

$$\kappa_2(A) = \|A\|_2 \, \|A^{-1}\|_2. \tag{3.3}$$

The condition number is used to quantify how much the solution of a linear system $Ax = b$ is changed, when the matrix and the right hand side are perturbed by a small amount.

Theorem 3.5. *Assume that A is nonsingular and that*

$$\|\delta A\| \, \|A^{-1}\| = r < 1.$$

Then the matrix $A + \delta A$ is nonsingular, and

$$\|(A + \delta A)^{-1}\| \leq \frac{\|A^{-1}\|}{1 - r}.$$

The solution of the perturbed system

$$(A + \delta A)y = b + \delta b$$

satisfies

$$\frac{\|y - x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - r}\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right).$$

A matrix with a large condition number is said to be *ill-conditioned*. Theorem 3.5 shows that a linear system with an ill-conditioned matrix is sensitive to perturbations in the data (i.e. the matrix and the right hand side).

3.4 Rounding Errors in Gaussian Elimination

From Section 1.5.2 on rounding errors in floating point arithmetic, we know that any real number (representable in the floating point system) is represented with a relative error not exceeding the unit roundoff μ. This fact can also be stated

$$fl[x] = x(1 + \epsilon), \qquad |\epsilon| \leq \mu.$$

When representing the elements of a matrix A and a vector b in the floating point system, there arise errors:

$$fl[a_{ij}] = a_{ij}(1 + \epsilon_{ij}), \qquad |\epsilon_{ij}| \leq \mu,$$

and analogously for b. Therefore, we can write

$$\begin{aligned} fl[A] &= A + \delta A, \\ fl[b] &= b + \delta b, \end{aligned}$$

where

$$\begin{aligned} \|\delta A\|_\infty &\leq \mu \|A\|_\infty, \\ \|\delta b\|_\infty &\leq \mu \|b\|_\infty. \end{aligned}$$

If, for the moment, we assume that no further rounding errors arise during the solution of the system $Ax = b$, we see that the computed solution $\hat{x}$ is the exact solution of

$$(A + \delta A)\hat{x} = b + \delta b.$$

From Theorem 3.5, we then get

$$\frac{\|\hat{x} - x\|_\infty}{\|x\|_\infty} \leq \frac{\kappa_\infty(A)}{1 - r}\, 2\mu,$$

(provided that $r = \mu\kappa_\infty(A) < 1$).

This is an example of *backward error analysis*: The computed solution $\hat{x}$ is the exact solution of a *perturbed* problem

$$(A + \delta A)\hat{x} = b + \delta b.$$

Using perturbation theory, we can estimate the error in $\hat{x}$.

We can also analyze how rounding errors in Gaussian elimination affect the result. The following theorem holds.

Theorem 3.6. *Assume that we use a floating point system with unit roundoff μ. Let $\hat{L}$ and $\hat{R}$ be the triangular factors obtained from Gaussian elimination with partial pivoting, applied to the matrix A. Further, assume that $\hat{x}$ is computed using forward and back substitution:*

$$\hat{L}\hat{y} = Pb, \quad \hat{R}\hat{x} = \hat{y}.$$

Then $\hat{x}$ is the exact solution of a system

$$(A + \delta A)\hat{x} = b,$$

where

$$\|\delta A\|_\infty \leq k_2(n)\mu\|A\|_\infty, \qquad k_2(n) = (n^3 + 3n^2)g_n,$$

$$g_n = \frac{\max\limits_{i,j,k} |\hat{a}_{ij}^{(k)}|}{\max\limits_{i,j} |a_{ij}|}.$$

($\hat{a}_{ij}^{(k)}$ are the elements computed in step $k-1$ of the elimination procedure).

We observe that g_n depends on the growth of the matrix elements during the Gaussian elimination, and not explicitly on the magnitude of the multipliers. g_n can be computed, and this way an a *posteriori* estimate of the rounding errors can be obtained.

A priori (in advance), one can show that $g_n \leq 2^{n-1}$, and matrices can be constructed where in fact the element growth is that serious (note that $g_{31} = 2^{30} \approx 10^9$). In practice, however, g_n is seldom larger than 8 in Gaussian elimination with partial pivoting.

However, there are classes of matrices, for which there is no element growth during Gaussian elimination, i.e., $g_n = 1$, even if no pivoting is done. This is true, e.g. if A is symmetric and positive definite.

We want to emphasize that the estimate in the theorem, in almost all cases, is much too pessimistic. In order to have equality, all rounding errors must be maximally large (equal to μ), and their accumulated effect must be maximally unfavourable. Therefore, the main object of this type of a priori error analysis is not to give error estimates for the solution of linear systems, but rather to expose potential instabilities of algorithms, and provide a basis for comparing different algorithms. Thus, Theorem 3.6 demonstrates the main weakness of Gauss transformations as compared to the orthogonal transformations that we will introduce in Chapter 4: they can cause a large growth of the matrix matrix elements, which in turn, induces rounding errors.

We close this section with a few examples, which further illustrate perturbation and error analysis.

Example 3.7 Consider the matrix

$$A = \begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix}.$$

This matrix has the inverse

$$A^{-1} = \frac{1}{\epsilon - 1} \begin{pmatrix} 1 & -1 \\ -1 & \epsilon \end{pmatrix},$$

and the condition number $\kappa_\infty(A) \approx 4$, if ϵ is small. Thus the *problem* of solving $Ax = b$ is well-conditioned.

If we use Gaussian elimination *without* pivoting, we get an enormous growth in the matrix elements:

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ 0 & 1 - 1/\epsilon \end{pmatrix}.$$

According to Theorem 3.6, we may now expect large errors in the solution. The factors L and U are very ill-conditioned:

$$\kappa_\infty(L) \approx \frac{2}{\epsilon}, \quad \kappa_\infty(U) \approx \frac{1}{\epsilon}.$$

The *algorithm* is unstable.

The LU decomposition of the same matrix with permuted rows is

$$\begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 - \epsilon \end{pmatrix}.$$

No growth of matrix elements occurs, and the factors are very well-conditioned:

$$\kappa_\infty(L) \approx 1, \quad \kappa_\infty(U) \approx 4.$$

■

Example 3.8 The matrix

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix},$$

is symmetric and positive definite. It is well-conditioned: $\kappa_\infty(A) = 3$. The LDL^T decomposition is

$$A = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & \frac{3}{2} \end{pmatrix} \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{pmatrix}.$$

No growth of matrix elements occurs.

Next, consider the matrix

$$B = \begin{pmatrix} \epsilon & 1 \\ 1 & 1/\epsilon + 1 \end{pmatrix}.$$

B is also symmetric and positive definite. It is ill-conditioned: $\kappa_\infty(A) \approx 1/\epsilon^2$, and has the LDL^T decomposition

$$B = \begin{pmatrix} 1 & 0 \\ 1/\epsilon & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1/\epsilon \\ 0 & 1 \end{pmatrix}.$$

No growth of matrix elements occurs. The problem of solving $Bx = c$ is indeed ill-conditioned, but the algorithm (Gaussian elimination *without* pivoting) does not introduce any *unnecessary* loss of accuracy. ∎

3.5 Banded Matrices

In many situations, e.g., boundary value problems for ordinary and partial differential equations, matrices arise where a large proportion of the elements are equal to zero. If the nonzero elements are concentrated around the main diagonal, then the matrix is called a band matrix. More precisely, a matrix A is said to be a *band matrix* if there are natural numbers p and q, such that

$$a_{ij} = 0 \text{ if } j - i > p \text{ or } i - j > q.$$

Example 3.9 Let $q = 2$, $p = 1$. Let A be a band matrix of dimension 6:

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} \end{pmatrix}$$

∎

$w = q + p + 1$ is called the *band width* of the matrix. From the example, we see that w is the maximal number of nonzero elements in any row of A.

When storing a band matrix, we do not store the elements outside the band. Likewise, when linear systems of equations are solved, one can take advantage of the band structure to reduce the number of operations.

We first consider the case $p = q = 1$. Such a band matrix is called *tridiagonal*. Let

$$A = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \gamma_2 & \alpha_2 & \beta_2 & & & \\ & \gamma_3 & \alpha_3 & \beta_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \gamma_{n-1} & \alpha_{n-1} & \beta_{n-1} \\ & & & & \gamma_n & \alpha_n \end{pmatrix}.$$

The matrix can be stored in three vectors. In the solution of a tridiagonal system $Ax = b$, it is easy to utilize the structure; we first assume that A is diagonally dominant, so that no pivoting is needed.

```
% LU Decomposition of a Tridiagonal Matrix.
for k=1:n-1
  gamma(k+1)=gamma(k+1)/alpha(k);
  alpha(k+1)=alpha(k+1)*beta(k);
end
% Forward Substitution for the Solution of Ly = b.
y(1)=b(1);
for k=2:n
  y(k)=b(k)-gamma(k)*y(k-1);
end
% Back Substitution for the Solution of $Ux = y$.}
x(n)=y(n)/alpha(n);
for k=n-1:-1:1
  x(k)=(y(k)-beta(k)*x(k+1))/alpha(k);
end
```

The number of operations (multiplications and additions) is approximately $3n$, and the number of divisions is $2n$.

In Gaussian elimination *with partial pivoting*, the band width of the upper triangular matrix increases. If A has band width $w = q + p + 1$ (q diagonals under the main diagonal and p over), then, with partial pivoting, the factor U will get band width $w_U = p + q + 1$. It is easy to see that no new nonzero elements will be created in L.

While the factors L and U in the LU decomposition of a band matrix A are band matrices, it turns out that A^{-1} is usually a dense matrix. Therefore, in most cases the inverse of a band matrix should not be computed explicitly.

Example 3.10 Let

$$A = \begin{pmatrix} 4 & 2 & & & \\ 2 & 5 & 2 & & \\ & 2 & 5 & 2 & \\ & & 2 & 5 & 2 \\ & & & 2 & 5 \end{pmatrix}.$$

A has the Cholesky decomposition $A = U^T U$, where

$$U = \begin{pmatrix} 2 & 1 & & & \\ & 2 & 1 & & \\ & & 2 & 1 & \\ & & & 2 & 1 \\ & & & & 2 \end{pmatrix}.$$

The inverse is

$$A^{-1} = \frac{1}{2^{10}} \begin{pmatrix} 341 & -170 & 84 & -40 & 16 \\ -170 & 340 & -168 & 80 & -32 \\ 84 & -168 & 336 & -160 & 64 \\ -40 & 80 & -160 & 320 & -128 \\ 16 & -32 & 64 & -128 & 256 \end{pmatrix}.$$

3.6 The Least Squares Problem

In this section we will introduce the least squares method, and solution of the linear least squares problem using the normal equations. Other methods for solving the least squares problem, based on orthogonal transformations, will be presented in Chapters 5 and 6. We will also give a perturbation result for the least squares problem in Section 6.6. For an extensive treatment of modern numerical methods for linear least squares problem, see [13].

Example 3.11 Assume that we want to determine the elasticity constant of a spring. Hooke's law states that the length l depends on the force F according to

$$e + \kappa F = l,$$

where κ is the elasticity constant. Assume that we have performed an experiment and obtained that data

F	1	2	3	4	5
l	7.97	10.2	14.2	16.0	21.2

The data are illustrated in Figure 3.1. As the measurements are subject to error

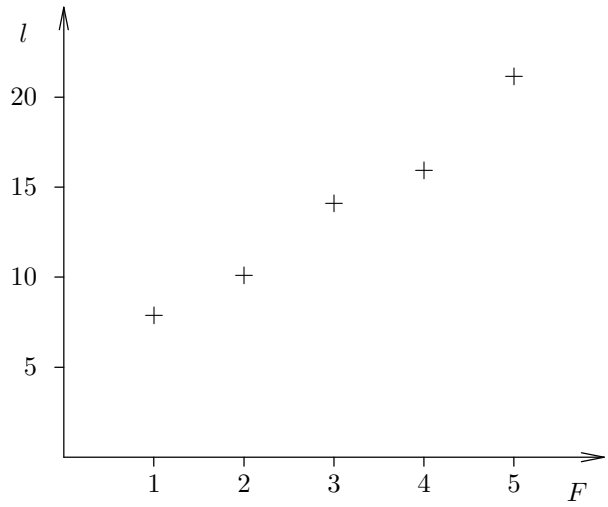


Figure 3.1. *Measured data in spring experiment.*

we want to use all the data in order to minimize the influence of the errors. Thus we are lead to a system with more data than unknowns, an *overdetermined system*

$$e + \kappa 1 = 7.97,$$

$$
\begin{aligned}
e + \kappa 2 &= 10.2,\\
e + \kappa 3 &= 14.2,\\
e + \kappa 4 &= 16.0,\\
e + \kappa 5 &= 21.2,
\end{aligned}
$$

or, in matrix form

$$
\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{pmatrix} \begin{pmatrix} e \\ \kappa \end{pmatrix} = \begin{pmatrix} 7.97 \\ 10.2 \\ 14.2 \\ 16.0 \\ 21.2 \end{pmatrix}.
$$

We will determine an approximation of the elasticity constant of the spring using the least squares method. ∎

Let $A \in \mathbb{R}^{m\times n}$, $m > n$. The system

$$Ax = b$$

is called *overdetermined*: it has more equations than unknowns. In general such a system has no solution. This can be seen geometrically by letting $m = 3$ and $n = 2$, i.e. we consider two vectors $a_{\cdot 1}$ and $a_{\cdot 2}$ in $\mathbb{R}^3$. We want to find a linear combination of the vectors such that

$$x_1 a_{\cdot 1} + x_2 a_{\cdot 2} = b.$$

In Figure 3.2 we see that usually such a problem has no solution. The two vectors

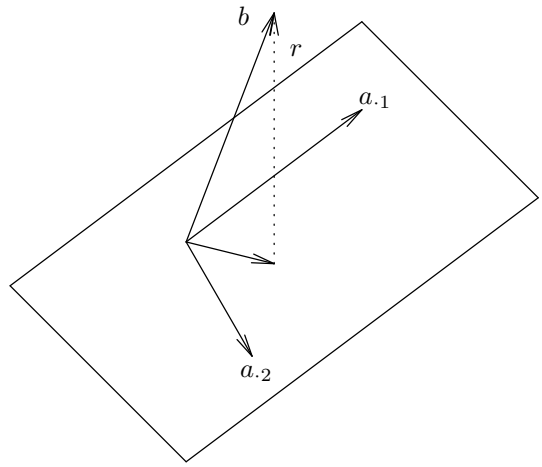


Figure 3.2. *The least squares problem, $m = 3$ and $n = 2$. The residual vector $b - Ax$ is dotted.*

span a plane, and if the right hand side b is not in the plane, then there is no linear combination of $a_{\cdot 1}$ and $a_{\cdot 2}$ such that $x_1 a_{\cdot 1} + x_2 a_{\cdot 2} = b$.

In this situation one obvious alternative to "solving the linear system" is to make the vector $r = b - x_1 a_{\cdot 1} - x_2 a_{\cdot 2} = b - Ax$ as small as possible. $b - Ax$ is called the *residual vector*, and is illustrated in Figure 3.2.

The solution of the problem depends on how we measure the length of the residual vector. In the *least squares method* we use the standard Euclidean distance. Thus we want to find a vector $x \in \mathbb{R}^n$, that solves the minimization problem

$$\min_x \| b - Ax \|_2. \tag{3.4}$$

As the unknown x occurs linearly in (3.4), this is also referred to as the *linear least squares problem.*

In the example we know immediately from our knowledge of distances in $\mathbb{R}^3$, that the distance between the tip of the vector b and the plane is minimized, if we choose the linear combination of vectors in the plane in such a way that the residual vector is orthogonal to the plane. Since the columns of the matrix A span the plane, we see that we get the solution by *making r orthogonal to the columns of A.* This geometric intuition is valid also in the general case:

$$r^T a_{\cdot j} = 0, \qquad j = 1, 2, \ldots, n.$$

(Cf. the definition of orthogonality in Section 2.3.) Equivalently, we can write

$$r^T \begin{pmatrix} a_{\cdot 1} & a_{\cdot 2} & \cdots & a_{\cdot n} \end{pmatrix} = r^T A = 0.$$

Then, using $r = b - Ax$, we get the *normal equations* (the name is now obvious):

$$A^T A x = A^T b,$$

for the determining the coefficients in x.

Theorem 3.12. *If the columns vectors of A are linearly independent, then the normal equations*

$$A^T A x = A^T b.$$

are non-singular and have a unique solution.

Proof. We first show that $A^T A$ is positive definite. Let x be an arbitrary nonzero vector. Then, from the definition of linear independence, we have $Ax \neq 0$. With $y = Ax$, we then have

$$x^T A^T A x = y^T y = \sum_{i=1}^{n} y_i^2 > 0,$$

which is equivalent to $A^T A$ being positive definite. Therefore, $A^T A$ is nonsingular, and the normal equations have a unique solution, which we denote $\hat{x}$.

Then, we show that $\hat{x}$ is the solution of the least squares problem, i.e., $\|\hat{r}\|_2 \leq \|r\|_2$ for all $r = b - Ax$. We can write

$$r = b - A\hat{x} + A(\hat{x} - x) = \hat{r} + A(\hat{x} - x),$$

and

$$\begin{aligned}\|r\|_2^2 &= r^T r = (\hat{r} + A(\hat{x} - x))^T(\hat{r} + A(\hat{x} - x)) \\ &= \hat{r}^T\hat{r} + \hat{r}^T A(\hat{x} - x) + (\hat{x} - x)^T A^T\hat{r} + (\hat{x} - x)^T A^T A(\hat{x} - x).\end{aligned}$$

Since $A^T\hat{r} = 0$, the two terms in the middle are equal to zero, and we get

$$\|r\|_2^2 = \hat{r}^T\hat{r} + (\hat{x} - x)^T A^T A(\hat{x} - x) = \|\hat{r}\|_2^2 + \|A(\hat{x} - x)\|_2^2 \geq \|\hat{r}\|_2^2,$$

which was to be proved. □

Example 3.13 We can now solve the example given at the beginning of the chapter. We have

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{pmatrix}, \qquad b = \begin{pmatrix} 7.97 \\ 10.2 \\ 14.2 \\ 16.0 \\ 21.2 \end{pmatrix}.$$

Using Matlab we then get

```
>> c=A'*A          % Normal equations

c =  5    15
    15    55

>> x=c\(A'*b)

x = 4.2360
    3.2260
```

■

Solving the linear least problems using the normal equations has two significant drawbacks:

1. *Forming $A^T A$ leads to loss of information.*
2. *The condition number $A^T A$ is the square of that of A*:

$$\kappa(A^T A) = (\kappa(A))^2.$$

The condition number of a rectangular matrix A is defined using the singular value decomposition of A. We will state a result on the conditioning of the least squares problem in Section 6.6.

Example 3.14 We compute the condition number of the matrix in the example using Matlab:

```
A =  1     1
     1     2
     1     3
     1     4
     1     5

cond(A) = 8.3657

cond(A'*A) = 69.9857
```

Worse example:

```
A =  1   101
     1   102
     1   103
     1   104
     1   105

cond(A) = 7.5038e+03

cond(A'*A) = 5.6307e+07
```

Note that a linear model

$$l(x) = c_0 + c_1 x,$$

with data vector $x = (101\,102\,103\,104\,105)^T$, should be replaced by the model

$$l(x) = b_0 + b_1(x - 103),$$

since in the latter case the normal equations become diagonal and much better conditioned (show that). ∎

It occurs quite often that one has a sequence of least squares problems with the same matrix,

$$\min_{x_i} \|Ax_i - b_i\|_2, \qquad i = 1, 2, \ldots, p,$$

with solutions

$$x_i = (A^T A)^{-1} A^T b_i, \qquad i = 1, 2, \ldots, p.$$

Defining $X = \begin{pmatrix} x_1 & x_2 & \ldots & x_p \end{pmatrix}$ and $X = \begin{pmatrix} b_1 & b_2 & \ldots & b_p \end{pmatrix}$ we can write this in matrix form:

$$\min_X \|AX - B\|_F, \tag{3.5}$$

with the solution

$$X = (A^T A)^{-1} A^T B.$$

This follows from the identity

$$\|AX - B\|_F^2 = \sum_{i=1}^{p} \|Ax_i - b_i\|_2^2,$$

and the fact that the p subproblems in (3.5) are independent.

Chapter 4

Orthogonality

Example 4.1 In Example 3.14 we saw that an unsuitable of choice of basis vectors in the least squares problem led to ill-conditioned normal equations. Along similar lines, define the two matrices

$$A = \begin{pmatrix} 1 & 1.05 \\ 1 & 1 \\ 1 & 0.95 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1/\sqrt{2} \\ 1 & 0 \\ 1 & -1/\sqrt{2} \end{pmatrix},$$

illustrated in Figure 4.1. It can be shown that the column vectors of the two matrices

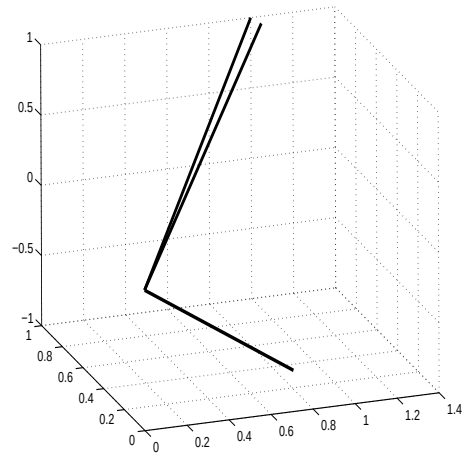


Figure 4.1. *Three vectors spanning a plane in* $\mathbb{R}^3$.

span the same plane in $\mathbb{R}^3$. From the figure it is intuitively clear that the columns of B, which are orthogonal, determine the plane much better than the columns of A, which are quite close. ∎

From several points of view, it is advantageous to use orthogonal vectors as basis vectors in a vector space. In this chapter we will list some important properties of orthogonal sets of vectors and orthogonal matrices. We assume that the vectors are in $\mathbb{R}^m$, with $m \geq n$.

4.1 Orthogonal Vectors and Matrices

We first recall that two non-zero vectors x and y are called *orthogonal* if $x^T y = 0$ (i.e. $\cos\theta(x, y) = 0$.)

Proposition 4.2. *Let q_j, $j = 1, 2, \ldots, n$ be orthogonal, i.e. $q_i^T q_j = 0, i \neq j$. Then they are linearly independent.*

Proof. Assume they are linearly dependent. Then from Proposition 2.2 there exists a q_k such that

$$q_k = \sum_{j \neq k} \alpha_j q_j.$$

Multiplying this equation by q_k^T we get

$$q_k^T q_k = \sum_{j \neq k} \alpha_j q_k^T q_j = 0,$$

since the vectors are orthogonal. This is a contradiction. ∎

Let the set of orthogonal vectors $q_j,\ j = 1, 2, \ldots, m$ in $\mathbb{R}^m$ be normalized,

$$\|q_j\|_2 = 1.$$

Then they are called *orthonormal*, and they constitute an *orthonormal basis* in $\mathbb{R}^m$.

A square matrix

$$\mathbb{R}^{m \times m} \ni Q = \begin{pmatrix} q_1 & q_2 & \cdots & q_m \end{pmatrix},$$

whose columns are orthonormal is called an *orthogonal matrix*.

Orthogonal matrices satisfy a number of important properties that we list in a sequence of propositions.

Proposition 4.3. *An orthogonal matrix Q satisfies $Q^T Q = I$.*

Proof.

$$Q^T Q = \begin{pmatrix} q_1 & q_2 & \cdots & q_m \end{pmatrix}^T \begin{pmatrix} q_1 & q_2 & \cdots & q_m \end{pmatrix} = \begin{pmatrix} q_1^T \\ q_2^T \\ \cdots \\ q_m^T \end{pmatrix} \begin{pmatrix} q_1 & q_2 & \cdots & q_m \end{pmatrix}$$

$$= \begin{pmatrix} q_1^T q_1 & q_1^T q_2 & \cdots & q_1^T q_m \\ q_2^T q_1 & q_2^T q_2 & \cdots & q_2^T q_m \\ \vdots & \vdots & & \vdots \\ q_m^T q_1 & q_m^T q_2 & \cdots & q_m^T q_m \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix},$$

due to orthonormality. □

The orthogonality of its columns implies that an orthogonal matrix has full rank, and it is trivial to find the inverse.

Proposition 4.4. *An orthogonal matrix $Q \in \mathbb{R}^{m\times m}$ has rank m, and, since $Q^TQ = I$, its inverse is equal to $Q^{-1} = Q^T$.*

Proposition 4.5. *The rows of an orthogonal matrix are orthogonal, i.e. $QQ^T = I$.*

Proof. Let x be an arbitrary vector. We shall show that $QQ^Tx = x$. Given x there is a uniquely determined vector y, such that $Qy = x$, since Q^{-1} exists. Then

$$QQ^Tx = QQ^TQy = Qy = x.$$

Since x is arbitrary, it follows that $QQ^T = I$. □

Proposition 4.6. *The product of two orthogonal matrices is orthogonal.*

Proof. Let Q and P be orthogonal, and put $X = PQ$. Then

$$X^TX = (PQ)^TPQ = Q^TP^TPQ = Q^TQ = I.$$

□

Any orthonormal basis of a subspace of $\mathbb{R}^m$ can be enlarge to an orthonormal basis of the whole space. In matrix terms:

Proposition 4.7. *Given a matrix $Q_1 \in \mathbb{R}^{m\times k}$, with orthonormal columns, there exists a matrix $Q_2 \in \mathbb{R}^{m\times(m-k)}$, such that $Q = (Q_1\, Q_2)$ is an orthogonal matrix.*

This proposition is a standard result in linear algebra. We will later demonstrate how Q can be computed.

One of the most important properties of orthogonal matrices is that they preserve the length of a vector.

Proposition 4.8. *The Euclidean length of a vector is invariant under an orthogonal transformation Q.*

Proof.

$$\|Qx\|_2^2 = (Qx)^TQx = x^TQ^TQx = x^Tx = \|x\|_2^2.$$

□

Also the the corresponding matrix norm and the Frobenius norm are *invariant under orthogonal transformations.*

Proposition 4.9. *Let $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ be orthogonal. Then for any $A \in \mathbb{R}^{m\times n}$,*

$$\| UAV \|_2 = \| A \|_2,$$
$$\| UAV \|_F = \| A \|_F.$$

Proof. The first equality is easily proved using Proposition 4.8. The second is proved using the alternative expression (2.7) for the Frobenius norm and the identity $\mathrm{tr}(BC) = \mathrm{tr}(CB)$. □

4.2 Elementary Orthogonal Matrices

We will use elementary orthogonal matrices to reduce matrices to compact form. For instance, we will transform a matrix $A \in \mathbb{R}^{m\times n}$, $m \geq n$, to triangular form.

4.2.1 Plane rotations

A 2×2 plane rotation matrix[5]

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \qquad c^2 + s^2 = 1.$$

is orthogonal (show this!). Multiplication of a vector x by G rotates the vector in a clock-wise direction by an angle θ, where $c = \cos\theta$. A plane rotation can be used to zero the second element of a vector x by choosing $c = x_1/\sqrt{x_1^2 + x_2^2}$ and $s = x_2/\sqrt{x_1^2 + x_2^2}$:

$$\frac{1}{\sqrt{x_1^2 + x_2^2}} \begin{pmatrix} x_1 & x_2 \\ -x_2 & x_1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \sqrt{x_1^2 + x_2^2} \\ 0 \end{pmatrix}.$$

By embedding a 2-D rotation in a larger unit matrix, one can manipulate vectors and matrices of arbitrary dimension.

Example 4.10 We can choose c and s in

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & 0 & s \\ 0 & 0 & 1 & 0 \\ 0 & -s & 0 & c \end{pmatrix}$$

[5] In the numerical literature plane rotations are often called Givens rotations, after Wallace Givens, who used them for eigenvalue computations around 1960. However, they had been used long before that by Jacobi, also for eigenvalue computations.

so that we zero element 4 in a vector $x \in \mathbb{R}^4$ *by a rotation in plane (2,4).* Execution of the Matlab script

```
x=[1;2;3;4];
sq=sqrt(x(2)^2+x(4)^2);
c=x(2)/sq; s=x(4)/sq;
G=[1 0 0 0; 0 c 0 s; 0 0 1 0; 0 -s 0 c];
y=G*x
```

gives the result

```
y = 1.0000
    4.4721
    3.0000
         0
```

∎

Using a sequence of plane rotations we can now transform an arbitrary vector to a multiple of a unit vector. This can be done in several ways. We demonstrate one in the following example.

Example 4.11 Given a vector $x \in \mathbb{R}^4$, we transform it to κe_1. First, by a rotation G_3 in the plane $(3,4)$ we zero the last element:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c_1 & s_1 \\ 0 & 0 & -s_1 & c_1 \end{pmatrix} \begin{pmatrix} \times \\ \times \\ \times \\ \times \end{pmatrix} = \begin{pmatrix} \times \\ \times \\ * \\ 0 \end{pmatrix}$$

Then, by a rotation G_2 in the plane $(2,3)$ we zero the element in position 3:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c_2 & s_2 & 0 \\ 0 & -s_2 & c_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \times \\ \times \\ \times \\ 0 \end{pmatrix} = \begin{pmatrix} \times \\ * \\ 0 \\ 0 \end{pmatrix}$$

Finally, the second element is annihilated by a rotation G_1:

$$\begin{pmatrix} c_3 & s_3 & 0 & 0 \\ -s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \times \\ \times \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \kappa \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

According to Proposition 4.8 the Euclidean length is preserved, and therefore we know that $\kappa = \|x\|_2$.

We summarize the transformations: We have

$$\kappa e_1 = G_1(G_2(G_3 x)) = (G_1 G_2 G_3)x.$$

Since the product of orthogonal matrices is orthogonal (Proposition 4.6) the matrix $P = G_1 G_2 G_3$ is orthogonal, and the overall result is $Px = \kappa e_1$. ∎

Plane rotations are very flexible and can be used efficiently for problems with a sparsity structure, e.g. band matrices. On the other hand, for dense matrices they require more flops than Householder transformation, see Section 4.3.

Example 4.12 In the Matlab example earlier in this section we explicitly embedded the 2×2 in a matrix of larger dimension. This is a waste of operations, since the computer execution of the code does not take into account the fact that only two rows of the matrix are changed. Instead the whole matrix multiplication is performed, which requires $2n^3$ flops in the case of matrices of dimension n. The following two Matlab functions illustrate how the rotation should be implemented to save operations (and storage).

```
function [c,s]=rot(x,y);
  % Construct a plane rotation that zeros the second
  % component in the vector [x;y]' (x and y are scalars)
  sq=sqrt(x^2 + y^2);
  c=x/sq; s=y/sq;

function [X]=approt(c,s,i,j,X);
  % Apply a plane (plane) rotation in plane (i,j)
  % to a matrix X
  X([i,j],:)=[c s; -s c]*X([i,j],:);
```

The following script reduces the vector x to a multiple of the standard basis vector e_1.

```
x=[1;2;3;4];
for i=3:-1:1
  [c,s]=rot(x(i),x(i+1));
  x=approt(c,s,i,i+1,x);
end

>> x = 5.47722557505166
       -0.00000000000000
       -0.00000000000000
       -0.00000000000000
```

After the reduction the first component of x is equal to $\| x \|_2$. ∎

4.2.2 Householder Transformations

Let $v \neq 0$ be an arbitrary vector and put

$$P = I - \frac{2}{v^T v} v v^T;$$

P is symmetric and orthogonal (verify this by a simple computation!). Such matrices are called *reflection matrices* or *Householder transformations*. Let x and y be given

vectors of the same length, $\|x\|_2 = \|y\|_2$, and ask the question "Can we determine a Householder transformation P such that $Px = y$?"

The equation $Px = y$ can be written

$$x - \frac{2v^T x}{v^T v} v = y,$$

which is of the form $\beta v = x - y$. Since v enters in P in such a way that a factor β cancels, we can choose $\beta = 1$. With $v = x - y$ we get

$$v^T v = x^T x + y^T y - 2x^T y = 2(x^T x - x^T y),$$

since $x^T x = y^T y$. Further,

$$v^T x = x^T x - y^T x = \frac{1}{2} v^T v.$$

Therefore we have

$$Px = x - \frac{2v^T x}{v^T v} v = x - v = y,$$

as we wanted. In matrix computations we often want to zero elements in a vector and we now choose $y = \kappa e_1$, where $\kappa = \pm\|x\|_2$, and $e_1^T = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}$. The vector v should be taken equal to

$$v = x - \kappa e_1.$$

In order to avoid cancellation we choose $\operatorname{sign}(\kappa) = -\operatorname{sign}(x_1)$. Now that we have computed v, we can simplify and write

$$P = I - \frac{2}{v^T v} v v^T = I - 2uu^T, \qquad u = \frac{1}{\|v\|_2} v;$$

Thus the Householder vector u has length 1. The computation of the Householder vector can be implemented in the following Matlab code.

```
function u=househ(x)
  % Compute the Householder vector u such that
  % (I - 2 u * u')x = k*e_1, where
  % |k| is equal to the euclidean norm of x
  % and e_1 is the first unit vector
  n=length(x);      % Number of components in x
  kap=norm(x); v=zeros(n,1);
  v(1)=x(1)+sign(x(1))*kap;
  v(2:n)=x(2:n);
  u=(1/norm(v))*v;
```

In most cases one should avoid to form the Householder matrix P explicitly, since it can be represented much more compactly by the vector u. Multiplication by P should be done according to $Px = x - (2u^T x)u$, where the matrix-vector

multiplication requires $4n$ flops (instead of $O(n^2)$ if P were formed explicitly). The matrix multiplication PX is done

$$PX = A - 2u(u^T X). \tag{4.1}$$

Multiplication by a Householder transformation is implemented in the following code.

```
function Y=apphouse(u,X);
  % Multiply the matrix X by a Householder matrix
  % Y = (I - 2 * u * u') * X
  Y=X-2*u*(u'*X);
```

Example 4.13 The first three elements of the vector $x = \begin{pmatrix}1 & 2 & 3 & 4\end{pmatrix}^T$ are zeroed by the following sequence of Matlab statements:

```
>>    x=[1; 2; 3; 4];
>>    u=househ(x);
>>    y=apphouse(u,x)

    y = -5.4772
              0
              0
              0
```

■

In a similar way as plane rotations can be embedded in unit matrices, in order to apply the transformation in a structured way, we can embed Householder transformations. Assume, for instance that we have transformed the first column in a matrix to a unit vector, and that we then want to zero all the elements in the second column below the main diagonal. Thus, in an example with a 5×4 matrix, we want to compute the transformation

$$P_2 A^{(1)} = P_2 \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{pmatrix} = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix} =: A^{(2)}. \tag{4.2}$$

Partition the second column of $A^{(1)}$ as follows:

$$\begin{pmatrix} a_{12}^{(1)} \\ a_{\cdot 2}^{(1)} \end{pmatrix}.$$

We know how to transform $a_{\cdot 2}^{(1)}$ to a unit vector; let $\hat{P}_2$ be a Householder transformation that does this. Then the transformation (4.2) can be implemented by embedding $\hat{P}_2$ in a unit matrix:

$$P_2 = \begin{pmatrix} 1 & 0 \\ 0 & \hat{P}_2 \end{pmatrix}. \tag{4.3}$$

It is obvious that P_2 leaves the first row of $A^{(1)}$ unchanged and computes the transformation (4.2). Also, it is easy to see that the newly created zeros in the first column are not destroyed.

In a similar way as in the case of plane rotations, one should not explicitly embed a Householder transformation in an identity matrix of larger dimension. Instead one should apply it to the rows (in the case of multiplication from the left) that are affected in the transformation).

Example 4.14 The transformation in (4.2) is done by the following script.

```
u=househ(A(2:m,2)); A(2:m,2:n)=apphouse(u,A(2:m,2:n));

>> A = -0.8992   -0.6708   -0.7788   -0.9400
       -0.0000    0.3299    0.7400    0.3891
       -0.0000    0.0000   -0.1422   -0.6159
       -0.0000   -0.0000    0.7576    0.1632
       -0.0000   -0.0000    0.3053    0.4680
```

■

4.3 Number of Floating Point Operations

We shall compare the number of flops to transform the first column of an $m \times n$ matrix A to a multiple of a unit vector κe_1 using plane and Householder transformations. Consider first plane rotations. Obviously, the computation of

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} cx + sy \\ -sx + cy \end{pmatrix}$$

requires 4 multiplications and 2 additions, i.e. 6 flops. Applying such a transformation to an $m \times n$ matrix requires $6n$ flops. In order to zero all elements but one in the first column of the matrix therefore we apply $m - 1$ rotations. Thus the overall flop count is $6(m-1)n \approx 6mn$.

If the corresponding operation is performed by a Householder transformation as in (4.1), then $4mn$ flops are needed (note that multiplication by 2 is not a flop, as it is implemented by the compiler as a shift, which is much faster than a flop; alternatively one can scale the vector u by $\sqrt{2}$).

4.4 Orthogonal Transformations in Floating Point Arithmetic

Orthogonal transformations are very stable in floating point arithmetic. For instance, it is shown in [95, pp. 152-162] that a computed Householder transformation $\hat{P}$ that approximates P, satisfies

$$\| P - \hat{P} \|_2 = O(\mu),$$

and that

$$fl(\hat{P}A) = P(A + E), \qquad \| E \|_2 = O(\mu \| A \|_2).$$

Thus the result is the exact product of a matrix product where the data matrix has been perturbed a very small amount.

Analogous results hold for plane rotations.

Chapter 5
QR Decomposition

One of the main themes of this book is decomposition of matrices to compact (e.g. triangular or diagonal) form by orthogonal transformations. We will now introduce the first such decomposition, the *QR decomposition*. We will factorize a matrix A as a product of an orthogonal matrix and a triangular matrix. This is considerably more ambitious than computing the LU decomposition, where the two factors are both only required to be triangular.

5.1 Orthogonal Transformation to Triangular Form

By a sequence of Householder transformations (or plane rotations[6]) we can transform any matrix $A \in \mathbb{R}^{m\times n}$, $m \geq n$

$$A \longrightarrow Q^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}, \qquad R \in \mathbb{R}^{n\times n},$$

where R is upper triangular, and $Q \in \mathbb{R}^{m\times m}$ is orthogonal. The procedure can be conveniently illustrated using a matrix of small dimension. Let $A \in \mathbb{R}^{5\times 4}$. In the first step we zero the elements below the main diagonal in the first column:

$$H_1 A = H_1 \begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{pmatrix} = \begin{pmatrix} + & + & + & + \\ 0 & + & + & + \\ 0 & + & + & + \\ 0 & + & + & + \\ 0 & + & + & + \end{pmatrix},$$

where + denotes elements that have changed in the transformation. The orthogonal matrix H_1 can be taken equal to a Householder transformation. In the second step we use an embedded Householder transformation as in (4.3) to zero the elements

[6]For simplicity we assume in this chapter that Householder transformations are used, unless otherwise stated.

below the main diagonal in the second column:

$$H_2 \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{pmatrix} = \begin{pmatrix} \times & \times & \times & \times \\ 0 & + & + & + \\ 0 & 0 & + & + \\ 0 & 0 & + & + \\ 0 & 0 & + & + \end{pmatrix}.$$

Again, on the right hand side + denotes elements that have been changed in the transformation, and × elements that are unchanged in present transformation.

In the third step we annihilate elements below the diagonal in the third column:

$$H_3 \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix} = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & + & + \\ 0 & 0 & 0 & + \\ 0 & 0 & 0 & + \end{pmatrix}.$$

After the fourth step we have computed the upper triangular matrix R. The sequence of transformations is summarized

$$Q^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}, \qquad Q^T = H_4 H_3 H_2 H_1.$$

Note that the matrices H_i have the following structure (we here assume that $A \in \mathbb{R}^{m \times n}$).

$$\begin{aligned} H_1 &= I - 2u_1 u_1^T, \quad u_1 \in \mathbb{R}^m, \\ H_2 &= \begin{pmatrix} 1 & 0 \\ 0 & P_2 \end{pmatrix}, \quad P_2 = I - 2u_2 u_2^T, \quad u_2 \in \mathbb{R}^{m-1}, \\ H_3 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & P_3 \end{pmatrix}, \quad P_3 = I - 2u_3 u_3^T, \quad u_3 \in \mathbb{R}^{m-2}, \end{aligned} \tag{5.1}$$

etc. Thus we embed the Householder transformation of successively smaller dimension in identity matrices, and the vectors u_i become shorter in each step. It is easy to see that the matrices H_i are also Householder transformations. For instance,

$$H_3 = I - 2u^{(3)} {u^{(3)}}^T, \quad u^{(3)} = \begin{pmatrix} 0 \\ 0 \\ u_3 \end{pmatrix}.$$

The transformation to triangular form is equivalent to a decomposition of the matrix A.

Theorem 5.1 (QR decomposition). *Any matrix $A \in \mathbb{R}^{m \times n}$, $m \geq n$, can be transformed to upper triangular form by an orthogonal matrix. The transformation*

is equivalent to a decomposition

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where $Q \in \mathbb{R}^{m\times m}$ is orthogonal and $R \in \mathbb{R}^{n\times n}$ is upper triangular. If the columns of A are linearly independent, then R is non-singular.

Proof. The constructive procedure outlined in the preceding example can easily be adapted to the general case, under the provision that if the vector to be transformed to a unit vector is a zero vector, then the orthogonal transformation is chosen equal to the identity.

The linear independence of the columns of

$$\begin{pmatrix} R \\ 0 \end{pmatrix}$$

follows from Proposition 2.4. Since R is upper triangular, the linear independence implies that its diagonal elements are non-zero (if one column had a zero on the diagonal, then it would be a linear combination of those to the left of it). Thus, the determinant of R is non-zero, which means that R is nonsingular. □

We illustrate the QR decomposition symbolically in Figure 5.1.

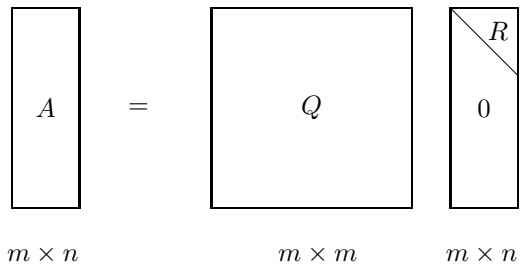


Figure 5.1. *Symbolic illustration of the QR decomposition.*

Quite often it is convenient to write the decomposition in an alternative way, where only the part of Q is kept that corresponds to a orthogonalization of the columns of A, see Figure 5.2. This can be derived by partitioning $Q = (Q_1\ Q_2)$, where $Q_1 \in \mathbb{R}^{m\times n}$, and noting that in the multiplication the block Q_2 is multiplied by zero,

$$A = (Q_1\ Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R. \tag{5.2}$$

It is seen from this equation that $\mathcal{R}(A) = \mathcal{R}(Q_1)$; thus we have now computed an *orthogonal basis* of the range space $\mathcal{R}(A)$. Furthermore, if we write out column j in (5.2),

$$a_j = Q_1 r_j = \sum_{i=1}^{j} r_{ij} q_j,$$

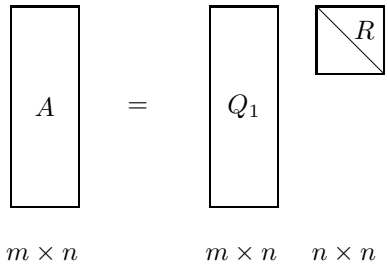


Figure 5.2. *Thin QR decomposition* $A = Q_1 R$.

we see that *column* j *in* R *holds the coordinates of* a_j *in the orthogonal basis.*

Example 5.2 We give a simple numerical illustration of the computation of a QR decomposition in Matlab:

```
A =  1     1     1
     1     2     4
     1     3     9
     1     4    16
 >> [Q,R]=qr(A)

Q =-0.5000    0.6708    0.5000    0.2236
   -0.5000    0.2236   -0.5000   -0.6708
   -0.5000   -0.2236   -0.5000    0.6708
   -0.5000   -0.6708    0.5000   -0.2236

R =-2.0000   -5.0000  -15.0000
         0   -2.2361  -11.1803
         0         0    2.0000
         0         0         0
```

The thin QR decomposition is obtained by the command `qr(A,0)`:

```
>> [Q,R]=qr(A,0)

Q =-0.5000    0.6708    0.5000
   -0.5000    0.2236   -0.5000
   -0.5000   -0.2236   -0.5000
   -0.5000   -0.6708    0.5000

R =-2.0000   -5.0000  -15.0000
         0   -2.2361  -11.1803
         0         0    2.0000
```

■

5.2 Solving the Least Squares Problem

Using the QR decomposition we can solve the least squares problem

$$\min_x \| b - Ax \|_2, \tag{5.3}$$

where $A \in \mathbb{R}^{m\times n}$, $m \geq n$, without forming the normal equations. To do this we use the fact that the Euclidean vector norm is invariant under orthogonal transformations (Proposition 4.8):

$$\|Qy\|_2 = \|y\|_2.$$

Introducing the QR decomposition of A in the residual vector we get

$$\begin{aligned}\|r\|_2^2 = \|b - Ax\|_2^2 &= \left\| b - Q \begin{pmatrix} R \\ 0 \end{pmatrix} x \right\|_2^2 \\ &= \left\| Q(Q^T b - \begin{pmatrix} R \\ 0 \end{pmatrix} x) \right\|_2^2 = \left\| Q^T b - \begin{pmatrix} R \\ 0 \end{pmatrix} x \right\|_2^2.\end{aligned}$$

Then we partition $Q = (Q_1\ Q_2)$, where $Q_1 \in \mathbb{R}^{m\times n}$, and denote

$$Q^T b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} := \begin{pmatrix} Q_1^T b \\ Q_2^T b \end{pmatrix}.$$

Now we can write

$$\|r\|_2^2 = \left\| \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} - \begin{pmatrix} Rx \\ 0 \end{pmatrix} \right\|_2^2 = \|b_1 - Rx\|_2^2 + \|b_2\|_2^2. \tag{5.4}$$

Under the assumption that the columns of A are linearly independent, we can solve

$$Rx = b_1,$$

and minimize $\|r\|_2$ by making the first term in (5.4) equal to zero. We summarize.

Theorem 5.3 (Least squares solution by QR decomposition). *Let the matrix $A \in \mathbb{R}^{m\times n}$ have full column rank, and thin QR decomposition $A = Q_1 R$. Then the least squares problem $\min_x \| Ax - b \|_2$ has the unique solution*

$$x = R^{-1} Q_1^T b.$$

Example 5.4 As an example we solve the least squares problem from the beginning of Chapter 3.6. The matrix and right hand side are

```
A =  1     1          b = 7.9700
     1     2             10.2000
     1     3             14.2000
     1     4             16.0000
     1     5             21.2000
```

with thin QR decomposition and least squares solution

```
>> [Q1,R]=qr(A,0)      % thin QR

Q1 = -0.4472   -0.6325
     -0.4472   -0.3162
     -0.4472    0.0000
     -0.4472    0.3162
     -0.4472    0.6325


R = -2.2361   -6.7082
          0    3.1623

>> x=R\(Q1'*b)

x = 4.2360
    3.2260
```

Note that the Matlab statement `x=A\b` gives the same result, using exactly the same algorithm. ∎

5.3 Computing or not Computing Q

The orthogonal matrix Q can be computed at the same time as R by applying the transformations to the identity matrix. Similarly, Q_1 in the thin QR decomposition can be computed by applying the transformations to the partial identity matrix

$$\begin{pmatrix} I \\ 0 \end{pmatrix}.$$

However, in many situations we do not need Q explicitly. Instead, it may be sufficient to apply the same sequence of Householder transformations. Due to the structure of the embedded Householder matrices exhibited in (5.1), the vectors that are used to construct the Householder transformations for reducing the matrix A to upper triangular form can be stored below the main diagonal in A, in the positions that were made equal to zero. However, an extra vector is then needed to store the elements on the diagonal of R.

In the solution of only the least squares problem (5.3) there is no need to compute Q at all. By adjoining the right hand side to the matrix, we compute

$$\begin{pmatrix} A & b \end{pmatrix} \quad \rightarrow \quad Q^T \begin{pmatrix} A & b \end{pmatrix} = \begin{pmatrix} R & Q_1^T b \\ 0 & Q_2^T b \end{pmatrix},$$

and the least squares solution can be obtained by solving $Rx = Q_1^T b$. We also see that

$$\min_x \| Ax - b \|_2 = \min_x \left\| \begin{pmatrix} Rx - Q_1^T b \\ Q_2^T b \end{pmatrix} \right\|_2 = \| Q_2^T b \|_2.$$

Thus the norm of the optimal residual is obtained as a byproduct of the triangularization procedure.

5.4 Flop count for QR Factorization

As is seen in Section 4.3, applying a Householder transformation to an $m \times n$ matrix to zero the elements of the first column below the diagonal requires approximately $4mn$ flops. In the following transformation only rows 2 to m and columns 2 to n are changed, see (5.1), and the dimension of the sub-matrix that is affected by the transformation is reduced by one in each step. Therefore the number of flops for computing R is approximately

$$4\sum_{k=0}^{n-1}(m-k)(n-k) \approx 2mn^2 - \frac{2n^3}{3}.$$

Then the matrix Q is available in factored form, as a product of Householder transformations. If we compute explicitly the full matrix Q, then in step $k+1$ we need $4(m-k)m$ flops, which leads to a total of

$$4\sum_{k=0}^{n-1}(m-k)m \approx 4mn(m-\frac{n}{2}).$$

It is possible to take advantage of structure in the accumulation of Q to reduce the flop count somewhat [36, Section 5.1.6].

5.5 Error in the Solution of the Least Squares Problem

As we stated in Section 4.4, Householder transformations and plane rotations have excellent properties with respect to rounding errors in floating point arithmetic. We here will give a theorem, the proof of which can be found in [46, Theorem 19.3].

Theorem 5.5. *Assume that $A \in \mathbb{R}^{m\times n}$, $m \geq n$ has full column rank, and that the least squares problem $\min_x \| Ax - b \|_2$ is solved using QR factorization by Householder transformations. Then the computed solution $\hat{x}$ is the exact least squares solution of*

$$\min_x \| (A + \Delta A)\hat{x} - (b + \delta b) \|_2,$$

where

$$\| \Delta A \|_F \leq c_1 mn\mu \| A \|_F + O(\mu^2), \quad \| \delta b \|_2 \leq c_2 mn \| b \| + O(\mu^2),$$

and c_1 and c_2 are small constants.

It is seen that, in the sense of backward errors, the solution is as accurate as can be hoped for (i.e. the method is backward stable). Using the perturbation

theory in Section 6.6 one can estimate the forward error in the computed solution. In Chapter 3.6 we suggested that the normal equations method for solving the least squares problem has less satisfactory properties in floating point arithmetic. It can be shown that that method is not backward stable, unless the matrix A is well-conditioned. The pros an cons of the two methods are nicely summarized in [46, p. 399]. We here give an example that, although somewhat special, demonstrates that for certain least squares problems the solution given by the method of normal equations can be much less accurate than that produced using a QR decomposition.

Example 5.6 Let $\epsilon = 10^{-7}$, and consider the matrix

$$A = \begin{pmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{pmatrix}.$$

The condition number of A is of the order 10^7. The following Matlab script

```
x=[1;1]; b=A*x;
xq=A\b;              % QR decomposition
xn=(A'*A)\(A'*b);    % Normal equations
[xq xn]
```

gave the result

```
1.00000000000000   1.01123595505618
1.00000000000000   0.98876404494382
```

which shows that the normal equations method is suffering from the fact that the condition number of the matrix A^TA is the square of that of A. ∎

5.6 Updating the Solution of a Least Squares Problem

In some applications, e.g. in signal processing, the rows of A and the corresponding elements of b are measured in real time. Let us call one row and the element of b an *observation*. Every time an observation arrives, a new least squares solution is to be computed. If we would recompute the solution from scratch, this would cost $O(mn^2)$ flops for each new observation. This is too costly in most situations, and an unnecessary heavy computation, since the least squares solution can be computed by *updating* the QR decomposition in $O(n^2)$ flops every time a new observation is available. Furthermore, the updating algorithm does not require that we have save the orthogonal matrix Q!

Assume that we have reduced the matrix and the right hand side

$$\begin{pmatrix} A & b \end{pmatrix} \quad \rightarrow \quad Q^T \begin{pmatrix} A & b \end{pmatrix} = \begin{pmatrix} R & Q_1^T b \\ 0 & Q_2^T b \end{pmatrix}, \tag{5.5}$$

from which the least squares solution is readily available. Assume that we have not saved Q. Then let a new observation be denoted $(a^T\ \beta)$, where $a \in \mathbb{R}^n$ and β is a

scalar. We then want to find the solution of the augmented least squares problem

$$\min_x \left\| \begin{pmatrix} A \\ a^T \end{pmatrix} x - \begin{pmatrix} b \\ \beta \end{pmatrix} \right\|. \tag{5.6}$$

Now, in terms of the new matrix, we can write the reduction (5.5) in the form

$$\begin{pmatrix} A & b \\ a^T & \beta \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} Q^T & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} A & b \\ a^T & \beta \end{pmatrix} = \begin{pmatrix} R & Q_1^T b \\ 0 & Q_2^T b \\ a^T & \beta \end{pmatrix}.$$

Therefore, we can find the solution of the augmented least squares problem (5.6) if we can reduce

$$\begin{pmatrix} R & Q_1^T b \\ 0 & Q_2^T b \\ a^T & \beta \end{pmatrix}.$$

to triangular form, by a sequence of orthogonal transformations. The vector $Q_2^T b$ will play no part in this reduction, and therefore we exclude it from the derivation.

We will now show how to perform the reduction to triangular form using a sequence of plane rotations. The ideas of the algorithm will be illustrated using a small example with $n = 4$. We start with

$$\begin{pmatrix} R & b_1 \\ a^T & \beta \end{pmatrix} = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}.$$

By a rotation in the $(1, n+1)$ plane, we zero the first element of the bottom row vector. The result is

$$\begin{pmatrix} + & + & + & + & + \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ 0 & + & + & + & + \end{pmatrix},$$

where + denotes elements that have been changed in the present transformation. Then the second element of the bottom vector is zeroed by a rotation in $(2, n+1)$:

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ & + & + & + & + \\ & & \times & \times & \times \\ & & & \times & \times \\ & 0 & + & + & + \end{pmatrix}.$$

Note that the zero that was introduced in the previous step is not destroyed. After

two more analogous steps the final result is achieved:

$$\begin{pmatrix} \tilde{R} & \tilde{b}_1 \\ 0 & \tilde{\beta} \end{pmatrix} = \begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \end{pmatrix}.$$

A total of n rotations are needed to compute the reduction. The least squares solution of (5.6) is now obtained by solving $\tilde{R}x = \tilde{b}_1$.

Exercises

5.1. Show that

$$4\sum_{k=0}^{n-1}(m-k)(n-k) = 2mn^2 - \frac{2n^3}{3} + O(n^2),$$

for large n.

5.2. Prove Proposition 4.7 as a corollary of Theorem 5.1.

5.3. Write a MATLAB function for reducing

$$\begin{pmatrix} R \\ x^T \end{pmatrix} \longrightarrow Q^T \begin{pmatrix} R \\ x^T \end{pmatrix} = \begin{pmatrix} \hat{R} \\ 0 \end{pmatrix},$$

where R is upper triangular, using n plane rotations.

5.4. a) Write a MATLAB function for reducing a non-symmetric square matrix to upper Hessenberg form

$$A \longrightarrow Q^T A Q = H,$$

using Householder transformations (how many?). A 6×6 Hessenberg matrix has the structure

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{pmatrix}.$$

b) The same method can be used to reduce a symmetric matrix to tridiagonal form

$$A \longrightarrow Q^T A Q = T.$$

(Check that). Modify the function by taking advantage of symmetry to save operations.

Chapter 6

Singular Value Decomposition

Even if the QR decomposition is very useful for solving least squares problems, and has excellent stability properties, it has the drawback that it treats the rows and columns of the matrix differently: It only gives a basis for the *column space*. The singular value decomposition (SVD) deals with the rows and columns in a symmetric fashion, and therefore it supplies more information about the matrix. It also "orders" the information contained in the matrix so that, loosely speaking, the "dominating part" becomes visible. This is the property that makes the SVD so useful in data mining and may other areas.

6.1 The Decomposition

Theorem 6.1 (SVD). *Any $m \times n$ matrix A, with $m \geq n$, can be factorized*

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T, \tag{6.1}$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal, and $\Sigma \in \mathbb{R}^{n \times n}$ is diagonal

$$\Sigma = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n),$$

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

Proof. The assumption $m \geq n$ is no restriction: in the other case, just apply the theorem to A^T.

We give a proof along the lines of that in [36]. Consider the maximization problem

$$\sup_{\| x \|_2 = 1} \| Ax \|_2.$$

Since we are seeking the supremum of a continuous function over a closed set, the supremum is attained for some vector x. Put $Ax = \sigma_1 y$, where $\| y \|_2 = 1$ and

$\sigma_1 = \| A \|_2$ (by definition). Using Proposition 4.7 we can construct orthogonal matrices

$$Z_1 = (y\ \bar{Z}_2) \in \mathbb{R}^{m\times m}, \quad W_1 = (x\ \bar{W}_2) \in \mathbb{R}^{n\times n}.$$

Then

$$Z_1^T A W_1 = \begin{pmatrix} \sigma_1 & y^T A \bar{W}_2 \\ 0 & \bar{Z}_2^T A \bar{W}_2 \end{pmatrix},$$

since $y^T Ax = \sigma_1$, and $Z_2^T Ax = \sigma_1 \bar{Z}_2^T y = 0$. Put

$$A_1 = Z_1^T A W_1 = \begin{pmatrix} \sigma_1 & w^T \\ 0 & B \end{pmatrix}.$$

Then

$$\frac{1}{\sigma_1^2 + w^T w} \left\| A_1 \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2^2 = \frac{1}{\sigma_1^2 + w^T w} \left\| \begin{pmatrix} \sigma_1^2 + w^T w \\ Bw \end{pmatrix} \right\|_2^2 \geq \sigma_1^2 + w^T w.$$

But $\| A_1 \|_2^2 = \| Z_1^T A W_1 \|_2^2 = \sigma_1^2$; therefore $w = 0$ must hold. Thus we have taken one step towards a diagonalization of A. The proof is now completed by induction: Assume that

$$B = Z_2 \begin{pmatrix} \Sigma_2 \\ 0 \end{pmatrix} W_2, \quad \Sigma_2 = \operatorname{diag}(\sigma_2, \ldots, \sigma_n).$$

Then we have

$$A = Z_1 \begin{pmatrix} \sigma_1 & 0 \\ 0 & B \end{pmatrix} W_1^T = Z_1 \begin{pmatrix} 1 & 0 \\ 0 & Z_2 \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & W_2^T \end{pmatrix} W_1^T.$$

Thus, the theorem follows with

$$U = Z_1 \begin{pmatrix} 1 & 0 \\ 0 & Z_2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{pmatrix}, \quad V = W_1 \begin{pmatrix} 1 & 0 \\ 0 & W_2 \end{pmatrix}.$$

□

The columns of U and V are called *singular vectors* and the diagonal elements σ_i *singular values*.

We emphasize at this point that not only is this an important theoretical result, but also there are very efficient and accurate algorithms for computing the SVD, cf. Section 6.9.

The SVD appears in other scientific areas under different names. In statistics and data analysis the singular vectors are closely related to *principal components*, see Section 6.4. and in image processing the SVD goes under the name of *Karhunen-Loewe expansion*.

We illustrate the SVD symbolically in Figure 6.1.

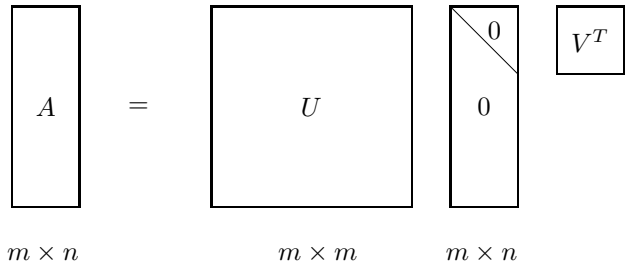


Figure 6.1. *Symbolic illustration of the singular value decomposition.*

With the partitioning $U = (U_1\, U_2)$, where $U_1 \in \mathbb{R}m \times n$, we get the "thin version" of the SVD,

$$A = U_1 \Sigma V^T.$$

We illustrate the thin SVD symbolically in Figure 6.2.

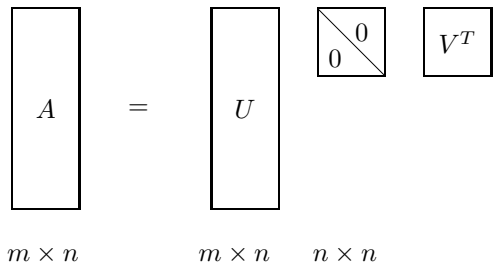


Figure 6.2. *Symbolic illustration of the thin SVD.*

If we write out the matrix equations

$$AV = U_1\Sigma, \qquad A^T U_1 = V\Sigma$$

column by column, we get the equivalent equations

$$Av_i = \sigma_i u_i, \qquad A^T u_i = \sigma_i v_i, \qquad i = 1, 2, \ldots, n.$$

The SVD can also be written as an expansion of the matrix:

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T. \tag{6.2}$$

This is usually called the *outer product form*, and it is derived by starting from the thin version:

$$A = U_1\Sigma V^T = \begin{pmatrix} u_1 & u_2 & \cdots & u_n \end{pmatrix} \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \\ \vdots \\ \sigma_n \end{pmatrix}$$

$$= \begin{pmatrix} u_1 & u_2 & \cdots & u_n \end{pmatrix} \begin{pmatrix} \sigma_1 v_1^T \\ \sigma_2 v_2^T \\ \vdots \\ \sigma_n \sigma_n \end{pmatrix} = \sum_{i=1}^{n} \sigma_i u_i v_i^T.$$

The outer product form of the SVD is illustrated in Figure 6.3.

Figure 6.3. *The SVD as a sum of rank one matrices.*

Example 6.2 We illustrate with the SVD of a matrix with full column rank:

```
A =  1     1
     1     2
     1     3
     1     4
 >> [U,S,V]=svd(A)

U = 0.2195   -0.8073    0.0236    0.5472
    0.3833   -0.3912   -0.4393   -0.7120
    0.5472    0.0249    0.8079   -0.2176
    0.7110    0.4410   -0.3921    0.3824

S = 5.7794         0
         0    0.7738
         0         0
         0         0

V = 0.3220   -0.9467
    0.9467    0.3220
```

The thin version of the SVD is

```
>> [U,S,V]=svd(A,0)

U = 0.2195   -0.8073
    0.3833   -0.3912
    0.5472    0.0249
    0.7110    0.4410

S = 5.7794         0
         0    0.7738
```

```
V = 0.3220   -0.9467
    0.9467    0.3220
```

The matrix 2-norm was defined in Section 2.4. From the proof of Theorem 6.1 we know already that $\| A \|_2 = \sigma_1$. This is such an important fact that it is worth a separate proposition.

Proposition 6.3. *The 2-norm of a matrix is given by*

$$\| A \|_2 = \sigma_1.$$

Proof. The following is an alternative proof. Let the SVD of A be $A = U\Sigma V^T$. Since the norm is invariant under orthogonal transformations, we see that

$$\| A \|_2 = \| \Sigma \|_2.$$

The result now follows, since the 2-norm of a diagonal matrix is equal to the absolute value of the largest element, cf. Exercise 3. □

6.2 Fundamental Subspaces

The SVD gives orthogonal bases of the four fundamental subspaces of a matrix.

The *range of the matrix* A is the linear subspace

$$\mathcal{R}(A) = \{y \mid y = Ax, \text{ for arbitrary } x\}.$$

Assume that A has rank r:

$$\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0.$$

Then, using the outer product form, we have

$$y = Ax = \sum_{i=1}^{r} \sigma_i u_i v_i^T x = \sum_{i=1}^{r} (\sigma_i v_i^T x) u_i = \sum_{i=1}^{r} \alpha_i u_i.$$

The *null-space of the matrix* A is the linear subspace

$$\mathcal{N}(A) = \{x \mid Ax = 0\}.$$

Since $Ax = \sum_{i=1}^{r} \sigma_i u_i v_i^T x$, we see that any vector $z = \sum_{i=r+1}^{n} \beta_i v_i$ is in the null-space:

$$Az = (\sum_{i=1}^{r} \sigma_i u_i v_i^T)(\sum_{i=r+1}^{n} \beta_i v_i) = 0.$$

After a similar demonstration for A^T we have the following theorem.

Theorem 6.4 (Fundamental subspaces).

1. *The singular vectors* $u_1, u_2, \ldots, u_r$ *are an orthonormal basis in* $\mathcal{R}(A)$ *and*
$$\text{rank}(A) = \dim(\mathcal{R}(A)) = r.$$
2. *The singular vectors* $v_{r+1}, v_{r+2}, \ldots, v_n$ *are an orthonormal basis in* $\mathcal{N}(A)$ *and*
$$\dim(\mathcal{N}(A)) = n - r.$$
3. *The singular vectors* $v_1, v_2, \ldots, v_r$ *are an orthonormal basis in* $\mathcal{R}(A^T)$.
4. *The singular vectors* $u_{r+1}, u_{r+2}, \ldots, u_m$ *are an orthonormal basis in* $\mathcal{N}(A^T)$.

Example 6.5 We create a rank deficient matrix by constructing a third column in the previous example as a linear combination of columns 1 and 2:

```
>> A(:,3)=A(:,1)+0.5*A(:,2)

A = 1.0000    1.0000    1.5000
    1.0000    2.0000    2.0000
    1.0000    3.0000    2.5000
    1.0000    4.0000    3.0000

>> [U,S,V]=svd(A,0)

U = 0.2612   -0.7948   -0.5000
    0.4032   -0.3708    0.8333
    0.5451    0.0533   -0.1667
    0.6871    0.4774   -0.1667

S = 7.3944         0         0
         0    0.9072         0
         0         0         0

V = 0.2565   -0.6998    0.6667
    0.7372    0.5877    0.3333
    0.6251   -0.4060   -0.6667
```

The third singular value is equal to zero and the matrix is rank deficient. Obviously, the third column of `V` is a basis vector in $\mathcal{N}(A)$:

```
>> A*V(:,3)

ans =
    1.0e-15 *
         0
   -0.2220
   -0.2220
         0
```

■

6.3 Matrix Approximation

Assume that A is a low rank matrix plus noise: $A = A_0 + N$, where the noise N is small compared to A_0. Then typically the singular values of A have the behaviour illustrated in Figure 6.4. In such a situation, if the noise is significantly smaller in

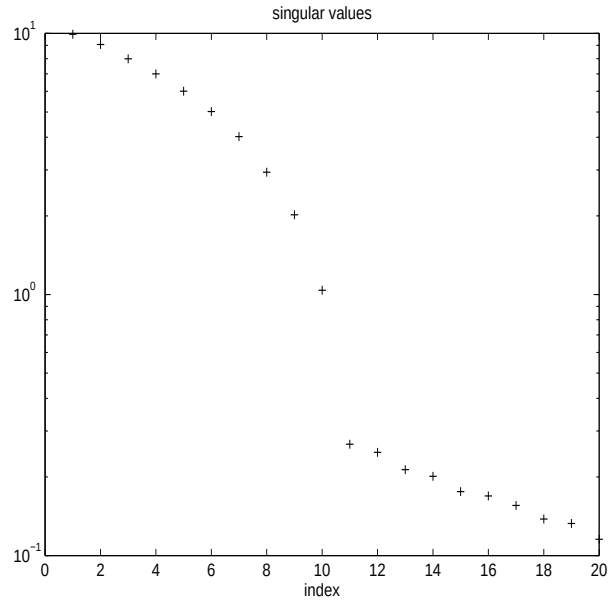


Figure 6.4. *Singular values of a matrix of rank 10 plus noise.*

magnitude, the number of large singular values is often referred to as the *numerical rank* of the matrix. If we know the correct rank of A_0, or can estimate it, e.g. by inspecting the singular values, then we can "remove the noise" by approximating A by a matrix of the correct rank. The obvious way to do this is simply to truncate the singular value expansion (6.2). Assume that the numerical rank is equal to k. Then we approximate

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T \approx \sum_{i=1}^{k} \sigma_i u_i v_i^T =: A_k.$$

The truncated SVD is very important, not only for removing noise, but also for compressing data, see Chapter 12, and for stabilizing the solution of problems that are extremely ill-conditioned.

It turns out that the truncated SVD is the solution of approximation problems, where one wants to approximate a given matrix by one of lower rank. We will consider low-rank approximation of a matrix A in two norms. First we give the theorem for the matrix 2-norm.

Theorem 6.6. *Assume that the matrix $A \in \mathbb{R}^{m\times n}$ has rank $r > k$. The matrix approximation problem*

$$\min_{\text{rank}(Z)=k} \|A - Z\|_2$$

has the solution

$$Z = A_k := U_k \Sigma_k V_k^T,$$

where $U_k = (u_1, \ldots, u_k)$, $V_k = (v_1, \ldots, v_k)$ and $\Sigma_k = \text{diag}(\sigma_1, \ldots, \sigma_k)$. The minimum is

$$\|A - A_k\|_2 = \sigma_{k+1}.$$

A proof of this theorem can be found, e.g. in [36, Section 2.5.5]. Next recall the definition of the *Frobenius matrix norm* (2.6)

$$\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2}.$$

It turns out that the approximation result is the same for this case.

Theorem 6.7. *Assume that the matrix $A \in \mathbb{R}^{m\times n}$ has rank $r > k$. The Frobenius norm matrix approximation problem*

$$\min_{\text{rank}(Z)=k} \|A - Z\|_F$$

has the solution

$$Z = A_k = U_k \Sigma_k V_k^T,$$

where $U_k = (u_1, \ldots, u_k)$, $V_k = (v_1, \ldots, v_k)$ and $\Sigma_k = \text{diag}(\sigma_1, \ldots, \sigma_k)$. The minimum is

$$\|A - A_k\|_F = \left(\sum_{i=k+1}^{p} \sigma_i^2 \right)^{1/2},$$

where $p = \min(m, n)$.

For the proof of this theorem we need a lemma.

Lemma 6.8. *Consider the mn-dimensional vector space $\mathbb{R}^{m\times n}$ with inner product*

$$\langle A, B \rangle = \text{tr}(A^T B) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} b_{ij}, \tag{6.3}$$

and norm

$$\|A\|_F = \langle A, A \rangle^{1/2}.$$

Let $A \in \mathbb{R}^{m\times n}$ with SVD $A = U\Sigma V^T$. Then the matrices

$$u_i v_j^T, \qquad i = 1, 2, \ldots, m, \qquad j = 1, 2, \ldots, n, \tag{6.4}$$

are an orthonormal basis in $\mathbb{R}^{m\times n}$.

Proof. Using the identities $\langle A, B\rangle = \mathrm{tr}(A^T B) = \mathrm{tr}(BA^T)$ we get

$$\langle u_i v_j^T, u_k v_l^T\rangle = \mathrm{tr}(v_j u_i^T u_k v_l^T) = \mathrm{tr}(v_l^T v_j\, u_i^T u_k) = (v_l^T v_j)\,(u_i^T u_k),$$

which shows that the matrices are orthonormal. Since there are mn such matrices, they constitute a basis in $\mathbb{R}^{m\times n}$. □

Proof. (**Theorem** (6.7). The proof is essentially the same as in [35].) Write the matrix $Z \in \mathbb{R}^{m\times n}$ in terms of the basis (6.4),

$$Z = \sum_{i,j} \zeta_{ij} u_i v_j^T,$$

where the coefficients are to be chosen. Since Z is assumed to have rank k, only k of the coefficients should be non-zero. For the purpose of this proof we denote the elements of Σ by σ_{ij}. Due to the orthonormality of the basis, we have

$$\|A - Z\|_F^2 = \sum_{i,j} (\sigma_{ij} - \zeta_{ij})^2 = \sum_i (\sigma_{ii} - \zeta_{ii})^2 + \sum_{i\neq j} \zeta_{ij}^2.$$

Obviously, we can choose the second term equal to zero, and to minimize the objective function, we should then choose

$$\zeta_{ii} = \sigma_{ii}, \quad i = 1, 2, \ldots, k,$$

which gives the desired result. □

The low rank approximation of a matrix is illustrated in Figure 6.5.

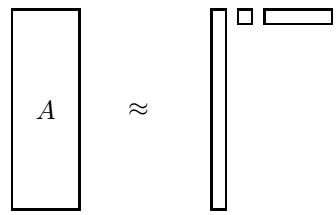


Figure 6.5. *Symbolic illustration of low rank approximation by SVD:* $A \approx A_k = U_k \Sigma_k V_k^T$.

6.4 Principal Component Analysis

The approximation properties of the SVD can be used to elucidate the equivalence between the SVD and *principal component analysis (PCA).* Assume that $X \in$

$\mathbb{R}^{m\times n}$ is a data matrix, where each column is an observation of a real-valued random vector with mean zero. The matrix is assumed to be centered, i.e. the mean of each column is equal to zero. Let the SVD of X be $X = U\Sigma V^T$. The right singular vectors v_i are called *principal components directions* of X [43, p. 62]. The vector

$$z_1 = Xv_1 = \sigma_1 u_1$$

has the largest sample variance amongst all normalized linear combinations of the columns of X:

$$\mathrm{Var}(z_1) = \mathrm{Var}(Xv_1) = \frac{\sigma_1^2}{m}.$$

Finding the vector of maximal variance is equivalent, using linear algebra terminology, to maximizing the Rayleigh quotient:

$$\sigma_1^2 = \max_{v\neq 0} \frac{v^T X^T X v}{v^T v}, \qquad v_1 = \arg\max_{v\neq 0} \frac{v^T X^T X v}{v^T v}.$$

The normalized variable u_1 is called the *normalized first principal component* of X. The second principal component is the vector of largest sample variance of the deflated data matrix $X - \sigma_1 u_1 v_1^T$, and so on. Equivalently, any subsequent principal component is defined as the vector of maximal variance subject to the constraint that it is orthogonal to the previous ones.

Example 6.9 PCA is illustrated in Figure 6.6. 500 data points from a correlated normal distribution were generated, and collected in a data matrix $X \in \mathbb{R}^{3\times 500}$. The data points and the principal components are illustrated in the top plot. We then deflated the data matrix, $X_1 := X - \sigma_1 u_1 v_1^T$; the data points corresponding to X_1 are given in the bottom plot. ∎

6.5 Solving Least Squares Problems

The least squares problem can be solved using the SVD. Assume that we have an over-determined system $Ax \sim b$, where the matrix A has full column rank. Write the SVD

$$A = (U_1\ U_2)\begin{pmatrix}\Sigma\\ 0\end{pmatrix} V^T,$$

where $U_1 \in \mathbb{R}m \times n$. Then, using the SVD and the fact that the norm is invariant under orthogonal transformations

$$\|r\|^2 = \|b - Ax\|^2 = \left\|b - U\begin{pmatrix}\Sigma\\ 0\end{pmatrix} V^T x\right\|^2 = \left\|\begin{pmatrix}b_1\\ b_2\end{pmatrix} - \begin{pmatrix}\Sigma\\ 0\end{pmatrix} y\right\|^2,$$

where $b_i = U_i^T b$ and $y = V^T x$. Thus

$$\|r\|^2 = \|b_1 - \Sigma y\|^2 + \|b_2\|^2.$$

We can now minimize $\|r\|^2$ by putting $y = \Sigma^{-1} b_1$. Then the solution is given by

$$x = Vy = V\Sigma^{-1} b_1 = V\Sigma^{-1} U_1^T b. \tag{6.5}$$

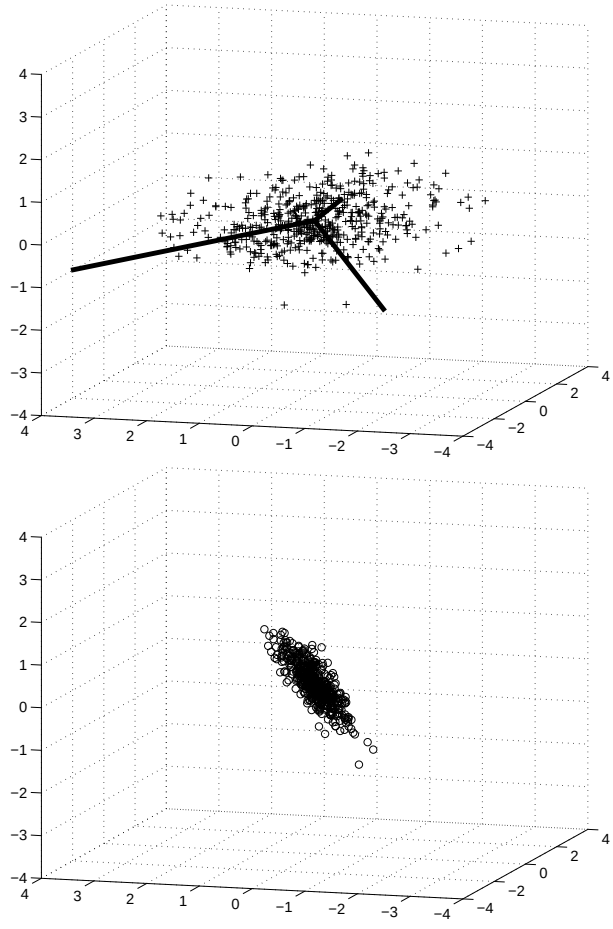


Figure 6.6. *Cluster of points in $\mathbb{R}^3$ with (scaled) principal components (top). The same data with the contributions along the first principal component deflated (bottom).*

Recall that Σ is diagonal,

$$\Sigma^{-1} = \text{diag}(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_n}),$$

so the solution can also be written

$$x = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i.$$

Note that the assumption that A has full column rank rank implies that all the singular values are non-zero: $\sigma_i > 0$, $i = 1, 2, \ldots, n$. We also see that in this case, the solution is unique.

Theorem 6.10 (Least squares solution by SVD). *Let the matrix $A \in \mathbb{R}^{m\times n}$ have full column rank, and thin SVD $A = U_1\Sigma V^T$. Then the least squares problem $\min_x \| Ax - b \|_2$ has the unique solution*

$$x = V\Sigma^{-1}U_1^T b = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i.$$

Example 6.11 As an example, we solve the least squares problem given at the beginning of Chapter 3.6. The matrix and right hand side are

```
A =  1     1              b = 7.9700
     1     2                 10.2000
     1     3                 14.2000
     1     4                 16.0000
     1     5                 21.2000

>> [U1,S,V]=svd(A,0)

U1 =0.1600   -0.7579
    0.2853   -0.4675
    0.4106   -0.1772
    0.5359    0.1131
    0.6612    0.4035

 S = 7.6912         0
         0    0.9194

 V = 0.2669   -0.9637
    0.9637    0.2669
```

It can be seen that the two column vectors in A are linearly independent since the singular values are both non-zero. The least squares problem is solved using (6.5)

```
>> x=V*(S\(U1'*b))

x = 4.2360
    3.2260
```

■

6.6 Condition Number and Perturbation Theory for the Least Squares Problem

The condition number of a rectangular matrix is defined in terms of the singular value decomposition. Let A have rank r, i.e., its singular values satisfy

$$\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0,$$

where $p = \min(m, n)$. Then the condition number is defined

$$\kappa(A) = \frac{\sigma_1}{\sigma_r}.$$

Note that in the case of a square, nonsingular matrix, this reduces to the definition (3.3).

The following perturbation theorem was proved by Wedin [94].

Theorem 6.12. *Assume that the matrix $A \in \mathbb{R}^{m\times n}$, where $m \geq n$ has full column rank, and let x be the solution of the least squares problem $\min_x \| Ax - b \|_2$. Let δA and δb be perturbations such that*

$$\eta = \frac{\| \delta A \|_2}{\sigma_n} = \kappa\epsilon_A < 1, \qquad \epsilon_A = \frac{\| \delta A \|_2}{\| A \|_2}.$$

Then the perturbed matrix $A + \delta A$ has full rank, and the perturbation of the solution δx satisfies

$$\| \delta x \|_2 \leq \frac{\kappa}{1-\eta}\left(\epsilon_A \| x \|_2 + \frac{\| \delta b \|_2}{\| A \|_2} + \epsilon_A\kappa\frac{\| r \|_2}{\| A \|_2}\right),$$

where r is the residual $r = b - Ax$.

There are at least two important observations to make here:

1. The number κ determines the condition of the least squares problem, and if $m = n$, then the residual r is equal to zero and the inequality becomes a perturbation result for a linear system of equations, cf. Theorem 3.5.

2. In the over-determined case the residual is usually not equal to zero. Then the conditioning depends on κ^2. This may be significant if the norm of the residual is large.

6.7 Rank-Deficient and Under-determined Systems

Now assume that A is rank-deficient, i.e. $\text{rank}(A) = r < \min(m, n)$. The least squares problem can still be solved, but the solution is no longer unique. In this case we can write the SVD

$$A = \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix}, \tag{6.6}$$

where

$$U_1 \in \mathbb{R}^{m\times r}, \qquad \Sigma_{\in}\mathbb{R}^{r\times r}, \qquad V_1 \in \mathbb{R}^{n\times r}, \tag{6.7}$$

and the diagonal elements of Σ_1 are all non-zero. The norm of the residual can now be written

$$\| r \|_2^2 = \| Ax - b \|_2^2 = \left\| \begin{pmatrix} U_1 & U_2 \end{pmatrix} \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix} x - b \right\|_2^2.$$

Putting

$$y = V^T x = \begin{pmatrix} V_1^T x \\ V_2^T x \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \qquad \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} U_1^T b \\ U_2^T b \end{pmatrix},$$

and using the invariance of the norm under orthogonal transformations, the residual becomes

$$\| r \|_2^2 = \left\| \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right\|_2^2 = \| \Sigma_1 y_1 - b_1 \|_2^2 + \| b_2 \|_2^2.$$

Thus, we can minimize the residual by choosing $y_1 = \Sigma_1^{-1} b_1$. In fact,

$$y = \begin{pmatrix} \Sigma_1^{-1} b_1 \\ y_2 \end{pmatrix}$$

where y_2 is arbitrary, solves the least squares problem. Therefore, *the solution of the least squares problem is not unique*, and, since the columns of V_2 span the null-space of A, it is in this null-space, where the indeterminacy is. We can write

$$\| x \|_2^2 = \| y \|_2^2 = \| y_1 \|_2^2 + \| y_2 \|_2^2,$$

and therefore we obtain the solution of *minimum norm* by choosing $y_2 = 0$.

We summarize the derivation in a theorem.

Theorem 6.13 (Minimum norm solution). *Assume that the matrix A is rank deficient with SVD* (6.6), (6.7). *Then the least squares problem* $\min_x \| Ax - b \|_2$ *does not have a unique solution. However, the problem*

$$\min_{x \in \mathcal{L}} \| x \|_2, \qquad \mathcal{L} = \{ x \mid \| Ax - b \|_2 = \min \},$$

has the unique solution

$$x = V \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T b = V_1 \Sigma_1^{-1} U_1^T b.$$

The matrix

$$A^\dagger = V \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T,$$

is called the *pseudoinverse* of A. It is defined for any non-zero matrix of arbitrary dimensions.

The SVD can also be used to solve *under-determined linear systems*, i.e. systems with more unknowns than equations. The SVD of such a matrix is give by

$$\mathbb{R}^{m \times n} \ni A = U \begin{pmatrix} \Sigma & 0 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix}, \qquad V_1 \in \mathbb{R}^{m \times m}. \tag{6.8}$$

Obviously A has full row rank if and only Σ is nonsingular.

We state a theorem concerning the solution of a linear system

$$Ax = b, \tag{6.9}$$

for the case when A has full row rank. The rank deficient case may or may not have a solution depending in the right hand side, and that case can be easily treated as in Theorem 6.13.

Theorem 6.14 (Solution of an under-determined linear system). *Let $A \in \mathbb{R}^{m\times n}$ have full row rank with SVD (6.8). Then the linear system (6.9) always has a solution, which, however, is non-unique. The problem*

$$\min_{x\in\mathcal{K}} \| x \|_2, \qquad \mathcal{K} = \{x \,|\, Ax = b\}, \tag{6.10}$$

has the unique solution

$$x = V_1^T \Sigma^{-1} U^T b. \tag{6.11}$$

Proof. Using the SVD (6.8) we can write

$$Ax = U \begin{pmatrix} \Sigma & 0 \end{pmatrix} \begin{pmatrix} V_1^T x \\ V_2^T x \end{pmatrix} =: U \begin{pmatrix} \Sigma & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = U\Sigma y_1.$$

Since Σ is nonsingular, we see that for any right hand side, (6.11) is a solution of the linear system. However, we can add an arbitrary solution component in the null-space of A, $y_2 = V_2^T x$, and we still have a solution. The minimum norm solution, i.e. the solution of (6.10), is given by (6.11). □

6.8 Generalized SVD

We have seen that the SVD can be used to solve linear equations and least squares problems. Quite often generalizations of least squares problems occur, involving two matrices. An example is the linear least squares problem with a linear equality constraint

$$\min_x \| Ax - c \|_2, \quad \text{subject to} \quad Bx = d. \tag{6.12}$$

Other problems involving two matrices will occur in Chapter 12.

Such problems can be analyzed and solved using a simultaneous diagonalization of the two matrices, the generalized SVD.

Theorem 6.15 (Generalized SVD). *Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, and $B \in \mathbb{R}^{p\times n}$. Then there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{p\times p}$, and a nonsingular $X \in \mathbb{R}^{n\times n}$, such that*

$$U^T A X = C = \operatorname{diag}(c_1, \ldots, c_n), \quad 0 \leq c_1 \leq \cdots \leq c_n \leq 1, \tag{6.13}$$

$$V^T B X = S = \operatorname{diag}(s_1, \ldots, s_q), \quad 1 \geq s_1 \geq \cdots \geq s_q \geq 0, \tag{6.14}$$

where $q = \min(p, n)$ *and*

$$C^T C + S^T S = I.$$

Proof. A proof can be found in [36, Section 8.7.3], see also [89, 66]. □

The generalized SVD is sometimes called the Quotient SVD. There is also a different generalization, called the *Product SVD*, see e.g. [23, 38].

We will now show how we can solve the least squares problem with equality constraints (6.12). For simplicity we assume that the following assumption holds,

$$\mathcal{N}(A) \cap \mathcal{N}(B) = \{0\}; \tag{6.15}$$

(otherwise the solution would not be unique). We further assume that the constraint has a full row rank matrix B, which has fewer rows than columns (otherwise, if A is square and full rank, the least squares part of (6.12) is vacuous). The GSVD can be written

$$\begin{aligned} A &= U \begin{pmatrix} C \\ 0 \end{pmatrix} X^{-1} \\ B &= V \begin{pmatrix} S & 0 \end{pmatrix} X^{-1}, \end{aligned}$$

where, due to the assumption that B has full row rank, the matrix S is nonsingular. With

$$y = X^{-1}x = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad y_1 \in \mathbb{R}^p,$$

the constraint becomes

$$d = V \begin{pmatrix} S & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = V S y_1,$$

which is satisfied by putting

$$y_1 = S^{-1} V^T d. \tag{6.16}$$

It remains to determine y_2 using the least squares problem. Using the orthogonal invariance of the norm we get the residual

$$\| r \|_2^2 = \left\| \begin{pmatrix} C \\ 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} - U^T c \right\|_2^2.$$

Partitioning

$$C = \begin{pmatrix} C_1 & 0 \\ 0 & C_2 \end{pmatrix}, \quad C_1 \in \mathbb{R}^{p \times p}, \tag{6.17}$$

we first see that, due to the assumption (6.15), the diagonal matrix C_2 must be nonsingular (cf. Exercise 3). Then, with the partitioning of U and $U^T c$,

$$U = \begin{pmatrix} U_1 & U_2 & U_3 \end{pmatrix}, \qquad U^T c = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} U_1^T c \\ U_2^T c \\ U_3^T c \end{pmatrix},$$

Decomposition	flops
LU	$2n^3/3$
QR (Householder)	$4n^3/3$
QR (plane rot.)	$8n^3/3$
SVD	$\leq 10n^3$

Table 6.1. *Flop counts for matrix decompositions.*

where $U_1 \in \mathbb{R}^{m\times p}$, and $U_2 \in \mathbb{R}^{m\times(n-p)}$, we can write the residual

$$\begin{aligned} \| r \|_2^2 &= \left\| \begin{pmatrix} C_1 & 0 \\ 0 & C_2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \right\|_2^2 \\ &= \| C_1 y_1 - c_1 \|_2^2 + \| C_2 y_2 - c_2 \|_2^2 + \| c_3 \|_2^2, \end{aligned}$$

which, since y_1 is already determined by the constraint, (6.16), is minimized by putting

$$y_2 = C_2^{-1} c_2.$$

Thus, we get an explicit formula for the solution of (6.12) in terms of the GSVD:

$$x = X \begin{pmatrix} S^{-1} & 0 \\ 0 & C_2^{-1} \end{pmatrix} \begin{pmatrix} V^T & 0 \\ 0 & U_2^T \end{pmatrix} \begin{pmatrix} d \\ c \end{pmatrix}.$$

6.9 Computing the SVD

The SVD is computed in MATLAB by the statement `[U,S,V]=svd(A)`. This statement is an implementation of algorithms from LAPACK [1] (the double precision high level driver algorithm for SVD is called `DGESVD`). In the algorithm the matrix is first reduced to bidiagonal form by a series of Householder transformation from the left and right. Then the bidiagonal matrix is iteratively reduced to diagonal form using a variant of the QR algorithm, see Chapter 16.

The SVD of a dense (full) matrix can be computed in $\mathcal{O}(mn^2)$ flops. The constant is usually $5 - 10$. For comparison we give flop counts for a few other matrix decompositions in Table 6.1.

The computation of a partial SVD of a large, sparse matrix is done in MATLAB by the statement `[U,S,V]=svds(A,k)`. This statement is based on Lanczos algorithms from ARPACK. We give a brief description of Lanczos algorithms in Chapter 16. For a more comprehensive treatment, see [4].

Exercises

6.1. Show that the left singular vectors u_i of A are eigenvectors of AA^T corresponding to the eigenvalues σ_i^2. Prove the corresponding statement for the right singular vectors.

6.2. Assume that the matrix A has non-negative elements. Prove, using Theorem 6.7, that the first singular vectors u_1 and v_1 have non-negative elements. (Hint: Assume that one component is negative. Show that it is then possible to make the objective function smaller by changing the element to be positive.)

6.3. Prove that the assumption (6.15) and the ordering convention in Theorem 6.15 imply that the matrix C_2 in (6.17) must be nonsingular.

Chapter 7

Complete Orthogonal Decompositions

In the case when the matrix is rank deficient, computing the SVD is the most reliable method for determining the rank. However, it has the drawbacks that it is comparatively expensive to compute, and it is expensive to update (when new rows and/or colums are added). Both this issues may be critical, e.g. in a real time application. Therefore, methods have been developed that approximate the SVD, so called *complete orthogonal decompositions*, which in the noisefree case and in exact arithmetic can be written

$$A = Q \begin{pmatrix} T & 0 \\ 0 & 0 \end{pmatrix} Z^T,$$

for orthogonal Q and Z and triangular $T \in \mathbb{R}^{r \times r}$ when A has rank r. Obviously, the SVD is a special case of a complete orthogonal decomposition.

In this chapter we will assume that the matrix $A \in \mathbb{R}^{m \times n}$, $m \geq n$ has exact or numerical rank r (recall the definition of numerical rank on p. 61).

7.1 QR with Column Pivoting

The first step towards obtaining a complete orthogonal decomposition is to perform column pivoting in the computation of a QR decomposition [19]. Consider the matrix before the algorithm has started: Compute the 2-norm of each column, and move the column with largest norm to the leftmost position. This is equivalent to multiplying the matrix by a permutation matrix P_1 from the right. In the first step of the reduction to triangular form, a Householder transformation is applied that annihilates elements in the first column:

$$A \longrightarrow AP_1 \longrightarrow Q_1^T A P_1 = \begin{pmatrix} r_{11} & r_1^T \\ 0 & B \end{pmatrix}.$$

Then in the next step, find the column with largest norm in B, permute the columns so that the column with largest norm is moved to position 2 in A (this only involves

columns $2-n$, of course), and reduce the first column of B:

$$Q_1^T A P_1 \longrightarrow Q_2^T Q_1^T A P_1 P_2 = \begin{pmatrix} r_{11} & r_{12} & \bar{r}_1^T \\ 0 & r_{22} & \bar{r}_2^T \\ 0 & 0 & C \end{pmatrix}.$$

It is seen that $|r_{11}| \geq |r_{22}|$.

After n steps we have computed

$$AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \qquad Q = Q_1 Q_2 \cdots Q_n, \qquad P = P_1 P_2 \cdots P_n.$$

The product of permutation matrices is itself a permutation matrix.

Proposition 7.1. *Assume that A has rank r. Then, in exact arithmetic, the QR decomposition with column pivoting is given by*

$$AP = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}, \qquad R_{11} \in \mathbb{R}^{r \times r},$$

and the diagonal elements of R_{11} are nonzero (R_{11} is nonsingular).

Proof. Obviously the diagonal elements occurring in the process are non-increasing,

$$|r_{11}| \geq |r_{22}| \geq \cdots$$

Assume that $r_{r+1,r+1} > 0$. That would imply that the rank of R in the QR decomposition is larger than r, which is a contradiction, since R and A must have the same rank. □

Example 7.2 The following Matlab script performs QR decomposition with column pivoting on a matrix that is constructed to be rank deficient.

```
[U,ru]=qr(randn(3)); [V,rv]=qr(randn(3));
D=diag([1 0.5 0]);  A=U*D*V';
[Q,R,P]=qr(A);   % QR with column pivoting

R = -0.8540    0.4311   -0.0642
          0    0.4961   -0.2910
          0         0    0.0000

>> R(3,3) = 2.7756e-17
```

■

In many cases QR decomposition with pivoting gives reasonably accurate information about the numerical rank of a matrix.

Example 7.3 We modify the matrix in the previous script by adding noise.

```
[U,ru]=qr(randn(3)); [V,rv]=qr(randn(3));
D=diag([1 0.5 0]);  A=U*D*V'+1e-4*randn(3);
[Q,R,P]=qr(A);

>> R =  0.8172   -0.4698   -0.1018
             0   -0.5758    0.1400
             0         0    0.0001
```

The smallest diagonal element is of the same order of magnitude as the smallest singular value:

```
>> svd(A) = 1.0000
            0.4999
            0.0001
```

∎

It turns out, however, that one cannot rely completely on this procedure to give correct information about possible rank deficiency. We give an example due to Kahan, see [46, Section 8.3].

Example 7.4 Let $c^2 + s^2 = 1$. For n large enough, the triangular matrix

$$T_n(c) = \operatorname{diag}(1, s, s^2, \ldots, s^{n-1}) \begin{pmatrix} 1 & -c & -c & \cdots & -c \\ & 1 & -c & \cdots & -c \\ & & \ddots & & \vdots \\ & & & 1 & -c \\ & & & & 1 \end{pmatrix}$$

is very ill-conditioned: For $n = 200$ and $c = 0.2$, we have

$$\kappa_2(T_n(c)) = \frac{\sigma_1}{\sigma_n} \approx \frac{12.7}{5.7 \cdot 10^{-18}}.$$

Thus, in IEEE double precision the matrix is singular. The columns of the triangular matrix all have length 1. Therefore, due to the fact that the elements in each row to the right of the diagonal are equal, QR decomposition with column pivoting, will not introduce any column interchanges, and the upper triangular matrix R is equal to $T_n(c)$. However, the bottom diagonal element is equal to $s^{199} \approx 0.0172$, so for this matrix QR with column pivoting does not give any information whatsoever about the ill-conditioning. ∎

7.2 Rank-Revealing URV Decompositions

7.3 Updating the URV Decomposition

Exercises

7.1. Write a Matlab function that performs QR factorization with column pivoting.

Chapter 8

Reduced Rank Least Squares Models

Consider a linear model

$$b = Ax + \eta, \qquad A \in \mathbb{R}^{m\times n},$$

where η is random noise, and A and b are given. If one chooses to determine x by minimizing the Euclidean norm of the residual $b - Ax$ then one has a linear least squares problem

$$\min_x \|Ax - b\|_2. \tag{8.1}$$

In some situations the actual solution x itself is not the primary object of interest, but rather it is an auxiliary, intermediate variable. This is the case, e.g. in certain classification methods, when the norm of the residual is the interesting quantity, see Chapter 11.

Least squares prediction is another area, where the solution x is intermediate quantity, not interesting in itself (except that it should be robust and reliable in a numerical sense). Each column of A consists of observations of *explanatory variables*, or *prediction variables*, which are used to explain the variation of a *dependent variable* b. In this context it is essential that much of the vaiation of b is explained by an approximate solution $\hat{x}$, i.e. the relative residual $\|A\hat{x} - b\|_2/\|b\|_2$ should be small. Given $\hat{x}$ and a new row vector a_{new}^T of observations of the explanatory variables, one can predict the corresponding value of the dependent variable:

$$b_{\text{predicted}} = a_{\text{new}}^T \hat{x}. \tag{8.2}$$

However, often it is not necessary, or even desirable, to find the solution that actually minimizes the residual in (8.1). For instance, in prediction, several of the explanatory variables may be (almost) linearly dependent. Therefore the matrix A is often very ill-conditioned and the least squares solution is highly influenced by measurement errors and floating point round of errors.

Example 8.1 The Matlab script

```
A=[1 0
   1 1
   1 1];
B=[A A*[1;0.5]+1e-7*randn(3,1)];
b=A*[1;1]+1e-4*randn(3,1);
x=B\b
```

creates a matrix B that is quite ill-conditioned: $\kappa_2(B) \approx 3.75 \cdot 10^7$. The script gives the result

```
x =  1.0e+03 *

   -5.0800
   -2.5395
    5.0810
```

This approximate solution explains the variation of the dependent variable very well, the residual is small,

```
resn = norm(B*x-b)/norm(b) = 1.2092e-13
```

However, due to the fact that the components of the solution are large, there will be severe cancellation (see Section 1.5) in the evaluation of (8.2), which leads to numerical errors. Furthermore, in an application it may be very difficult to interpret such a solution. On the other hand, the approximate solution

```
x = 0.3332
    0.6668
    0.6666
```

obtained using a truncated SVD (see Section 8.1 below), has residual norm $4.7 \cdot 10^{-5}$, which is reasonable in view of the perturbation of the right hand side. This vector may be much better for prediction, as the cancellation is much smaller. ∎

In order to reduce the ill-conditioning of the problem, and thus make the solution less sensitive to data, one sometimes introduces an approximate orthogonal basis of low dimension in $\mathbb{R}^n$, i.e. the space where the solution x lives. Let the basis vectors be $\begin{pmatrix} z_1 & z_2 & \dots & z_k \end{pmatrix} =: Z_k$, for some (small) value of k. Then in order to determine the coordinates of the solution in terms of the approximate basis, we make the *ansatz* $x = Z_k y$ in the least squares problem, and solve

$$\min_y \|AZ_k y - b\|_2. \tag{8.3}$$

This is a least squares problem corresponding to a *reduced rank model*. In the following two sections we describe two methods for determining such a matrix of basis vectors. The first is based on the SVD of the data matrix A. The second method is a *Krylov subspace method*, in which the right hand side influences the choice of basis.

8.1 Truncated SVD: Principal Components Regression

Assume that the data matrix has the SVD

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T,$$

where r is the rank of A (note that we allow $m \geq n$ or $m \leq n$). The minimum norm solution (see Section 6.7) of the least squares problem (8.1) is

$$x = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} v_i. \tag{8.4}$$

As the SVD "orders the variation" of the data matrix A starting with the dominating direction, we see that the terms in the sum (8.4) are also organized in this way: the first term is the solution component along the dominating direction of the data matrix, the second term is the component along the second most dominating direction, etc[7].

If we prefer to use the ordering induced by the SVD of the data matrix, then we should choose the matrix Z_k in (8.3) equal to the first k right singular vectors of A,

$$Z_k = V_k = \begin{pmatrix} v_1 & v_2 & \dots & v_k \end{pmatrix}.$$

Using the fact that

$$V^T V_k = \begin{pmatrix} I_k \\ 0 \end{pmatrix},$$

where $I_k \in \mathbb{R}^{k\times k}$, we get

$$\begin{aligned} \|AV_k y - b\|_2^2 &= \|U\Sigma V^T V_k y - b\|_2^2 = \|U\left(\Sigma \begin{pmatrix} I_k \\ o \end{pmatrix} y - U^T b\right)\|_2^2 \\ &= \left\| \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix} - \begin{pmatrix} u_1^T b \\ \vdots \\ u_k^T b \end{pmatrix} \right\|_2^2 + \sum_{i=k+1}^{r} (u_i^T b)^2. \end{aligned}$$

Thus, assuming that $k \leq n$, the least squares problem $\min_y \|AV_k y - b\|_2$ has the solution

$$y = \begin{pmatrix} u_1^T b/\sigma_1 \\ \vdots \\ u_k^T b/\sigma_k \end{pmatrix},$$

which is equivalent to taking

$$x_k := \sum_{i=1}^{k} \frac{u_i^T b}{\sigma_i} v_i,$$

[7] However, this does not mean that the terms in the sum are ordered by magnitude.

as an approximate solution of (8.1). Naturally, this is often referred to as the *truncated SVD solution*.

Often one wants to find as low a value of k such that the reduction of the residual is substantial enough. The procedure can be formulated as an algorithm, which is sometimes called *principal components regression*.

Principal components regression (truncated SVD)

---

1. Find the smallest value of k such that $\sum_{i=k+1}^{r}(u_i^T b)^2 < \text{tol}\,\|b\|_2^2$.
2. Put $x_k := \sum_{i=1}^{k} \frac{u_i^T b}{\sigma_i} v_i$

The parameter tol is a predefined tolerance.

Example 8.2 We use the matrix from Example 1.1 in Chapter 1. Let

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Recall that each column corresponds to a document (here a sentence). We want to see how well two query vectors q_1 and q_2 can be represented in terms of the first terms of the singular value expansion (8.4) of the solution, i.e. we will solve the least squares problems

$$\min_y \|AV_k y - q_i\|_2, \qquad i = 1, 2,$$

for different values of k. The two vectors are

$$q_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \qquad q_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

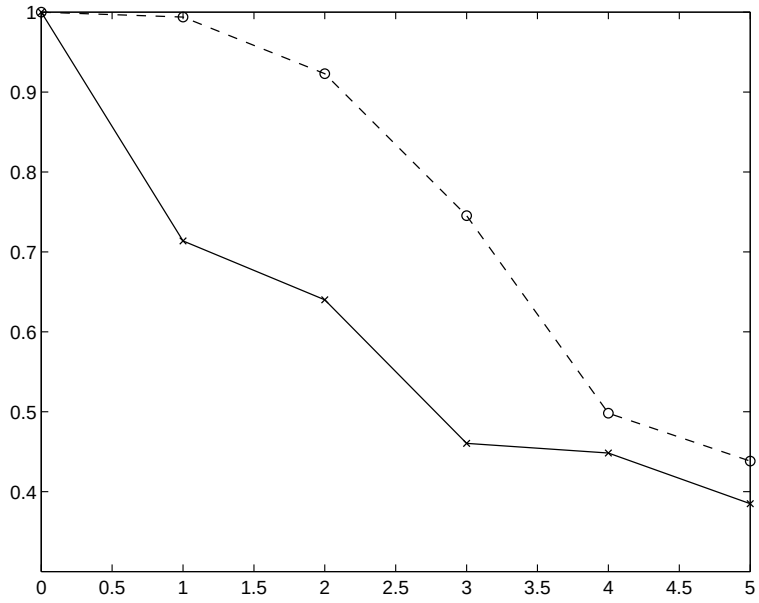


Figure 8.1. *The relative norm of the residuals for the query vectors q_1 and q_2.*

cooresponding to the words *rank, page, web*, and *England, FIFA*, respectively. In Figure 8.1 we plot the relative norm of residuals, $\|Ax_k - b\|_2/\|b\|_2$, for the two vectors as functions of k. From Example 1.1 we see that the main contents of the documents are related to the ranking web pages using the Google matrix, and this is reflected in the dominant singular vectors[8]. Therefore it is to be expected that the residual for q_1 decays faster than that of q_2 as a function of k.

The coordinates of q_1 and q_2 in terms of the left singular vectors are

```
U'*[q1 q2] = 1.2132    0.1574
            -0.5474    0.5215
             0.7698    0.7698
            -0.1817    0.7839
             0.3981   -0.3352
            -0.2445   -0.2044
            -0.1299    0.0307
             0.3741   -0.1408
             0.2966   -0.5490
             0.3741   -0.1408
```

Since q_2 has a small component in terms of u_1, there is only a mariginal reduction of the residual for $k = 1$.

If we want to reduce the relative residual to under 0.7, then we should choose $k = 2$ for q_1 and $k = 4$ for q_2. ∎

[8]See also Example 12.8 in Chapter 12.

8.2 A Krylov Subspace Method

When we use the truncated SVD (principal components regression) for a reduced rank model, then the right hand side does not influence the choice of basis vectors z_i at all. The effect of this is apparent in Example 8.2 above, where the rate of decay of the residual is considerably slower for the vector q_2 than for q_1.

In many situations one would like to have a fast decay of the residual as a function of the dimension of the matrix of basis vectors for all or most right hand sides. Then is is necessary to let the right hand side influence the choice of basis vectors. This is done in an algorithm that is called *Lanczos (Golub-Kahan) bidiagonalization* in numerical linear algebra. An essentially equivalent algorithm is known in chemometrics and other areas as *partial least squares*. It is an algorithm out of a large class of methods for the solution of large and sparse linear systems that go under the name of *Krylov subspace methods*, see e.g. [36, Chapters 9-10], [70].

Krylov subspace methods are recursive, but in our derivation we will start with the reduction of a matrix to bidiagonal form using Householder transformations. The presentation in this section is largely influenced by [14].

8.2.1 Householder Bidiagonalization

The first step in computing the SVD of a dense matrix C is to reduce it to upper bidiagonal form by Householder transformations from the left and right. The result is

$$C = P \begin{pmatrix} \hat{B} \\ 0 \end{pmatrix} W^T, \tag{8.5}$$

where P and W are orthogonal and $\hat{B}$ is upper bidiagonal. The decomposition in itself is useful also for other purposes. For instance, it is often used for the approximate solution of least squares problems, both dense and sparse.

We illustrate the Householder bidiagonalization procedure with a small example, where $C \in \mathbb{R}^{6\times 5}$. First all subdiagonal elements in the first column are zeroed by a transformation P_1^T from the left:

$$P_1^T C = P_1^T \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix} = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}.$$

Then, by a different Householder transformation W_1 from the right we zero elements in the first row, from position 3 to n. To achieve this we choose

$$\mathbb{R}^{5\times 5} \ni W_1 = \begin{pmatrix} 1 & 0 \\ 0 & Z_1 \end{pmatrix},$$

where Z_1 is a Householder transformation. Since this transformation does not change the elements in the first column, the zeros that we just introduced in the

first column remain. The result of the first step is

$$P_1^T C W_1 = \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix} = \begin{pmatrix} \times & * & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix} =: C_1.$$

We now continue in an analogous way, and zero all elements below the diagonal in the second column by a transformation from the left. The matrix P_2 is constructed so that it does not change the elements in the first row of C_1, i.e. P_2 has the structure

$$\mathbb{R}^{6\times 6} \ni P_2 = \begin{pmatrix} 1 & 0 \\ 0 & \tilde{P}_2 \end{pmatrix}$$

where $\tilde{P}_2 \in \mathbb{R}^{5\times 5}$ is a Householder transformation. We get

$$P_2^T C_1 = \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix}$$

Then, by a transformation from the right of the structure

$$W_2 = \begin{pmatrix} I_2 & 0 \\ 0 & Z_2 \end{pmatrix}, \qquad I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

we annihilate elements in the second row without destroying the newly introduced zeros:

$$P_1^T C_1 W_2 = \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & * & 0 & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{pmatrix} =: C_2.$$

We continue in an analogous manner and finally obtain

$$P^T C W = \begin{pmatrix} \times & \times & & & \\ & \times & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \\ & & & & \end{pmatrix} = \begin{pmatrix} \hat{B} \\ 0 \end{pmatrix}, \tag{8.6}$$

where, in the general case,

$$P = P_1 P_2 \cdots P_n \in \mathbb{R}^{m\times m}, \qquad W = W_1 W_2 \cdots W_{n-2} \in \mathbb{R}^{n\times n},$$

are products of Householder transformations, and

$$\hat{B} = \begin{pmatrix} \beta_1 & \alpha_1 & & & \\ & \beta_2 & \alpha_2 & & \\ & & \ddots & \ddots & \\ & & & \beta_{n-1} & \alpha_{n-1} \\ & & & & \beta_n \end{pmatrix}$$

is bidiagonal.

Due to the way the orthogonal matrices were constructed, they have a particular structure that will be useful in the rest of this chapter.

Proposition 8.3. *Denote the columns of the matrices P and W in the bidiagonal decomposition* (8.6) *by p_i, $i = 1, 2, \ldots, m$, and w_i, $i = 1, 2, \ldots, n$, respectively. Then*

$$p_1 = \beta_1 c_1, \qquad W = \begin{pmatrix} 1 & 0 \\ 0 & Z \end{pmatrix},$$

where Z is orthogonal, and c_1 is the first column of C.

Proof. The first relation follows immediately from $P^T c_1 = \beta_1 e_1$. The second follows form the fact that all W_i have the structure

$$W_i = \begin{pmatrix} 1 & 0 \\ 0 & Z_i \end{pmatrix},$$

for some orthogonal matrices Z_i. □

The reduction to bidiagonal is the first part of the algorithm for computing the SVD of a dense matrix, see Section 16.5. It requires $4mn^2 - 4n^3/3$ flops. If $m >> n$ then it is more efficient to first reduce A to upper triangular form and then bidiagonalize the R factor.

Assume now that we want to solve the least squares problem $\min_x \|b - Ax\|_2$. It turns out that if we choose $C = \begin{pmatrix} b & A \end{pmatrix}$ in the bidiagonalization procedure, then we get a equivalent bidiagonal least squares problem. First we see that using (8.6) and Proposition 8.3,

$$P^T C W = P^T \begin{pmatrix} b & A \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & Z \end{pmatrix} = \begin{pmatrix} P^T b & P^T A Z \end{pmatrix} = \begin{pmatrix} \beta_1 e_1 & B \\ 0 & 0 \end{pmatrix}, \tag{8.7}$$

where $\mathbb{R}^n \ni e_1 = \begin{pmatrix} 1 & 0 & \ldots & 0 \end{pmatrix}^T$, and

$$B = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} \\ & & & \beta_n \end{pmatrix} \in \mathbb{R}^{n \times (n-1)}.$$

Then, defining $y = Z^T x$ we can write the norm of the residual,

$$\begin{aligned}\|b - Ax\|_2 &= \left\| \begin{pmatrix} b & A \end{pmatrix} \begin{pmatrix} 1 \\ -x \end{pmatrix} \right\|_2 = \left\| P^T \begin{pmatrix} b & A \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & Z \end{pmatrix} \begin{pmatrix} 1 \\ -y \end{pmatrix} \right\|_2 \\ &= \| \begin{pmatrix} P^T b & P^T A Z \end{pmatrix} \|_2 = \|\beta_1 e_1 - By\|_2. \end{aligned} \tag{8.8}$$

The bidiagonal least squares problem $\min_y \|\beta_1 e_1 - By\|_2$ can be solved in $O(n)$ flops, if we reduce B to upper bidiagonal form using a sequence of plane rotations. We leave this as an exercise.

8.2.2 Lanczos Bidiagonalization

We will now give an alternative description of the bidiagonalization that allows us to compute a decomposition (8.6) in a recursive procedure. Using (8.7) we have

$$\begin{aligned} A^T P &= A^T \begin{pmatrix} p_1 & p_2 & \dots & p_n \end{pmatrix} \\ &= ZB^T = \begin{pmatrix} z_1 & z_2 & \dots & z_{n-1} \end{pmatrix} \begin{pmatrix} \alpha_1 & \beta_2 & & & & & & \\ & \alpha_2 & \beta_3 & & & & & \\ & & \ddots & \ddots & & & & \\ & & & & \beta_i & & & \\ & & & & \alpha_i & & & \\ & & & & & \ddots & \ddots & \\ & & & & & & \alpha_{n-1} & \beta_n \end{pmatrix}. \end{aligned}$$

Equating column i ($i \geq 2$) on both sides we get

$$A^T p_i = \beta_i z_{i-1} + \alpha_i z_i,$$

which can be written

$$\alpha_i z_i = A^T p_i - \beta_i z_{i-1}. \tag{8.9}$$

Similarly, by equating column i in

$$\begin{aligned} AZ &= A \begin{pmatrix} z_1 & z_2 & \dots & z_{n-1} \end{pmatrix} \\ &= PB = \begin{pmatrix} p_1 & p_2 & \dots & p_n \end{pmatrix} \begin{pmatrix} \alpha_1 & & & & & & & \\ \beta_2 & \alpha_2 & & & & & & \\ & \ddots & \ddots & & & & & \\ & & & \alpha_i & & & & \\ & & & \beta_{i+1} & & & & \\ & & & & \ddots & \ddots & & \\ & & & & & \beta_{n-1} & \alpha_{n-1} & \\ & & & & & & \beta_n & \end{pmatrix}, \end{aligned}$$

we get

$$Az_i = \alpha_i p_i + \beta_{i+1} p_{i+1},$$

which can be written

$$\beta_{i+1}p_{i+1} = Az_i - \alpha_i p_i. \tag{8.10}$$

Now, by compiling the starting equation $\beta_1 p_1 = b$ from Proposition 8.3, equations (8.9) and (8.10), we have derived a recursion:

Lanczos Bidiagonalization

1. $\beta_1 p_1 = b$, $z_0 = 0$
2. **for** $i = 1 : n-1$

   $\alpha_i z_i = A^T p_i - \beta_i z_{i-1}$
   $\beta_{i+1} p_{i+1} = Az_i - \alpha_i p_i$
3. **end**

The coefficients α_{i-1} and β_i are determined so that $\|p_i\| = \|z_i\| = 1$.

The recursion breaks down if any α_i or β_i becomes equal to zero. It can be shown, see e.g. [14, Section 7.2], that in the solution of least squares problems these cases are harmless.

The recursive bidiagonalization procedure gives, *in exact arithmetic*, the same result as the Householder bidiagonalization of $\begin{pmatrix} b & A \end{pmatrix}$, and thus the generated vectors $(p_i)_{i=1}^n$ and $(z_i)_{i=1}^n$ satisfy $p_i^T p_j = 0$ and $z_i^T z_j = 0$ if $i \neq j$. However, in floating point arithmetic, the vectors lose orthogonality as the recursion proceeds, see Section 8.2.7.

8.2.3 Approximate Solution of a Least Squares Problem

Define the matrices $P_k = \begin{pmatrix} p_1 & p_2 & \dots & p_k \end{pmatrix}$, $Z_k = \begin{pmatrix} z_1 & z_2 & \dots & z_k \end{pmatrix}$ and

$$B_k = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_{k-1} & \alpha_{k-1} \\ & & & \beta_k \end{pmatrix} \in \mathbb{R}^{k\times(k-1)}$$

In the same way as we could write the relations $AZ = PB$ and $A^T P = ZB^T$ as a recursion, we can now write the first k steps of the recursion as a matrix equation

$$AZ_k = P_{k+1}B_{k+1}. \tag{8.11}$$

Consider the least squares problem $\min_x \|Ax - b\|_2$. Note that the column vectors z_i are orthogonal vectors in $\mathbb{R}^n$, where the solution x lives. Now assume that we want to *find the best approximate solution in the subspace spanned by the vectors* $z_1, z_2, \dots, z_k$. That is equivalent to solving the least squares problem

$$\min_y \|AZ_k y - b\|_2,$$

where $y \in \mathbb{R}^k$. From (8.11) we see that this is equivalent to

$$\min_y \|P_{k+1}B_{k+1}y - b\|_2,$$

which, using the orthogonality of $P = \begin{pmatrix} P_{k+1} & P_\perp \end{pmatrix}$, we can rewrite

$$\begin{aligned}\|P_{k+1}B_{k+1}y - b\|_2 &= \|P^T(P_{k+1}B_{k+1}y - b)\|_2 \\ &= \left\| \begin{pmatrix} P_{k+1}^T \\ P_\perp^T \end{pmatrix} (P_{k+1}B_{k+1}y - b) \right\|_2 = \left\| \begin{pmatrix} B_{k+1}y \\ 0 \end{pmatrix} - \begin{pmatrix} \beta_1 e_1 \\ 0 \end{pmatrix} \right\|_2,\end{aligned}$$

since $b = \beta_1 p_1$. It follows that

$$\min_y \|AZ_k y - b\|_2 = \min_y \|B_{k+1}y - \beta_1 e_1\|_2, \tag{8.12}$$

which, due to the bidiagonal structure, we can solve in $O(n)$ flops.

Example 8.4 We compute approximate solutions using bidiagonalization to the same least squares problems as in Example 8.2. The relative norm of the residual,

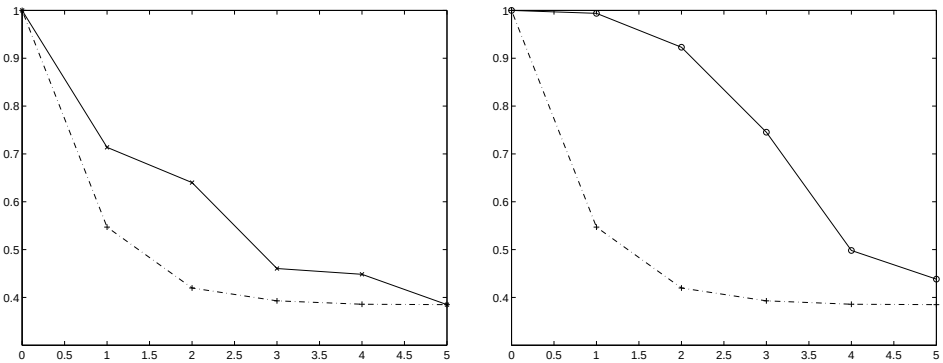


Figure 8.2. *The relative norm of the residuals for the query vectors q_1 (left) and q_2 (right) as a function of subspace dimension k. The residual curves for the truncated SVD solutions are solid and for bidiagonalization are dashed-dotted.*

$\|AZ_k y - b\|_2/\|b\|_2$ is plotted as a function of k in Figure 8.2 for the truncated SVD solution and the solution of (8.12). It is seen that in both cases the bidiagonalization-based method give a faster decay of the residual than the truncated SVD solutions. ∎

8.2.4 Matrix Approximation

The bidiagonalization procedure also gives a low rank approximation of the matrix A.

Proposition 8.5. *Given the matrix $A \in \mathbb{R}^{m\times n}$ and the matrix $P_k \in \mathbb{R}^{m\times k}$ with orthonormal columns. Then the least squares problem*

$$\min_D \|A - P_k D\|_F$$

has the solution

$$D = \widetilde{B}_k Z_k^T, \qquad \widetilde{B}_k = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \end{pmatrix} \in \mathbb{R}^{k\times k}.$$

Proof. Since the columns of P_k are orthonormal, the least squares problem has the solution

$$D = P_k^T A.$$

The first k columns of the relation $A^T P = ZB$ are

$$\begin{aligned} A^T P_k &= \begin{pmatrix} A^T p_1 & \dots & A^T p_k \end{pmatrix} \\ &= \begin{pmatrix} z_1 & \dots & z_k \end{pmatrix} \begin{pmatrix} \alpha_1 & \beta_2 & & \\ & \ddots & \ddots & \\ & & \alpha_{k-1} & \beta_k \\ & & & \alpha_k \end{pmatrix} = Z_k \widetilde{B}_k^T. \end{aligned}$$

Thus $D = P_k^T A = \widetilde{B}_k Z_k^T$. ∎

The low-rank approximation of a matrix is illustrated in Figure 8.3.

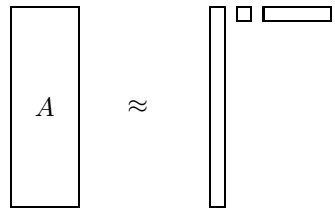


Figure 8.3. *Symbolic illustration of low rank approximation by the bidiagonal factorization: $A \approx P_k \widetilde{B}_k Z_k^T$.*

Obviously we have a rank-k approximation of the matrix A that is analogous to the low-rank approximation by SVD. We can also write it as a sum of rank-1 terms,

$$A \approx P_k \widetilde{B}_k Z_k^T = \sum_{i=1}^{k} p_i d_i^T,$$

where the d_i's are the column vectors of $D = Z_k \widetilde{B}_k^T$.

8.2.5 Krylov Subspaces

Obviously we create to sets of basis vectors the p_i's and the z_i's. It remains to demonstrate what subspaces they span. From the Lanczos recursion we see that z_1 is a multiple of $A^T b$ and that p_2 is a linear combination of b and $AA^T b$. By an easy induction proof one can show that

$$p_k \in \text{span}\{b,\, AA^T b,\, (AA^T)^2 b, \ldots, (AA^T)^{k-1} b\},$$
$$z_k \in \text{span}\{A^T b,\, (A^T A)A^T b,\, \ldots, (A^T A)^{k-1} A^T b\},$$

for $k = 1, 2, \ldots$. Denote

$$\mathcal{K}_k(C, b) = \text{span}\{b,\, Cb,\, C^2,\, \ldots, C^{k-1} b\}.$$

This a called a *Krylov subspace*.

The following result follows immediately.

Proposition 8.6. *The columns of P_k are an orthonormal basis of $\mathcal{K}_k(AA^T, b)$, and the columns of Z_k are an orthonormal basis of $\mathcal{K}_k(A^T A, A^T b)$.*

8.2.6 Partial Least Squares

Partial least squares (PLS) [97, 99] is a recursive algorithm for computing approximate least squares solutions that is often used in chemometrics. There are different variants of the algorithm, the perhaps most common is the so called NIPALS formulation.

The NIPALS PLS algorithm

1. $A_0 = A$
2. **for i=1,2,...,k**
   (a) $w_i = \frac{1}{\|A_{i-1}^T b\|} A_{i-1}^T b$
   (b) $\tilde{u}_i = \frac{1}{\|A_{i-1} w_i\|} A_{i-1} w_i$
   (c) $\tilde{v}_i = A_{i-1}^T \tilde{u}_i$
   (d) $A_i = A_{i-1} - \tilde{u}_i \tilde{v}_i^T$

This algorithm differs from Lanczos bidiagonalization in a few significant ways, the most important being that the data matrix is deflated once a new pair of vectors $(\tilde{u}_i, \tilde{v}_i)$ has been computed. However, it turns out [98, 28] that PLS algorithm is mathematically equivalent related to a variant of Lanczos bidiagonalization that is started not by choosing p_1 but instead $\alpha_1 z_1 = A^T b$. Thus the vectors $(w_i)_{i=1}^k$ form an orthonormal basis in $\mathcal{K}_k(A^T A, A^T b)$ and $(\tilde{u}_i)_{i=1}^k$ form an orthonormal basis in $\mathcal{K}_k(AA^T, AA^T b)$.

8.2.7 Computing the Bidiagonalization

The recursive versions of the bidiagonalization suffers from the weakness that the generated vectors loose orthogonality. This can be remedied by *reorthogonalizing* the vectors, using a Gram-Schmidt process. Householder bidiagonalization, on the other hand, generates vectors that are as orthogonal as can be expected in floating point arithmetic, cf. Section 4.4. Therefore, for dense matrices A of moderate to large dimension one should use this variant.

For large and sparse or otherwise structured matrices it is often necessary to use the recursive variants. Note that for such problems the PLS algorithm has the significant disadvantage that it deflates the matrix, and thus destroys the structure.

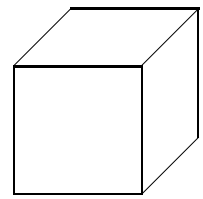
Exercises

8.1. Show that the bidiagonal least squares problem defined in (8.8) can be solved in $O(n)$ flops using plane rotations.

Chapter 9
Tensor Decomposition

9.1 Introduction

So far in this book we have considered linear algebra, where the main objects are vectors and matrices. These can be thought of as one-dimensional and two-dimensional arrays of data, respectively. In many applications it is common that data are organized in more than two categories. The corresponding mathematical objects are usually referred to as *tensors*, and the area of mathematics dealing with tensors is *multi-linear algebra*. Here, for simplicity, we will restrict ourselves to tensors $\mathcal{A} = (a_{ijk}) \in \mathbb{R}^{l\times m\times n}$ that are arrays of data with three subscripts; such a tensor can be illustrated symbolically as

$\mathcal{A} =$ 

Example 9.1 In the classification of handwritten digits, the *training set* consists of a collection of images, manually classified into 10 classes. Each such class is a collection of digits of one kind, which can be considered as a tensor, see Figure 9.1. If each digit is represented as a 16×16 matrix of numbers representing grey-scale, then a collection of n digits can be collected in a tensor $\mathcal{A} \in \mathbb{R}^{16\times 16\times n}$. ∎

We will use the terminology of [57] and refer to a tensor $\mathcal{A} \in \mathbb{R}^{l\times m\times n}$ as a 3-mode array[9], i.e. the different "dimensions" of the array are called *modes*. In this terminology, a matrix is a two-mode array. In this chapter we will present a generalization of the matrix SVD to 3-mode tensors, and then, in Chapter 15,

[9] In some literature the terminology 3-way, and, in the general case, n-way is used.

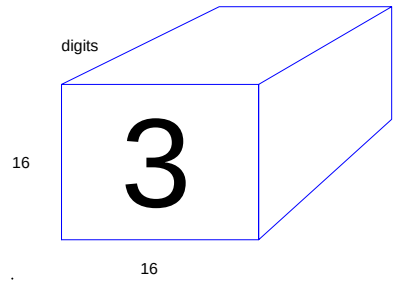


Figure 9.1. *The image of one digit is a* 16×16 *matrix, a collection of digits is a tensor.*

describe how it can be used for face recognition. The further generalization to n-mode tensors is easy, and can be found, e.g. in [57]. In fact, the face recognition application requires 5-mode arrays.

The use of tensors in data analysis applications was pioneered by researchers in psychometrics and chemometrics in the 1960's, see e.g. [78].

9.2 Basic Tensor Concepts

First define the *scalar product* (or *inner product*) of two tensors

$$\langle A, B \rangle = \sum_{i,j,k} a_{ijk} b_{ijk}. \tag{9.1}$$

The corresponding norm is

$$\| \mathcal{A} \|_F = \langle A, A \rangle^{1/2} = (\sum_{i,j,k} a_{ijk}^2)^{1/2}. \tag{9.2}$$

If we specialize the definition to matrices (2-mode tensors), we see that this is equivalent to the matrix Frobenius norm, see Section 2.4.

Next we define the *i-mode multiplication* of a tensor by a matrix. The 1-mode product of a tensor $\mathcal{A} \in \mathbb{R}^{l \times m \times n}$ by a matrix $U \in \mathbb{R}^{l_0 \times l}$, denoted by $\mathcal{A} \times_1 U$, is an $l_0 \times m \times n$ tensor in which the entries are given by

$$(\mathcal{A} \times_1 U)(j, i_2, i_3) = \sum_{k=1}^{l} u_{j,k}\, a_{k,i_2,i_3}. \tag{9.3}$$

For comparison, consider the matrix multiplication

$$A \times_1 U = UA, \quad (UA)(i,j) = \sum_{k=1}^{l} u_{i,k}\, a_{k,j} \tag{9.4}$$

We recall from Section 2.2 that matrix multiplication is equivalent to multiplying each column in A by then matrix U. Comparing (9.3) and (9.4) we see that the corresponding is true for tensor-matrix multiplication: In the 1-mode product all column vectors in the 3-mode array are multiplied by the matrix U.

Similarly, 2-mode multiplication of a tensor by a matrix V

$$(\mathcal{A} \times_2 V)(i_1, j, i_3) = \sum_{k=1}^{l} v_{j,k}\, a_{i_1,k,i_3},$$

means that all row vectors of the tensor are multiplied by V. Note that 2-mode multiplication of a matrix by V is equivalent to matrix multiplication by V^T from the right,

$$A \times_2 V = AV^T.$$

3-mode multiplication is analogous.

It is sometimes convenient to *unfold* a tensor into a matrix. The unfolding of a tensor $\mathcal{A}$ along the three modes is defined (using (semi-)Matlab notation; for a general definition[10], see [57]).

$$\begin{aligned}
\mathbb{R}^{l\times mn} \ni \text{unfold}_1(\mathcal{A}) := A_{(1)} &:= \begin{pmatrix} \mathcal{A}(:,1,:) & \mathcal{A}(:,2,:) & \cdots & \mathcal{A}(:,m,:) \end{pmatrix}, \\
\mathbb{R}^{m\times ln} \ni \text{unfold}_2(\mathcal{A}) := A_{(2)} &:= \begin{pmatrix} \mathcal{A}(:,:,1)^T & \mathcal{A}(:,:,2)^T & \cdots & \mathcal{A}(:,:,n)^T \end{pmatrix}, \\
\mathbb{R}^{n\times lm} \ni \text{unfold}_3(\mathcal{A}) := A_{(3)} &:= \begin{pmatrix} \mathcal{A}(1,:,:)^T & \mathcal{A}(2,:,:)^T & \cdots & \mathcal{A}(l,:,:)^T \end{pmatrix}.
\end{aligned}$$

It is seen that the unfolding along mode i makes that mode the first dimension in the matrix $A_{(i)}$, and the other modes are handled cyclically. For instance, row i of $A_{(j)}$ contains all the elements of $\mathcal{A}$, which have the j'th index equal to i.

Another way of putting it:

1. the column vectors of $\mathcal{A}$ are column vectors of $A_{(1)}$,
2. the row vectors of $\mathcal{A}$ are column vectors of $A_{(2)}$,
3. the 3-mode vectors of $\mathcal{A}$ are column vectors of $A_{(3)}$,

The 1-unfolding of $\mathcal{A}$ is equivalent to dividing the tensor up in *slices* $\mathcal{A}(:,i,:)$ (which are matrices) and arranging the slices in a long matrix $A_{(1)}$.

Example 9.2 Let $\mathcal{B} \in \mathbb{R}^{3\times3\times3}$ be a tensor, defined in MATLAB as

```
B(:,:,1) =          B(:,:,2) =          B(:,:,3) =
     1     2     3        11    12    13        21    22    23
     4     5     6        14    15    16        24    25    26
     7     8     9        17    18    19        27    28    29
```

Then unfolding along the third mode gives

[10] For the matrix case, $\text{unfold}_1(A) = A$, and $\text{unfold}_2(A) = A^T$.

```
>> B3=unfold(B,3)

b3=     1     2     3     4     5     6     7     8     9
       11    12    13    14    15    16    17    18    19
       21    22    23    24    25    26    27    28    29
```

■

The inverse of the unfolding operation is written

$$\text{fold}_i(\text{unfold}_i(\mathcal{A})) = \mathcal{A}.$$

In order for the folding operation to be well-defined, information about the target tensor must be supplied. In our somewhat informal presentation we suppress this.

Using the unfolding-folding operations, we can now formulate a matrix multiplication equivalent of i-mode tensor multiplication:

$$\mathcal{A} \times_i U = \text{fold}_i(U \ \text{unfold}_i(\mathcal{A})) = \text{fold}_i(UA_{(i)}). \tag{9.5}$$

It follows immediately from the definition that i-mode and j-mode multiplication commute if $i \neq j$:

$$(\mathcal{A} \times_i F) \times_j G = (\mathcal{A} \times_j G) \times_i F = \mathcal{A} \times_i F \times_j G.$$

Two i-mode multiplications satisfy the identity

$$(\mathcal{A} \times_i F) \times_i G = \mathcal{A} \times_i (GF).$$

This is easily proved using (9.5):

$$\begin{aligned}(\mathcal{A} \times_i F) \times_i G &= (\text{fold}_i(F(\text{unfold}_i(\mathcal{A})))) \times_i G \\ &= \text{fold}_i(G(\text{unfold}_i(\text{fold}_i(F(\text{unfold}_i(\mathcal{A})))))) \\ &= \text{fold}_i(GF \,\text{unfold}_i(\mathcal{A})) = \mathcal{A} \times_i (GF).\end{aligned}$$

9.3 A Tensor Singular Value Decomposition

The matrix singular value decomposition can be generalized to tensors in different ways. We will present one such generalization, that is analogous to an *approximate principal component analysis*. It is often referred to as the Higher Order SVD (HOSVD)[11].

Theorem 9.3 (HOSVD). *The tensor $\mathcal{A} \in \mathbb{R}^{l \times m \times n}$ can be written*

$$\mathcal{A} = \mathcal{S} \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}, \tag{9.6}$$

[11] Higher Order SVD is related to the Tucker model in psychometrics and chemometrics [85, 86].

where $U^{(1)} \in \mathbb{R}^{l\times l}$, $U^{(2)} \in \mathbb{R}^{m\times m}$, and $U^{(3)} \in \mathbb{R}^{n\times n}$ are orthogonal matrices. $\mathcal{S}$ is a tensor of the same dimensions as $\mathcal{A}$; it has the property of all-orthogonality*: any two slices of $\mathcal{S}$ are orthogonal in the sense of the scalar product* (9.1)*:*

$$\langle \mathcal{S}(i,:,:), \mathcal{S}(j,:,:)\rangle = \langle \mathcal{S}(:,i,:), \mathcal{S}(:,j,:)\rangle = \langle \mathcal{S}(:,:,i), \mathcal{S}(:,:,j)\rangle = 0,$$

for $i \neq j$. The 1-mode singular values *are defined*

$$\sigma_j^{(1)} = \|\mathcal{S}(i,:,:)\|_F, \qquad j = 1,\dots,l,$$

and they are ordered,

$$\sigma_1^{(1)} \geq \sigma_2^{(1)} \geq \cdots \geq \sigma_l^{(1)} \geq 0.$$

The singular values in other modes and their ordering are analogous.

The HOSVD is visualized in Figure 9.2.

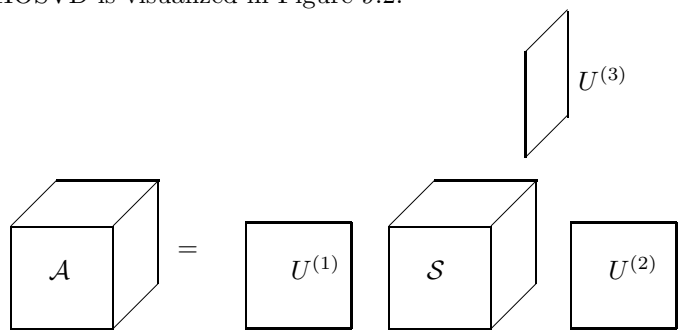


Figure 9.2. *Visualization of the HOSVD.*

Proof. (Sketch.) Compute the SVD's

$$A_{(i)} = U^{(i)}\Sigma^{(i)}(V^{(i)})^T, \quad i = 1,2,3, \tag{9.7}$$

and put

$$\mathcal{S} = \mathcal{A} \times_1 (U^{(1)})^T \times_2 (U^{(2)})^T \times_3 (U^{(3)})^T.$$

We now show that the slices of $\mathcal{S}$ are orthogonal,

$$\langle \mathcal{S}(i,:,:), \mathcal{S}(j,:,:)\rangle = 0, \tag{9.8}$$

if $i \neq j$ (and correspondingly in the other modes). From the definition of the unfolding operation it is seen that

$$S_{(1)}(i,:) = \begin{pmatrix}\mathcal{S}(i,1,:) & \mathcal{S}(i,2,:) & \dots & \mathcal{S}(i,m,:)\end{pmatrix},$$

i.e. the i'th row of $S_{(1)}$ consists of the row vectors of $\mathcal{S}(i,:,:)$, concatenated. Therefore, the orthogonality condition (9.8) is equivalent to $S_{(1)}S_{(1)}^T$ being diagonal. Using the identity (proved in Appendix A)

$$S_{(1)} = (U^{(1)})^T A_{(1)}(U^{(2)} \otimes U^{(3)}) = \Sigma^{(1)}(V^{(1)})^T(U^{(2)} \otimes U^{(3)}), \tag{9.9}$$

we get

$$\begin{aligned} S_{(1)}S_{(1)}^T &= (U^{(1)})^T A_{(1)}(U^{(2)} \otimes U^{(3)})(U^{(2)} \otimes U^{(3)})^T A_{(1)}^T U^{(1)} \\ &= \Sigma^{(1)}(V^{(1)})^T V^{(1)}(\Sigma^{(1)})^T = \Sigma^{(1)}(\Sigma^{(1)})^T, \end{aligned}$$

where we have used (9.7). The matrix $\Sigma^{(1)}(\Sigma^{(1)})^T$ is obviously diagonal.

It remains to show that the i-mode singular values are decreasingly ordered. Put $i = 1$. Then from (9.9) we see that $S_{(1)} = \Sigma^{(1)}Z$, where the rows of Z have Euclidean length 1. Since the 1-mode singular values are the norms of the rows of $S_{(1)}$, and since the diagonal elements of $\Sigma_{(1)}$ are decreasingly ordered, it follows that the 1-mode singular values are decreasingly ordered. The proof is analogous in the other modes. □

The all-orthogonal tensor $\mathcal{S}$ is usually referred to as the *core tensor*.

The equation (9.6) can also be written

$$\mathcal{A}_{ijk} = \sum_{p=1}^{l}\sum_{q=1}^{m}\sum_{s=1}^{n} u_{ip}^{(1)} u_{jq}^{(2)} u_{ks}^{(3)} \mathcal{S}_{pqs},$$

which has the following interpretation: The element $\mathcal{S}_{pqs}$ reflects the variation by the combination of the singular vectors $u_p^{(1)}$, $u_q^{(2)}$, and $u_s^{(3)}$.

The computation of the HOSVD is straightforward, and is implemented by the following Matlab code, although somewhat inefficiently[12].

```
function [U1,U2,U3,S,s1,s2,s3]=svd3(A);
% Compute the HOSVD of a 3-way tensor A

[U1,s1,v]=svd(unfold(A,1));
[U2,s2,v]=svd(unfold(A,2));
[U3,s3,v]=svd(unfold(A,3));

S=tmul(tmul(tmul(A,U1',1),U2',2),U3',3);
```

The function `tmul(A,X,i)` is assumed to multiply the tensor `A` by the matrix `X` in mode `i`.

Let V be orthogonal of the same dimension as U_i; then from the identities [57]

$$\mathcal{S} \times_i U^{(i)} = \mathcal{S} \times_i (U^{(i)}VV^T) = (\mathcal{S} \times_i V^T)(\times_i U^{(i)}V),$$

it may appear that the tensor SVD is not unique. However, the property that the i-mode singular values are ordered is destroyed by such transformations. Thus, the HOSVD is essentially unique; the exception is when there are equal singular values along any mode[13].

[12] Exercise: In what sense is the computation inefficient?

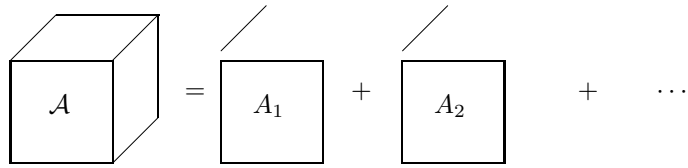
[13] This is the same type of non-uniqueness as with the matrix SVD.

9.3.1 Approximating a Tensor by HOSVD

A matrix can be written in terms of the SVD as a sum of rank one terms, see (6.2). An analogous expansion of a tensor can be derived using the definition of tensor-matrix multiplication: A tensor $\mathcal{A} \in \mathbb{R}^{l\times m\times n}$ can be expressed as a sum of matrices times singular vectors:

$$\mathcal{A} = \sum_{i=1}^{n} A_i \times_3 u_i^{(3)}, \qquad A_i = \mathcal{S}(:,:,i) \times_1 U^{(1)} \times_2 U^{(2)}, \tag{9.10}$$

where $u_i^{(3)}$ are column vectors in $U^{(3)}$. The A_i are to be identified as both matrices in $\mathbb{R}^{m\times n}$ and tensors in $\mathbb{R}^{m\times n\times 1}$. The expansion (9.10) is illustrated:



This expansion is analogous along the other modes.

It is easy to show that the A_i matrices are orthogonal in the sense of the scalar product (9.1):

$$\begin{aligned}\langle A_i, A_j\rangle &= \mathrm{tr}[U^{(2)}\mathcal{S}(:,:,i)^T(U^{(1)})^T U^{(1)}\mathcal{S}(:,:,j)(U^{(2)})^T] \\ &= \mathrm{tr}[\mathcal{S}(:,:,i)^T\mathcal{S}(:,:,j)] = 0;\end{aligned}$$

(here we have identified the slices $\mathcal{S}(:,:,i)$ with matrices, and used the identity $\mathrm{tr}(AB) = \mathrm{tr}(BA)$, which holds for square matrices).

It is now seen that the expansion (9.10) can be interpreted as follows: Each slice along the third mode of the tensor $\mathcal{A}$ can be written (exactly) in terms of the orthogonal basis $(A_i)_{i=1}^{r_3}$, where r_3 is the number of positive 3-mode singular values of $\mathcal{A}$:

$$\mathcal{A}(:,:,j) = \sum_{i=1}^{r_3} z_i^{(j)} A_i, \tag{9.11}$$

where $z_i^{(j)}$ is the j'th component of $u_i^{(3)}$. In addition, we have a simultaneous orthogonal factorization of the A_i,

$$A_i = \mathcal{S}(:,:,i) \times_1 U^{(1)} \times_2 U^{(2)},$$

which, due to the ordering of all the j-mode singular values for different j, has the property that the "mass" of each $\mathcal{S}(:,:,i)$ is concentrated at the upper left corner.

We illustrate the HOSVD in the following example.

Example 9.4 Given 131 handwritten digits[14], where each digit is a 16×16 matrix, we computed the HOSVD of the $16 \times 16 \times 131$ tensor. In Figure 9.3 we plot the

[14] From a US Postal Service Data Base, downloaded from the home page of [43].

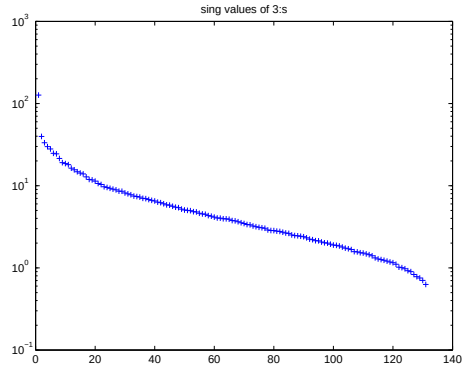


Figure 9.3. *The singular values in the digit (third) mode.*

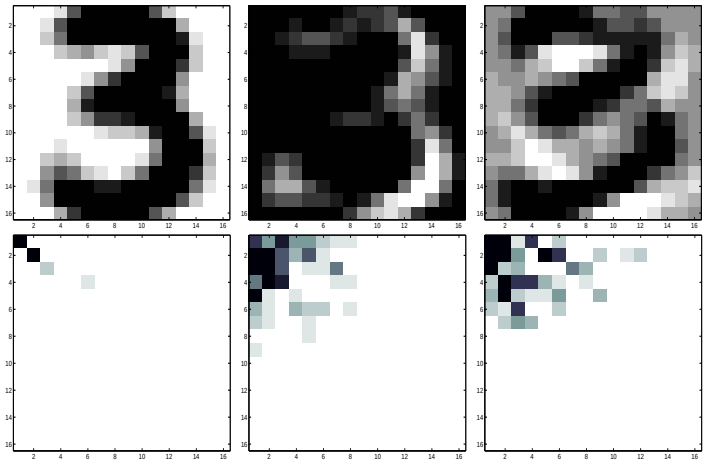


Figure 9.4. *The top row shows the three matrices A_1, A_2, and A_3, and the bottom row the absolute values of the three slices of the core tensor, $\mathcal{S}(:,:,1)$, $\mathcal{S}(:,:,2)$, and $\mathcal{S}(:,:,3)$.*

singular values along the third mode (different digits); it is seen that quite a large percentage of the variation of the digits is accounted for by the first 20 singular values (note the logarithmic scale). In fact

$$\frac{\sum_{i=1}^{20}(\sigma_i^{(3)})^2}{\sum_{i=1}^{131}(\sigma_i^{(3)})^2} \approx 0.91.$$

The first three matrices A_1, A_2, and A_3 are illustrated in Figure 9.4. It is seen that the first matrix looks like a mean value of different 3's; that is the dominating "direction" of the 131 digits, when considered as points in $\mathbb{R}^{256}$. The next two

images represent the dominating directions of variation from the "mean value" among the different digits.

In the bottom row of Figure 9.4 we plot the absolute values of the three slices of the core tensor, $\mathcal{S}(:,:,1)$, $\mathcal{S}(:,:,2)$, and $\mathcal{S}(:,:,3)$. It is seen that the mass of these matrices is concentrated at the upper left corner. ∎

If we truncate the expansion (9.10),

$$\mathcal{A} = \sum_{i=1}^{k} A_i \times_3 u_i^{(3)}, \qquad A_i = \mathcal{S}(:,:,i) \times_1 U^{(1)} \times_2 U^{(2)},$$

for some $k \leq n$, then we have an approximation of the tensor (here in the third mode) in terms of an orthogonal basis. We saw in (9.11) that each 3-mode slice $\mathcal{A}(:,:,j)$of $\mathcal{A}$ can be written as a linear combination of the orthogonal basis matrices A_j. Consider for instance the data mining application of classification of handwritten digits, cf. Chapter 11. There one may want to compute the coordinates of an unknown digit in terms of the orthogonal basis. This is easily done due to the orthogonality of the basis.

Example 9.5 Let Z denote an unknown digit. For classification purposes we want to compute the coordinates of Z in terms of the basis of 3's from the previous example. This is done by solving the least squares problem

$$\min_z \| Z - \sum_j z_j A_j \|_F,$$

where the norm is the matrix Frobenius norm. Put

$$G(z) = \frac{1}{2}\| Z - \sum_j z_j A_j \|_F^2 = \frac{1}{2}\langle Z - \sum_j z_j A_j, Z - \sum_j z_j A_j \rangle.$$

Since the basis is orthogonal with respect to the scalar product,

$$\langle A_i, A_j \rangle = 0, \quad \text{for} \quad i \neq j,$$

we can rewrite

$$G(z) = \frac{1}{2}\langle Z, Z \rangle - \sum_j z_j \langle Z, A_j \rangle + \frac{1}{2}\sum_j z_j^2 \langle A_j, A_j \rangle.$$

In order to find the minimum, we compute the partial derivatives with respect to the z_j, and put them equal to zero,

$$\frac{\partial G}{\partial z_j} = -\langle Z, A_j \rangle + z_j \langle A_j, A_j \rangle = 0,$$

which gives the solution of the least squares problem as

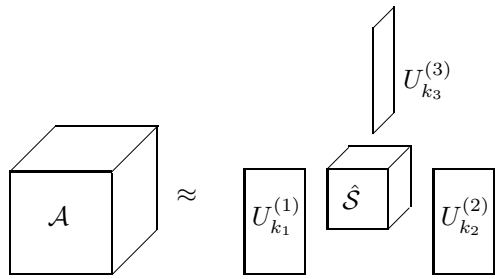
$$z_j = \frac{\langle Z, A_j \rangle}{\langle A_j, A_j \rangle}, \quad j = 1, 2, \ldots.$$

∎

It is the property that the mass of the core tensor is concentrated for small values of the three indices that makes it possible to perform a simultaneous data compression in all three modes by the HOSVD. We here assume that we compress to k_i columns in mode i. Let $U_{k_i}^{(i)} = U^{(i)}(:, 1:k_i)$ and $\hat{\mathcal{S}} = \mathcal{S}(1:k_1, 1:k_2, 1:k_3)$. Then consider the approximation

$$\mathcal{A} \approx \hat{\mathcal{A}} = \hat{\mathcal{S}} \times_1 U_{k_1}^{(i)} \times_2 U_{k_2}^{(2)} \times_3 U_{k_3}^{(3)}.$$

We illustrate this as follows.



Example 9.6 We compressed the basis matrices A_j of 3's from the example on page 97. In Figure 9.5 we illustrate the compressed basis matrices

$$\hat{A}_j = \mathcal{S}(1:8, 1:8, j) \times_1 U_8^{(1)} \times_2 U_8^{(2)}.$$

Cf. the corresponding full basis matrices in Figure 9.4. Note that the new basis matrices $\hat{A}_j$ are no longer orthogonal. ∎

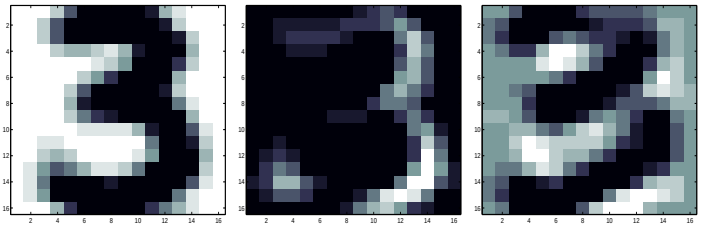


Figure 9.5. *Compressed basis matrices of handwritten 3's.*

Chapter 10
Clustering

One important method for data compression is to organize data points in *clusters*: A cluster is a subset of the set of data points that are close together, using some distance measure. One can compute the mean value of each cluster separately, and use the means as representatives of the clusters. Equivalently, the means are used as basis vectors, and all the data points are represented by their coordinates with respect to this basis.

Example 10.1 In Figure 10.1 we illustrate a set of data points in $\mathbb{R}^3$, generated from two correlated normal distributions. Assuming that we know that we have

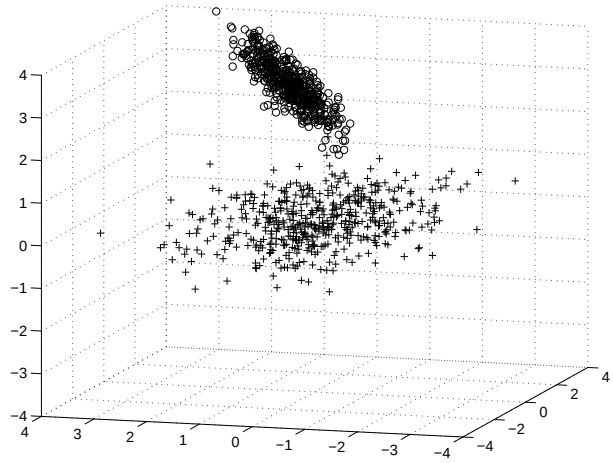


Figure 10.1. *Two clusters in* $\mathbb{R}^3$.

two clusters, we can easily determine visually which points belong to which class. A clustering algorithm takes the complete set of points and classifies them using some distance measure. ∎

There are several methods for computing a clustering. One of the most important is the *k-means algorithm*. We describe it in Section 10.1. There are also methods that are based on a low rank SVD approximation. One such method is given in Section 10.2.

We assume that there are n data points $(a_j)_{j=1}^m \in \mathbb{R}^m$, which can also be thought of as columns in a matrix $A \in \mathbb{R}^{m\times n}$. The vectors are normalized $\| a_j \|_2 = 1$.

10.1 The Spherical k-Means Algorithm

Let $\Pi = (\pi_i)_{i=1}^k$ denote a partitioning of the vectors $a_1, a_1, \ldots, a_n$ into k disjoint clusters:

$$\pi_i = \{j \,|\, a_j \text{ belongs to cluster } i\}.$$

Let the mean, or the centroid, of the cluster be

$$m_j = \frac{1}{n_j} \sum_{\nu\in\pi_j} a_\nu,$$

where n_j is the number of elements in π_j. The concept vectors [26] are the normalized centroids,

$$c_j = \frac{1}{\| m_j \|_2} m_j = \frac{1}{\| \sum_{\nu\in\pi_j} a_\nu \|_2} \sum_{\nu\in\pi_j} a_\nu.$$

We will describe the *spherical k-means algorithm*[15] [26] which is based on the cosine distance measure. This algorithm can be motivated from a maximization problem. We will need the following lemma.

Lemma 10.2. *The concept vector c_j is the unit vector that is closest in the cosine measure to all the document vectors in π_j, i.e., for an arbitrary unit vector z,*

$$\left| \sum_{\nu\in\pi_j} a_\nu^T z \right| \le \left| \sum_{\nu\in\pi_j} a_\nu^T c_j \right|, \tag{10.1}$$

holds.

Proof. Let z be an arbitrary unit vector, $\| z \|_2 = 1$. Using the Cauchy-Schwarz inequality ($|x^T y| \le \| x \|_2 \| y \|_2$), we have

$$\left| \sum_{\nu\in\pi_j} a_\nu^T z \right| \le \| \sum_{\nu\in\pi_j} a_\nu \|_2 \| z \|_2 = \| \sum_{\nu\in\pi_j} a_\nu \|_2.$$

On the other hand,

$$\begin{aligned} \left| c_j^T \sum_{\nu\in\pi_j} a_\nu \right| &= \frac{1}{\| \sum_{\nu\in\pi_j} a_\nu \|_2} \left| \left(\sum_{\nu\in\pi_j} a_\nu \right)^T \left(\sum_{\nu\in\pi_j} a_\nu \right) \right| \\ &= \frac{1}{\| \sum_{\nu\in\pi_j} a_\nu \|_2} \| \sum_{\nu\in\pi_j} a_\nu \|_2^2 = \| \sum_{\nu\in\pi_j} a_\nu \|_2, \end{aligned}$$

[15] "Spherical" due to the fact that the documents and the concept vectors are on the unit sphere.

so we have established (10.1). □

Motivated by the inequality (10.1) we can measure the coherence, or the quality, of cluster j by

$$\sum_{\nu \in \pi_j} a_\nu^T c_j,$$

and the combined coherence of the partitioning by the following objective function

$$Q(\Pi) = \sum_{j=1}^{k} \sum_{\nu \in \pi_j} a_\nu^T c_j.$$

Thus, we seek a partitioning that has optimal coherence, in the sense that it is the solution of the maximization problem

$$\max_{\Pi} Q(\Pi).$$

We now formulate the algorithm.

The spherical k-means algorithm

---

1. Start with an initial partioning $\Pi^{(0)}$ and compute the corresponding concept vectors, $(c_j^{(0)})_{j=1}^k$. Compute $Q(\Pi^{(0)})$. Put $t = 1$.
2. For each document vector a_i, find the closest concept vector using the cosine distance measure. If the closest vector is $c_p^{(t-1)}$, assign a_i to $\pi_p^{(t)}$.
3. Compute the concept vectors $(c_j^{(t)})_{j=1}^k$ of the new partitioning $\Pi^{(t)}$.
4. if $|Q(\Pi^{(t-1)}) - Q(\Pi^{(t)})| <$ tol then stop, otherwise increment t by 1 and go to step 2.

It can be proved [26] that the objective function in this algorithm is nondecreasing,

$$Q(\Pi^{(t)}) \leq Q(\Pi^{(t+1)}).$$

However, we cannot guarantee that the algorithm finds the global minimum.

Example 10.3 We computed 50 centroids for the Medline collection (with stemming, term frequency and inverse document frequency weighting (12.1)), using the spherical k-means algorithm. Before computing the clustering we normalized the columns to equal Euclidean length. The spherical k-means algorithm converged in 10 iterations (the relative change in the value of the objective function was then less than 10^{-3}), see Figure 10.2, where we give the values of the objective function. ■

10.2 Spectral Clustering

To be written.

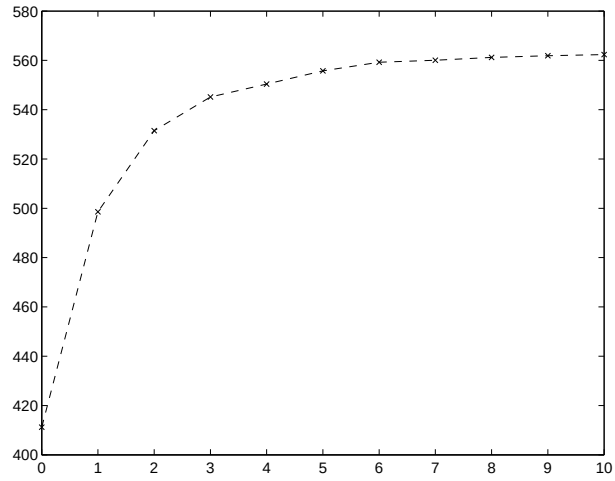


Figure 10.2. *The objective function in the spherical k-means algorithm for the Medline collection and 50 clusters.*

Part II

Applications

Chapter 11

Recognition of Handwritten Digits

Classification by computer of handwritten digits is a standard problem in pattern recognition. The typical application is automatic reading of zip codes on envelopes. A comprehensive review of different algorithms is given in [58].

11.1 Handwritten Digits and a Simple Algorithm

In Figure 11.1 we illustrate handwritten digits that we will use in the examples in this chapter.

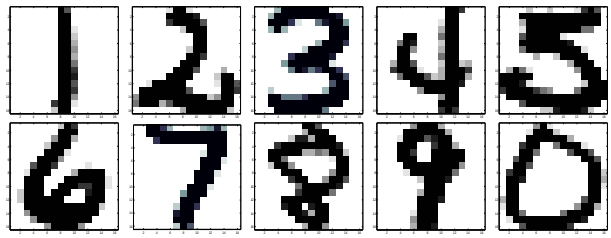


Figure 11.1. *Handwritten digits from the US Postal Service Database, see e.g. [43].*

We will treat the digits in three different, but equivalent ways:

1. as 16×16 greyscale images,
2. as functions of two variables,
3. as vectors in $\mathbb{R}^{256}$.

In Figure 11.2 we illustrate a digit as a function of two variables, $s = s(x, y)$.

In the classification of an unknown digit we need to compute the distance to known digits. Different distance measures can be used, perhaps the most natural is to use Euclidean distance: stack the columns of the image in a vector and identify each digit as a vector in $\mathbb{R}^{256}$. Then define the distance function

$$\underset{\cdot}{(x, y)} = \| x - y \|_2.$$

An alternative distance function is the cosine between two vectors.

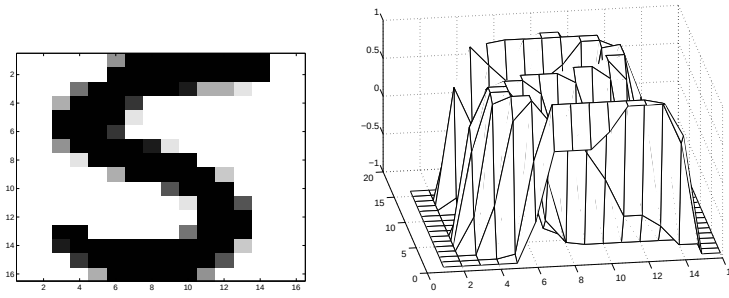


Figure 11.2. *A digit considered as a function of two variables.*

In a real application of recognition of handwritten digits, e.g. zip code reading, there are hardware and real time factors that must be taken into account. In this chapter we will describe an idealized setting. The problem is:

> *Given a set of of manually classified digits (the training set), classify a set of unknown digits (the test set).*

In the US Postal Service Data, the training set contains 7291 handwritten digits. Here we will use a subset of 1707 digits, relatively equally distributed between 0 and 9. The test set has 2007 digits.

When we consider the training set digits as vectors or points, then it is reasonable to assume that all digits of one kind form a cluster of points in a Euclidean 256-dimensional vector space. Ideally the clusters are well separated (otherwise the task of classifying unknown digits will be very difficult), and the separation between the clusters depends on how well written the training digits are.

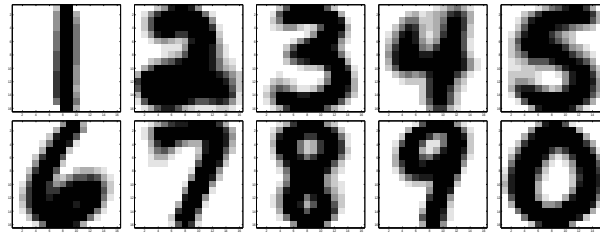


Figure 11.3. *The means (centroids) of all digits in the training set.*

In Figure 11.3 we illustrate the means (centroids) of the digits in the training set. From the figure we get the impression that a majority of the digits are well-written (if there were many badly written digits the means would be very diffuse). This indicates that the clusters are rather well separated. Therefore it seems likely that a simple classification algorithm that computes the distance from each unknown digit to the means may be reasonably accurate.

A simple classification algorithm

Training: Given the training set, compute the mean (centroid) of all digits of one kind.

Classification: For each digit in the test set, compute the distance to all ten means, and classify as the closest.

It turns out that for our test set the success rate of this algorithm is around 75 %, which is not good enough. The reason for this realtively bad performance is that the algorithm does not use any information about the variation within each class of digits.

11.2 Classification using SVD Bases

We will now describe a classification algorithm that is based on the modelling of the variation within each digit class using orthogonal basis vectors computed using the SVD.

If we consider the images as 16×16 matrices, then the data are multi-dimensional, see Figure 11.4. Stacking all the columns of each image above each other gives a matrix. Let $A \in \mathbb{R}^{m \times n}$, with $m = 256$, be the matrix consisting of

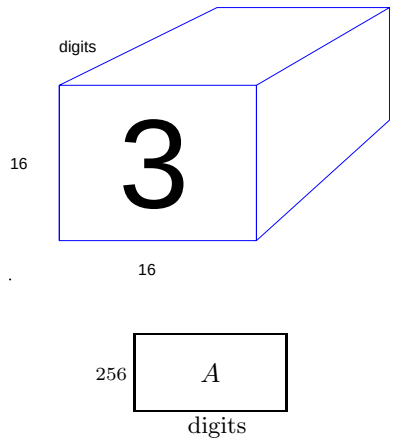


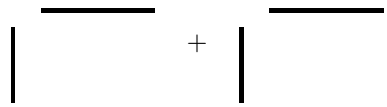
Figure 11.4. *The image of one digit is a matrix, and the set of images of one kind form a tensor. In the lower part of the figure each digit (of one kind) is represented by a column in the matrix.*

all the training digits of one kind, the 3's, say. The columns of A span a linear

subspace of $\mathbb{R}^m$. However, this subspace can not be expected to have a large dimension, because if it did have that, then the subspaces of the different kinds of digits would intersect (remember that we are considering subspaces of $\mathbb{R}^m$).

The idea now is to "model" the variation within the set of training (and test) digits of one kind using an orthogonal basis of the subspace. An orthogonal basis can be computed using the SVD, and any matrix A is a sum of rank 1 matrices (6.2)

$$A = \sum_{i=1}^{m} \sigma_i u_i v_i^T =$$



Each column in A is an image of a digit 3, and therefore the left singular vectors u_i are an orthogonal basis in the "image space of 3's". We will refer to the left singular vectors as "singular images". The coordinates of image j in A in terms of this basis is

$$a_{\cdot j} = \sum_{i=1}^{m} (\sigma_i v_{ij}) u_i.$$

From the matrix approximation properties of the SVD (Theorems 6.6 and 6.7) we know that the first singular vector represents the "dominating" direction of the data matrix. Therefore, if we fold the vectors u_i back to images, we expect the first singular vector to look like a 3, and the following singular images should represent the dominating variations of the training set around the first singular image. In Figure 11.5 we illustrate the singular values and the first three singular images for the training set 3's. In the middle graph we plot the coordinates of each of the 131 digits in terms of the first three singular vectors. We see that all the digits have a large portion (between 0.05 and 0.1) of the first singular image, which, in fact, looks very much like the mean of 3's in Figure 11.3. We then see that there is a rather large variation in the coordinates in terms of the second and third singular image.

The SVD basis classification algorithm will be based on the following assumptions.

1. Each digit (in the training and test set) is well characterized by a few of the first singular images of its own kind. The more precise meaning of "few" should be investigated by experiments.

2. An expansion in terms of the first few singular images discriminates well between the different classes of digits.

3. If an unknown digit can be better approximated in one particular basis of singular images, the basis of 3's say, and than in the bases of the other classes, then it is likely that the unknown digit is a 3.

Thus we should compute how well an unknown digit can be represented in the ten different bases. This can be done by computing the residual vector in *least*

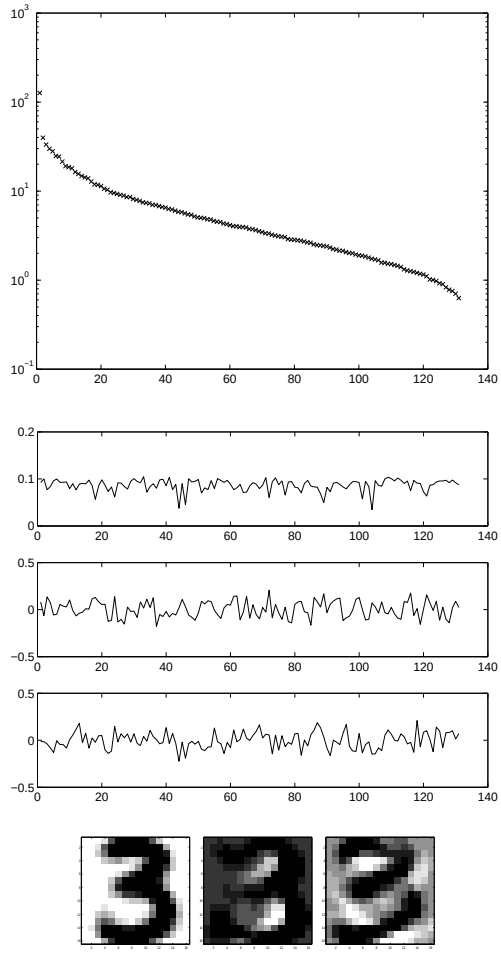


Figure 11.5. *Singular values (top), coordinates of the 131 test digits in terms of the first three right singular vectors v_i (middle), and the first three singular images.*

squares problems of the type

$$\min_{\alpha_i} \|z - \sum_{i=1}^{k} \alpha_i u_i\|,$$

where z represents an unknown digit, and u_i the singular images. We can write this problem in the form

$$\min_{\alpha} \| z - U_k \alpha \|_2,$$

where $U_k = \begin{pmatrix} u_1 & u_2 & \cdots & u_k \end{pmatrix}$. Since the columns of U_k are orthogonal, the solu-

tion of this problem is given by $\alpha = U_k^T z$, and the norm of the residual vector of the least squares problems is

$$\| (I - U_k U_k^T) z \|_2, \tag{11.1}$$

i.e., the norm of the projection of the unknown digit onto the subspace orhtogonal to span(U_k).

In order to demonstrate that the assumptions above are reasonable we illustrate in Figure 11.6 the relative residual norm for all test 3's and 7's in terms of all 10 bases. In the two figures there is one curve for each unknown digits, and

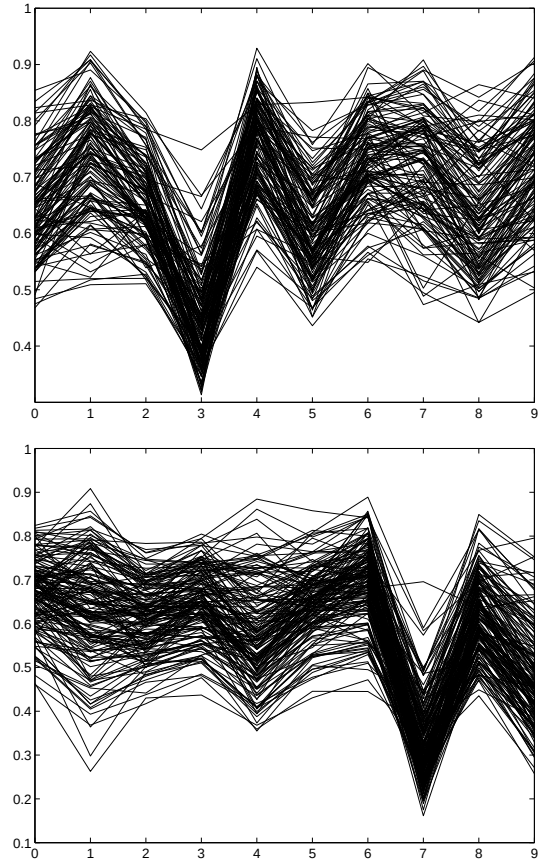


Figure 11.6. *Relative residuals of all test 3's (top) and 7's in terms of all bases. Ten basis vectors were used for each class.*

naturally it is not possible to see the individual curves. However, one can see that most of the test 3's and 7's are best approximated in terms of "their own basis". The graphs also give information about which classification errors are more likely

than others (e.g. 3's and 5's are similar whereas 3's and 4's are quite different; of course this only confirms what we already know).

It is also interesting to see how the residual depends on the number of terms in the basis. In Figure 11.7 we illustrate the approximation of a nicely written 3 in terms of the 3-basis with different numbers of basis images. In Figures 11.8 and 11.9 we show the approximation of an ugly 3 in the 3-basis and a nice 3 in the 5-basis.

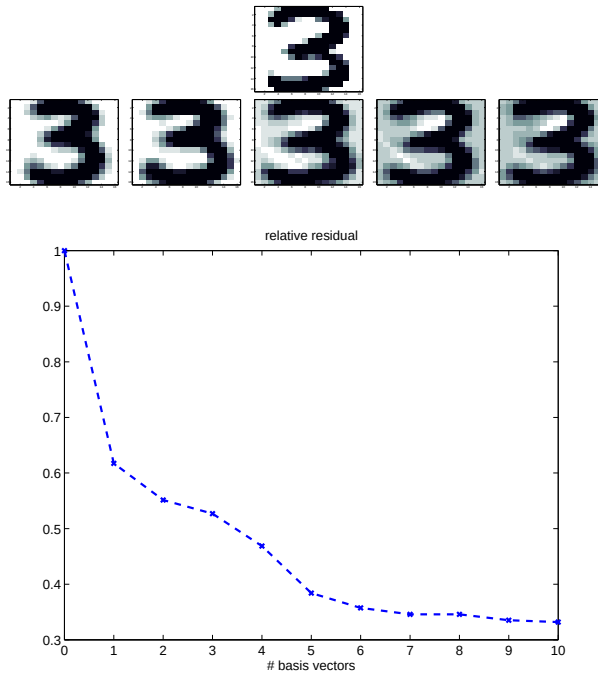


Figure 11.7. *Unknown digit (nice 3) and approximations using 1, 3, 5, 7, and 9 terms in the 3-basis (top). Relative residual* $\| (I - U_k U_k^T) z \|_2 / \| z \|_2$ *in least squares problem (bottom).*

From Figures 11.7 and 11.9 we see that the relative residual is considerably smaller for the nice 3 in the 3-basis than in the 5-basis. We also see from Figure 11.8 that the ugly 3 is not well represented in terms of the 3-basis. Therefore, naturally, if the digits are very badly drawn, then we can not expect to get a clear classification based the SVD bases.

It is possible to devise several classification algorithm based on the model of expanding in terms of SVD bases. Below we give a simple variant.

A SVD basis classification algorithm

Training:

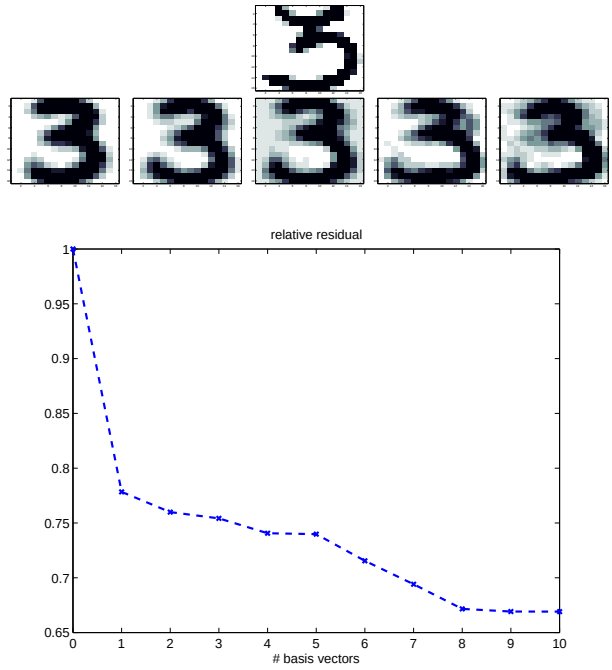


Figure 11.8. *Unknown digit (ugly 3) and approximations using 1, 3, 5, 7, and 9 terms in the 3-basis (top). Relative residual in least squares problem (bottom).*

For the training set of known digits, compute the SVD of each set of digits of one kind.

Classification:

For a given test digit, compute its relative residual in all ten bases. If one residual is significantly smaller than all the others, classify as that. Otherwise give up.

The work in this algorithm can be summarized as follows.

Training: compute SVD's of 10 matrices of dimension $m^2 \times n_i$
Each digit is a $m \times m$ digitized image
n_i: the number of training digits i

Test: Compute 10 least squares residuals (11.1).

Thus the test phase is quite fast, and this algorithm should be suitable for real time computations. The algorithm is related to the SIMCA method [77].

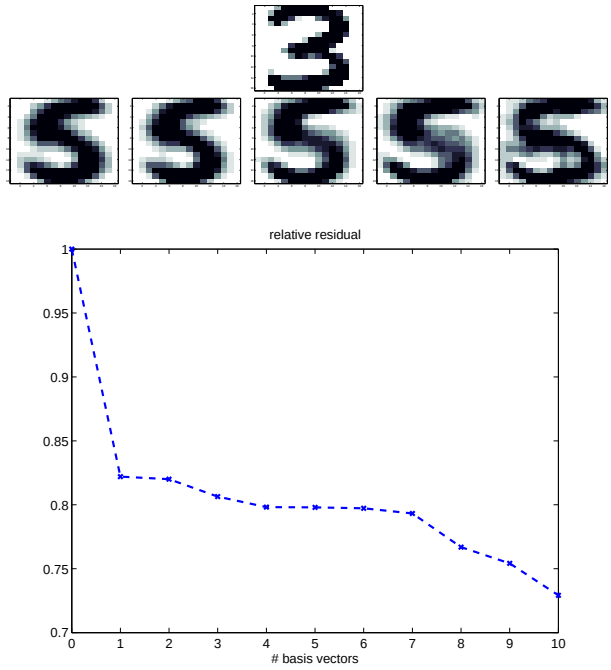


Figure 11.9. *Unknown digit (nice 3) and approximations using 1, 3, 5, 7, and 9 terms in the 5-basis (top). Relative residual in least squares problem (bottom).*

We next give some test results[16] for the US Postal Service data set, here with 7291 training digits and 2007 test digits, [43]. In Table 11.1 we give classification results as a function of the number of basis images for each class.

# basis images	1	2	4	6	8	10
correct (%)	80	86	90	90.5	92	93

Table 11.1. *Correct classifications as a function of the number of basis images (for each class).*

Even if there is a very significant improvement in performance compared to the method where one only used the centroid images, the results are not good enough, as the best algorithms reach about 97 % correct classifications. It should be said that the training and test contain some digits that are very difficult to classify, we give a few examples in Figure 11.10. Such badly written digits are very difficult to handle automatically.

[16] From the Masters thesis of Berkant Savas, Linköping University [72].

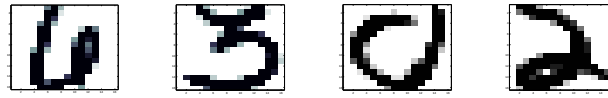


Figure 11.10. *Ugly digits in the US Postal Service data set.*

11.3 Tangent Distance

Even if the badly written digits in Figure 11.10 might be considered as too bad to be classified, there are some deviations from an ideal digit that are very common and acceptable. We illustrate a few such variations in Figure 11.11. In the first

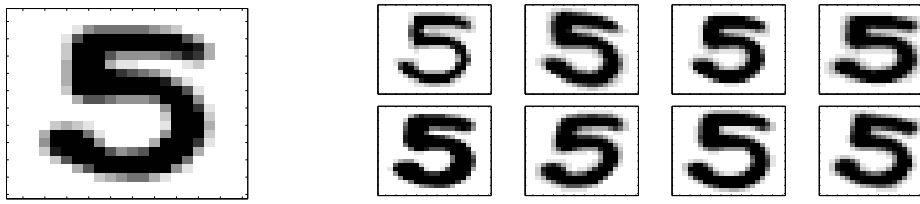


Figure 11.11. *A digit and acceptable transformations.*

transformation the digit has been written[17] with a thinner and thicker pen, in the second the digit has been stretched diagonalwise, in the third it has been compressed and elongated vertically, and in the fourth it has been rotated. Such transformations constitute no difficulties for a human reader and ideally they should be very easy to deal with in automatic digit recognition. A distance measure, *tangent distance*, that is invariant under small such transformations is described in [74, 75].

16×16 images can be interpreted as points in $\mathbb{R}^{256}$. Let p be a fixed pattern in an image. We shall first consider the case of only one allowed transformation, translation of the pattern (digit) in the x-direction, say. This translation can be thought of as moving the pattern along a curve in $\mathbb{R}^{256}$. Let the curve be parametrized by a real parameter α so that the curve is given by $s(p, \alpha)$, and in such a way that $s(p, 0) = p$. In general, the curve is nonlinear, and can be approximated by the first two terms in the Taylor expansion,

$$s(p, \alpha) = s(p, 0) + \frac{ds}{d\alpha}(p, 0)\, \alpha + O(\alpha^2) \approx p + t_p \alpha,$$

where $t_p = \frac{ds}{d\alpha}(p, 0)$ is a vector in $\mathbb{R}^{256}$. By varying α slightly around 0, we make a small movement of the pattern along the tangent at the point p on the curve. Assume that we have another pattern e that is approximated similarly

$$s(e, \alpha) \approx e + t_e \alpha.$$

Since we consider small movements along the curves as allowed, such small move-

[17] Note that the transformed digits have not been written manually but using the techniques described later in this section. The presentation in this section is based on the papers [74, 75] and

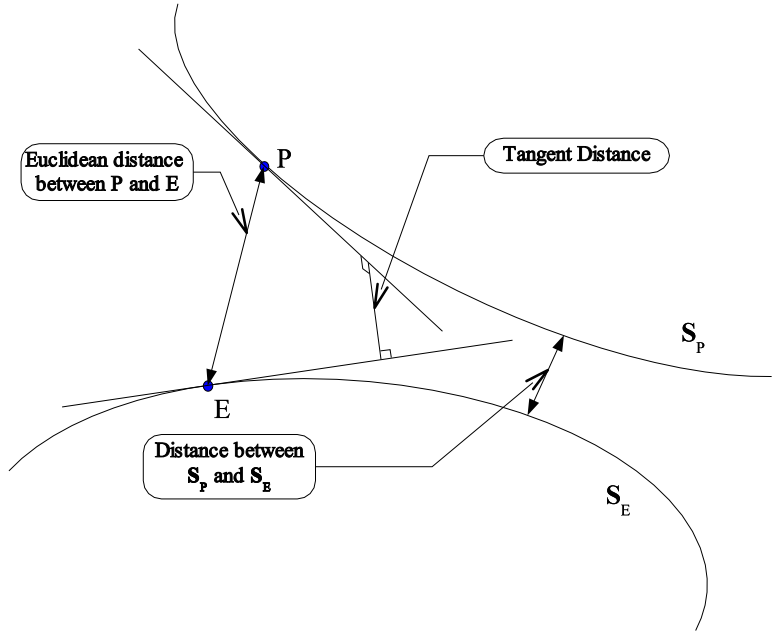


Figure 11.12. *Tangent distance (cf. [75]).*

ments should not influence the distance function. Therefore, ideally we would like to define our measure of closeness between p and e as the closest distance between the two curves, see Figure 11.12.

However, since in general we cannot compute the distance the curves we can use the first order approximations, and compute the closest distance between the two tangents in the points p and e. Thus we shall move the patterns independently along their respective tangents, until we find the smallest distance. If we measure this distance in the usual euclidean norm, we shall solve the least squares problem

$$\min_{\alpha_p,\alpha_e} \| p + t_p\alpha_p - e - t_e\alpha_e \|_2 = \min_{\alpha_p,\alpha_e} \| (p-e) - \begin{pmatrix} -t_p & t_e \end{pmatrix} \begin{pmatrix} \alpha_p \\ \alpha_e \end{pmatrix} \|_2.$$

Consider now the case when we are allowed to move the pattern p along l different curves in $\mathbb{R}^{256}$, parametrized by $\alpha = (\alpha_1 \cdots \alpha_l)^T$. This is equivalent to moving the pattern on an l-dimensional surface (manifold) in $\mathbb{R}^{256}$. Assume that we have two patterns, p and e, each of which is allowed to move on its surface of allowed transformations. Ideally we would like to find the closest distance between the surfaces, but instead, since this is not possible to compute, we now define a

the Masters thesis of Berkant Savas, Linköping University [72].

distance measure where we compute the distance between the two *tangent planes* of the surface in the points p and e.

As before, the tangent plane is given by the first two terms in the Taylor expansion of the function $s(p, \alpha)$:

$$s(p,\alpha) = s(p,0) + \sum_{i}^{l} \frac{ds}{d\alpha_i}(p,0)\,\alpha_i + O(\| \alpha \|_2^2) \approx p + T_p\alpha,$$

where T_p is the matrix

$$T_p = \begin{pmatrix} \frac{ds}{d\alpha_1} & \frac{ds}{d\alpha_2} & \cdots & \frac{ds}{d\alpha_l} \end{pmatrix},$$

and the derivatives are all evaluated in the point $(p, 0)$.

Thus the *tangent distance* between the points p and e is defined as the smallest possible residual in the least squares problem

$$\min_{\alpha_p,\alpha_e} \| p + T_p\alpha_p - e - T_e\alpha_e \|_2 = \min_{\alpha_p,\alpha_e} \| (p-e) - \begin{pmatrix} -T_p & T_e \end{pmatrix} \begin{pmatrix} \alpha_p \\ \alpha_e \end{pmatrix} \|_2.$$

The least squares problem can be solved, e.g. using the singular value decomposition of $A = \begin{pmatrix} -T_p & T_e \end{pmatrix}$. Note that we are actually not interested in the solution itself but only the norm of the residual. Write the lest squares problem in the form

$$\min_{\alpha} \| b - A\alpha \|_2, \qquad b = p - e, \quad \alpha = \begin{pmatrix} \alpha_p \\ \alpha_e \end{pmatrix}.$$

If we use the QR decomposition[18]

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_1\, Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R,$$

the norm of the residual is given by

$$\begin{aligned} \min_{\alpha} \| b - A\alpha \|_2^2 &= \min_{\alpha} \left\| \begin{pmatrix} Q_1^T b - R\alpha \\ Q_2^T b \end{pmatrix} \right\|^2 \\ &= \min_{\alpha} \left[\| \left(Q_1^T b - R\alpha \right) \|_2^2 + \| Q_2^T b \|_2^2 \right] = \| Q_2^T b \|_2^2. \end{aligned}$$

The case when the matrix A should happen not to have full column rank is easily dealt with using the SVD, see Section 6.7. The probability is high that the columns of the tangent matrix are almost linearly dependent when the two patterns are close.

The most important property of this distance function is that it is *invariant under movements of the patterns on the tangent planes*. For instance, if we make a small translation in the x-direction of a pattern, then with this measure the distance it has been moved is equal to zero.

[18] A has dimension $256 \times 2l$; since the number of transformations is usually less than 10, the linear system is overdetermined.

11.3.1 Transformations

We will here consider the image pattern as a function of two variables, $p = p(x, y)$, and we will demonstrate that the derivative of each transformation can be expressed as a differentiation operator that is a linear combination of the derivatives $p_x = \frac{dp}{dx}$ and $p_y = \frac{dp}{dy}$.

Translation The simplest transformation is the one when the pattern is translated by α_x in the x-direction, i.e.

$$s(p, \alpha_x)(x, y) = p(x + \alpha_x, y).$$

Obviously, using the chain rule,

$$\frac{d}{d\alpha_x} \left(s(p, \alpha_x)(x, y) \right) |_{\alpha_x = 0} = \frac{d}{d\alpha_x} p(x + \alpha_x, y)|_{\alpha_x = 0} = p_x(x, y)$$

In Figure 11.13 we give a pattern and its x-derivative. Then we demonstrate that by adding a small multiple of the derivative the pattern can be translated to the left and to the right.

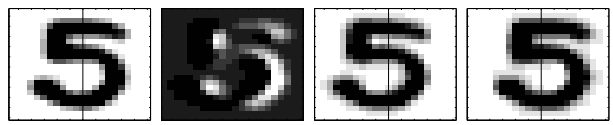


Figure 11.13. *A pattern, its x-derivative, and x-translations of the pattern.*

Analogously, for y-translation we get

$$\frac{d}{d\alpha_y} \left(s(p, \alpha_y)(x, y) \right) |_{\alpha_y = 0} = p_y(x, y).$$

Rotation A rotation of the pattern by an angle α_r is made by replacing the value of p in the point (x, y) by the value in the point

$$\begin{pmatrix} \cos \alpha_r & \sin \alpha_r \\ -\sin \alpha_r & \cos \alpha_r \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

Thus we define the function

$$s(p, \alpha_r)(x, y) = p(x \cos \alpha_r + y \sin \alpha_r, -x \sin \alpha_r + y \cos \alpha_r),$$

and we get the derivative

$$\frac{d}{d\alpha_r} \left(s(p, \alpha_r)(x, y) \right) = (-x \sin \alpha_r + y \cos \alpha_r) p_x + (-x \cos \alpha_r - y \sin \alpha_r) p_y.$$

Setting $\alpha_r = 0$, we have

$$\frac{d}{d\alpha_r}\left(s(p,\alpha_r)(x,y)\right)|_{\alpha_r=0} = yp_x - xp_y,$$

where the derivatives are evaluated at (x, y).

An example of a rotation transformation is given in Figure 11.14.

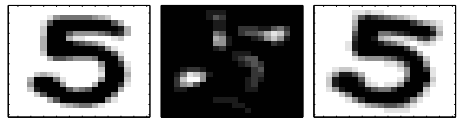


Figure 11.14. *A pattern, its "rotational" derivative, and a rotation of the pattern.*

Scaling A scaling of the pattern is achieved by defining

$$s(p,\alpha_s)(x,y) = p((1+\alpha_s)x, (1+\alpha_s)y),$$

and we get the derivative

$$\frac{d}{d\alpha_s}\left(s(p,\alpha_s)(x,y)\right)|_{\alpha_s=0} = xp_x + yp_y.$$

The scaling transformation is illustrated in Figure 11.15.

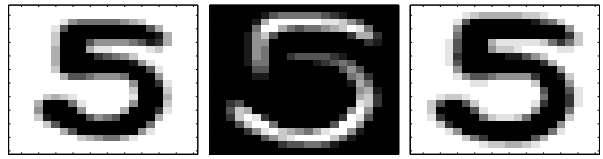


Figure 11.15. *A pattern, its "scaling" derivative, and an "up-scaling" of the pattern.*

Parallel Hyperbolic Transformation By defining

$$s(p,\alpha_p)(x,y) = p((1+\alpha_p)x, (1-\alpha_p)y),$$

we can stretch the pattern parallel to the axis. The derivative is

$$\frac{d}{d\alpha_p}\left(s(p,\alpha_p)(x,y)\right)|_{\alpha_p=0} = xp_x - yp_y.$$

In Figure 11.16 we illustrate the parallel hyperbolic transformation.

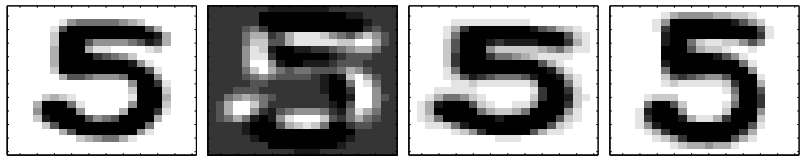


Figure 11.16. *A pattern, its "parallel hyperbolic" derivative, and two stretched patterns.*

Diagonal Hyperbolic Transformation By defining

$$s(p, \alpha_h)(x, y) = p(x + \alpha_h y, y + \alpha_h x),$$

we can stretch the pattern along diagonals. The derivative is

$$\frac{d}{d\alpha_h} \left(s(p, \alpha_h)(x, y)\right) |_{\alpha_h=0} = yp_x + xp_y.$$

In Figure 11.17 we illustrate the diagonal hyperbolic transformation.

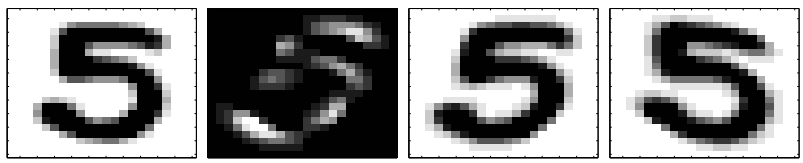


Figure 11.17. *A pattern, its "diagonal hyperbolic" derivative, and two stretched patterns.*

Thickening The pattern can be made thinner or thicker using similar techniques, for more detail, see [75]. The "thickening" derivative is

$$(p_x)^2 + (p_y)^2$$

Thickening and thinning is illustrated in Figure 11.18.

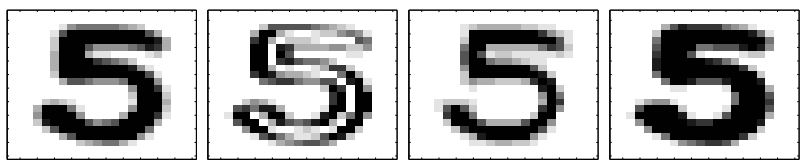


Figure 11.18. *A pattern, its "thickening" derivative, a thinner and a thicker paatern.*

A tangent distance classification algorithm

Training:

For the training set of known digits, compute the tangent matrix T_p.

Classification:

For each test digit:

- Compute its tangent
- Compute the tangent distance to all training (known) digits and classify as the one with shortest distance

Although this algorithm is quite good in terms of classification performance (96.9 % correct classification for the US Postal Service data set [72]), it is very expensive, since each test digit is compared to all the training digits. In order to make it competitive it must be combined with some other algorithm that reduces the number of tangent distance comparisons to make.

We end this chapter by remarking that it is necessary to preprocess the digits in different ways in order to enhance the classification, see [58]. For instance, performance is improved if the images are smoothed (convolved with a Gaussian kernel) [75]. In [72] the derivatives p_x and p_y are computed numerically by finite differences.

Chapter 12

Text Mining

By text mining we understand methods for extracting useful informations from large and often unstructured collections of texts. Another, closely related term is *information retrieval*. A typical application is search in data bases of abstract of scientific papers. For instance, in a medical applications one may want to find all the abstracts in the data base that deal with a particular syndrome. So one puts together a search phrase, a *query*, with key words that are relevant for the syndrome. Then the retrieval system is used to match the query to the documents in the data base, and present to the user all the documents that are relevant, ranked according to relevance.

Example 12.1 The following is a typical query for search in a collection of medical abstracts.

> 9. *the use of induced hypothermia in heart surgery, neurosurgery, head injuries and infectious diseases.*

The query is taken from a test collection for information retrieval, called MEDLINE[19]. In the sequel we will refer to this query as Q9. ∎

Library cataloques are another example of text mining applications.

Example 12.2 To illustrate one issue in information retrieval we performed a search in Linköping University library journal catalogue:

Search phrase	Result
computer science engineering	Nothing found
computing science engineering	IEEE: Computing in Science and Engineering

Naturally we would like the system to be insensitive to small errors on the part of the user. Any human can see that the IEEE journal is close to the query. From this example we see that in many cases straightforward word matching is not good enough. ∎

A very well-known area of text mining is web search engines. There the search phrase is usually very short, and often there are so many relevant documents that

[19] See e.g. `http://www.dcs.gla.ac.uk/idom/ir_resources/test_collections/`

it is out of the question to present them all to the user. In that application the ranking of the search result is critical for the efficiency of the search engine. We will come back to this problem in Chapter 13.

For an overview of information retrieval, see, e.g. [39]. In this chapter we will describe briefly one of the most common methods for text mining, namely the *vector space model* [71]. Here we will give a brief overview of the vector space model and a couple of variants: Latent Semantic Indexing (LSI), which uses the SVD of the term-document matrix, and clustering-based methods. For a more detailed account of the different techniques used in connection with the vector space model, see [12].

12.1 Preprocessing the Documents and Queries

In this section we describe the preprocessing that is done to the texts before the vector space model of a particular collection of documents is set up.

In information retrieval, keywords that carry information about the contents of a document are called *terms*. A basic step in information retrieval is to create a list of all the terms in a document collection, a so called *index*. For each term a list is stored of all the documents that contain that particular term. This is called an *inverted index*.

But before the index is made, two preprocessing steps must be done: elimination of all stop words, and stemming.

Stop words, are words that one can find in virtually any documents that is long enough. Thus the occurrence of such a word in a document does not distinguish this document from other documents. The following is the beginning of one particular stop list[20]

> a, a's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, aside, ask,...

Stemming is the process of reducing each word that is conjugated or has a suffix to its stem. Clearly, from the point of view of information retrieval, no information is lost in the following reduction.

$$\left.\begin{array}{r}\textbf{comput}\text{able}\\ \textbf{comput}\text{ation}\\ \textbf{comput}\text{ing}\\ \textbf{comput}\text{ed}\\ \textbf{comput}\text{ational}\end{array}\right\} \longrightarrow \quad \textbf{comput}$$

Public domain stemming algorithms are available on the Internet[21].

[20] `ftp://ftp.cs.cornell.edu/pub/smart/english.stop`

[21] `http://gatekeeper.dec.com/pub/net/infosys/wais/ir-book-sources/stemmer`, `http://www.tartarus.org/~martin/PorterStemmer/`.

without stemming	with stemming
action	action
actions	
activation	activ
active	
actively	
activities	
activity	
acts	
actual	actual
actually	
acuity	acuiti
acute	acut
ad	ad
adaptation	adapt
adaptations	
adaptive	
add	add
added	
addition	addit
additional	

Table 12.1. *The beginning of the index for the MEDLINE collection. The Porter stemmer and the GTP parser [33] were used.*

Example 12.3 We parsed the 1063 documents (actually 30 queries and 1033 documents) in the MEDLINE collection, without and with stemming, in both cases removing stop words. For consistency it is necessary to perform the same stemming to the stop list. In the first case the number of terms was 5839 and in the second 4281. We show partial lists of terms in Table 12.1. ∎

12.2 The Vector Space Model

The main idea in the vector space model is to create a *term-document matrix*, where each document is represented by a column vector. The column has nonzero entries in the positions that correspond to terms that can be found in the document. Consequently, each row represents a term, and has nonzero entries in those that positions that correspond to the documents where the term can be found.

A simplified example of a term-document matrix is given in Chapter 1. There we manually counted the frequency of the terms. For realistic problems one uses a *text parser* to create the term-document matrix.

Two public-domain parsers are described in [33, 101]. Unless otherwise stated we have used the one from [101] for the larger examples in this chapter. Text parsers for information retrieval usually include both a stemmer and an option to remove stop words. In addition there are filters, e.g. for removing formatting code in the

documents, e.g. HTML or XML.

It is common not only to count the occurrence of terms in documents but also to apply a *term weighting scheme*, where the elements of A are weighted depending on the characteristics of the document collection. Similarly, document weighting is usually done. A number of schemes are described in [12, Section 3.2.1]. For example, one can define the elements in A by

$$a_{ij} = f_{ij} \log(n/n_i), \tag{12.1}$$

where f_{ij} is term frequency, the number of times term i appears in document j, and n_i is the number of documents that contain term i (inverse document frequency). If a term occurs frequently in only a few documents, then both factors are large. In this case the term discriminates well between different groups of documents, and it gets a large weight in the documents where it appears.

Normally, the term-document matrix is *sparse*: most of the matrix elements are equal to zero. Then, of course, one avoids storing all the zeros, and uses instead a sparse matrix storage scheme, see Section 16.7.

Example 12.4 For the stemmed Medline collection, parsed using GTP, the matrix (including 30 query columns) is 4163×1063 with 48263 non-zero elements, i.e. approximately 1 %. The first 500 rows and columns of the matrix are illustrated in Figure 12.1. ∎

12.3 Query Matching and Performance Modelling

Query matching is the process of finding all documents that are relevant to a particular query q. This is often done using the cosine distance measure: A document a_j is returned if the angle between the query q and a_j is small enough. Equivalently, a_j is returned if

$$\cos(\theta(q, a_j)) = \frac{q^T a_j}{\| q \|_2 \| a_j \|_2} > \text{tol},$$

where tol is predefined tolerance. If the tolerance is lowered, then more documents are returned, and it is likely that many of those are relevant to the query. But at the same time there is a risk that when the tolerance is lowered more and more documents that are not relevant are also returned.

Example 12.5 We did query matching for query Q9 in the stemmed Medline collection. With tol = 0.19 only document 409 was considered relevant. When the tolerance was lowered to 0.17, then documents 409, 415, and 467 were retrieved. ∎

We illustrate the different categories of documents in a query matching for two values of the tolerance in Figure 12.2. The query matching produces a good result when the intersection between the two sets of returned and relevant documents is as large as possible, and the number of returned irrelevant documents is small. For a high value of the tolerance, the retrieved documents are likely to be

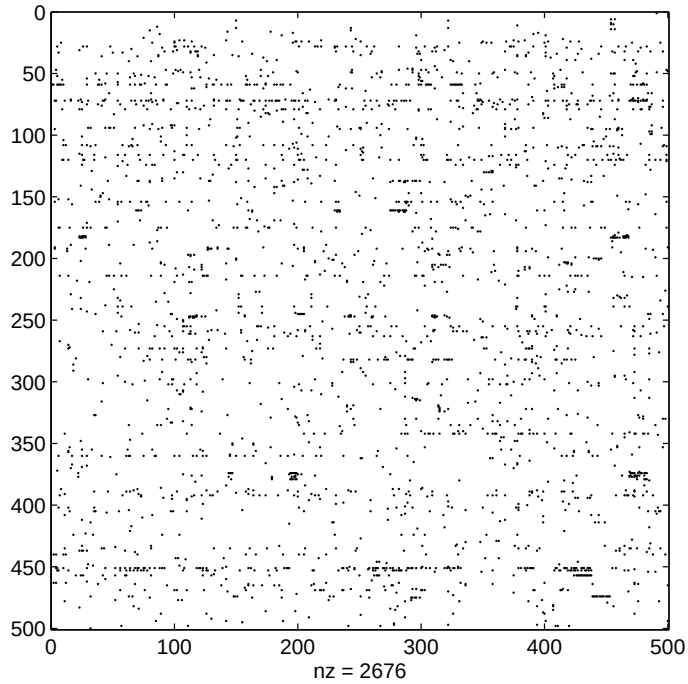


Figure 12.1. *The first 500 rows and columns of the Medline matrix. Each dot represents a non-zero element.*

relevant (the small circle in Figure 12.2). When the cosine tolerance is lowered, then the intersection is increased, but at the same time, more irrelevant documents are returned.

In performance modelling for information retrieval we then define the following measures:

$$\textit{Precision:} \qquad P = \frac{D_r}{D_t},$$

where D_r is the number of relevant documents retrieved, and D_t the total number of documents retrieved,

$$\textit{Recall:} \qquad R = \frac{D_r}{N_r},$$

where N_r is the total number of relevant documents in the data base. With the cosine measure, we see that with a large value of tol we have high precision, but low recall. For a small value of tol we have high recall, but low precision.

In the evaluation of different methods and models for information retrieval usually a number of queries are used. For testing purposes all documents have been read by a human and those that are relevant for a certain query are marked. This makes it possible to draw diagrams of recall versus precision, that illustrate the

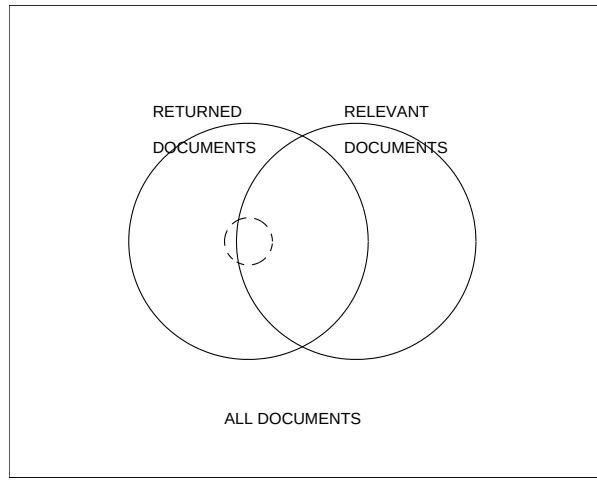


Figure 12.2. *Returned and relevant documents for two values of the tolerance: The dashed circle represents the retrieved documents for a high value of the cosine tolerance.*

performance of a certain method for information retrieval.

Example 12.6 We did query matching for query Q9 in the Medline collection (stemmed) using the cosine measure, and obtained recall and precision as illustrated in Figure 12.3. In the comparison of different methods it is more illustrative to draw

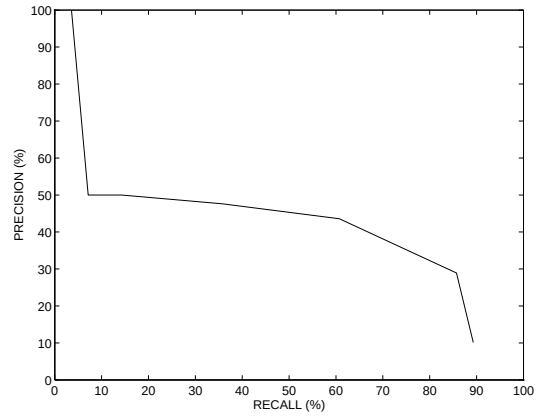


Figure 12.3. *Query matching for Q9 using the vector space method. Recall versus precision.*

the recall versus precision diagram. Ideally a method has high recall at the same

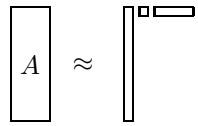
time as the precision is high. Thus, the closer the curve is to the upper right corner, the better the method.

In this example and the following examples the matrix elements were computed using term frequency and inverse document frequency weighting (12.1). ∎

12.4 Latent Semantic Indexing

Latent Semantic Indexing[22] (LSI) [24, 8] "is based on the assumption that there is some underlying latent semantic structure in the data ... that is corrupted by the wide variety of words used ..." (quoted from [67]) and that this semantic structure can be enhanced by projecting the data (the term-document matrix and the queries) onto a lower-dimensional space using the singular value decomposition.

Let $A = U\Sigma V^T$ be the SVD of the term-document matrix and approximate it by a matrix of rank k:



$$= U_k(\Sigma_k V_k^T) =: U_k D_k.$$

The columns of U_k live in the document space and are an orthogonal basis that we use to approximate the documents: Write D_k in terms of its column vectors, $D_k = (d_1, d_2, \ldots, d_n)$. From $A \approx U_k D_k$ we have $a_j \approx U_k d_j$, which means that column j of D_k holds the coordinates of document j in terms of the orthogonal basis. With this rank-k approximation the term-document matrix is represented by $A_k = U_k D_k$ and in query matching we compute $q^T A_k = q^T U_k D_k = (U_k^T q)^T D_k$. Thus, we compute the coordinates of the query in terms of the new document basis and compute the cosines from

$$\cos\theta_j = \frac{q_k^T d_j}{\|q_k\|_2 \, \|d_j\|_2}, \qquad q_k = U_k^T q. \tag{12.2}$$

This means that the query-matching is performed in a k-dimensional space.

Example 12.7 We did query matching for Q9 in the Medline collection, approximating the matrix using the truncated SVD of rank 100. The recall-precision curve is given in Figure 12.4. It is seen that for this query LSI improves the retrieval performance. In Figure 12.5 we also demonstrate a fact that is common to many term-document matrices: It is rather well-conditioned, and there is no gap in the sequence of singular values. Therefore, we cannot find a suitable rank of the LSI approximation by inspecting the singular values, it must be determined by retrieval experiments.

Another remarkable fact is that with $k = 100$ the approximation error in the matrix approximation,

$$\frac{\| A - A_k \|_F}{\| A \|_F} \approx 0.8,$$

[22] Sometimes also called Latent Semantic Analysis (LSA) [49].

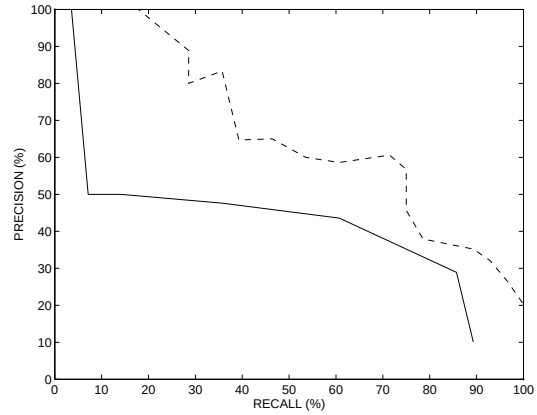


Figure 12.4. *Query matching for Q9. Recall versus precision for the full vector space model (solid line) and the rank 100 approximation (dashed).*

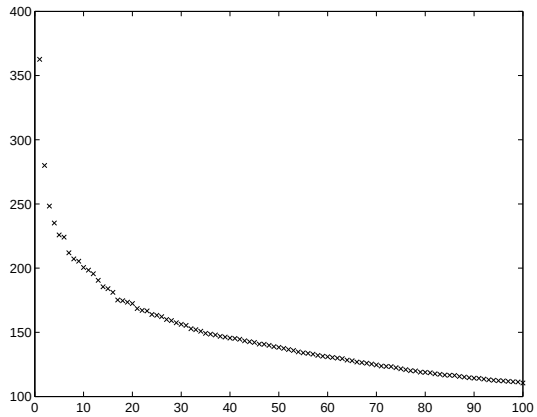


Figure 12.5. *First 100 singular values of the Medline (stemmed) matrix.*

is large, and we still get *improved retrieval performance.* In view of the large approximation error in the truncated SVD approximation of the term-document matrix, one may question whether the "optimal" singular vectors constitute the best basis for representing the term-document matrix. On the other hand, since we get such good results, perhaps a more natural conclusion may be that the Frobenius norm is not a good measure of the information contents in the term-document matrix.

It is also interesting to see what are the most important "directions" in the data. From Theorem 6.6 we know that the first few left singular vectors are the dominant directions in the document space, and their largest components should indi-

cate what these directions are. The Matlab statements `find(abs(U(:,k))>0.13)`, combined with look-up in the dictionary of terms, gave the following results for `k=1,2`:

`U(:,1)`	`U(:,2)`
cell	case
growth	cell
hormon	children
patient	defect
	dna
	growth
	patient
	ventricular

■

It should be said that LSI does not give significantly better results for all queries in the Medline collection: there are some where it gives results comparable to the full vector model, and some where it gives worse performance. However, it is often the average performance that matters.

In [49] a systematic study of different aspects of LSI is done. It is shown that LSI improves retrieval performance for surprisingly small values of the reduced rank k. At the same time the relative matrix approximation errors are large. It is probably not possible to prove any general results that explain in what way and for which data LSI can improve retrieval performance. Instead we give an artificial example (constructed using similar ideas as a corresponding example in [11]) that gives a partial explanation.

Example 12.8 Consider the term-document matrix from Example 1.1, and the query "**ranking** of **web pages**". Obviously, Documents 1-4 are relevant with respect to the query, while Document 5 is totally irrelevant. However, we obtain the following cosines for query and the original data

$$\begin{pmatrix} 0 & 0.6667 & 0.7746 & 0.3333 & 0.3333 \end{pmatrix}.$$

We then compute the SVD of the term-document matrix, and use a rank 2 approximation. After projection to the two-dimensional subspace the cosines, computed according to (12.2), are

$$\begin{pmatrix} 0.7857 & 0.8332 & 0.9670 & 0.4873 & 0.1819 \end{pmatrix}$$

It turns out that Document 1, which was deemed totally irrelevant for the query in the original representation, is now highly relevant. In addition, the scores for the relevant Documents 2-4 have been reinforced. At the same time, the score for Document 5 has been significantly reduced. Thus, in this artificial example, the dimension reduction enhanced the retrieval performance.

In Figure 12.6 we plot the five documents and the query in the coordinate system of the first two left singular vectors. Obviously, in this representation, the

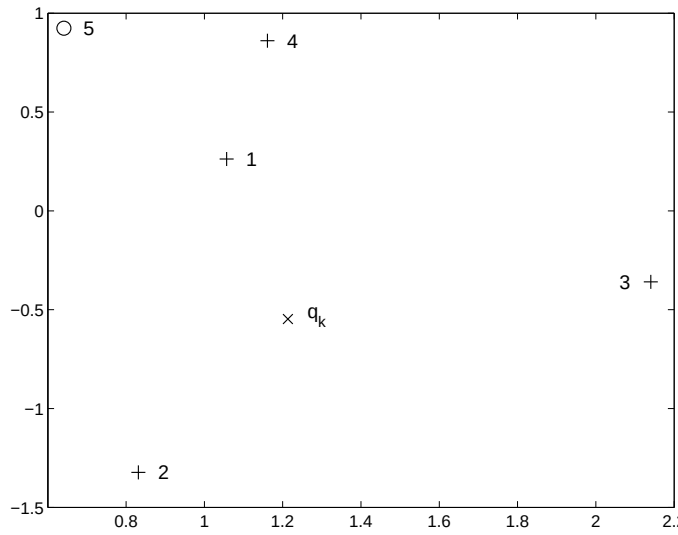


Figure 12.6. *The documents and the query projected to the coordinate system of the first two left singular vectors.*

first document is is closer to the query than document 5. The first two left singular vectors are

$$u_1 = \begin{pmatrix} 0.1425 \\ 0.0787 \\ 0.0787 \\ 0.3924 \\ 0.1297 \\ 0.1020 \\ 0.5348 \\ 0.3647 \\ 0.4838 \\ 0.3647 \end{pmatrix}, \qquad \begin{pmatrix} 0.2430 \\ 0.2607 \\ 0.2607 \\ -0.0274 \\ 0.0740 \\ -0.3735 \\ 0.2156 \\ -0.4749 \\ 0.4023 \\ -0.4749 \end{pmatrix},$$

and the singular values are $\Sigma = \mathrm{diag}(2.8546, 1.8823, 1.7321, 1.2603, 0.8483)$. The first four columns in A are strongly coupled via the words *Google*, *matrix*, etc., and those words are the dominating contents of the document collection (cf. the singular values). This shows in the composition of u_1. So even if none of the words in the query is matched by document 1, that document is so strongly correlated to the the dominating direction that it becomes relevant in the reduced representation. ∎

12.5 Clustering for Information Retrieval

In the case of document data-bases it is natural to assume that there are groups of documents with a similar contents. If we think of the documents as points in $\mathbb{R}^m$, we may be able to visualize the groups as clusters in $\mathbb{R}^m$. Representing each cluster

by its mean value, the *centroid*[23], , we obtain a compression of the data in terms of the centroids. Thus clustering is another method for low-rank approximation. The application of clustering to information retrieval is described in [67, 26, 68].

12.5.1 Matrix Approximation and Query Matching

In analogy to LSI, the matrix $C_k \in \mathbb{R}^{m\times k}$ of (normalized but not orthogonal) centroids can be used as an approximate basis in the "document space". For query matching we then need to determine the coordinates of all the documents in this basis. This can be made by solving the matrix least squares problem

$$\min_{\hat{G}_k} \| A - C_k\hat{G}_k \|_F.$$

However, it is more convenient first to orthogonalize the columns of C, i.e. compute its QR decomposition:

$$C_k = P_k R, \qquad P_k \in \mathbb{R}^{m\times k}, \qquad R \in \mathbb{R}^{k\times k},$$

and solve

$$\min_{G_k} \| A - P_k G_k \|_F. \tag{12.3}$$

Writing each column of $A - P_k G_k$ separately, we see that this is equivalent to solving the n independent standard least squares problems

$$\min_{g_j} \| a_j - P_k g_j \|_2, \qquad j = 1, \ldots, n,$$

where g_j is column j in G_k. Since P_k has orthonormal columns, we get $g_j = P_k^T a_j$, and the solution of (12.3) becomes

$$G_k = P_k^T A.$$

For matching of a query q we compute the product

$$q^T A \approx q^T P_k G_k = (P_k^T q)^T G_k = q_k^T G_k,$$

where $q_k = P_k^T$. Thus, the cosines in the low-dimensional approximation are

$$\frac{q_k^T g_j}{\| q_k \|_2 \| g_j \|_2}.$$

Example 12.9 We did query matching for Q9 in the Medline collection. Before computing the clustering we normalized the columns to equal Euclidean length. We approximated the matrix using the orthonormalized centroids from a clustering into 50 clusters. The recall-precision diagram is given in Figure 12.7. We see that for high values of recall, the centroid method is as good as the LSI method with double the rank, see Figure 12.4.

[23] Closely related to the *concept vector*, see below and [26].

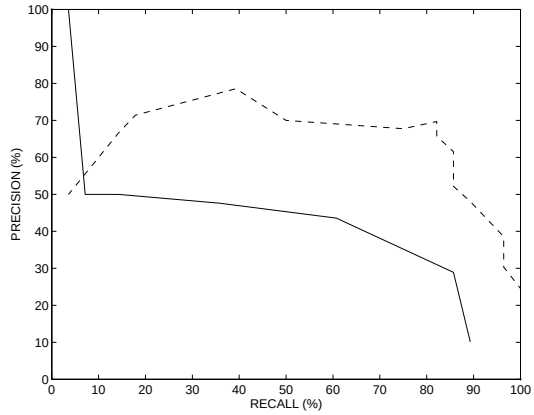


Figure 12.7. *Query matching for Q9. Recall versus precision for the full vector space model (solid line) and the rank 50 centroid approximation (dashed).*

For rank 50 the approximation error in the centroid method,

$$\| A - CD \|_F / \| A \|_F \approx 0.9,$$

is even higher than for LSI of rank 100.

The improved performance can be explained in a similar way as for LSI. Being the "average document" of a cluster, the centroid captures the main links between the dominant documents in the cluster. By expressing all documents in terms of the centroids, the dominant links are emphasized. ∎

When we tested all the 30 queries in the Medline collection, we found that the centroid method with rank equal to 50 has a similar performance as LSI with rank 100: There are some queries where the full vector space model is considerably better, but there are also some for which the centroid method is much better.

12.6 Lanczos Bidiagonalization

There will be a description of the application of bidiagonalization to textmining, based on Chapter 8 and the work of Blom and Ruhe [15].

12.7 Linear Discriminant Analysis – GSVD

Theorem 12.10 (Generalized Singular Value Decomposition [66]). *For any two matrices $K_A \in \mathbb{R}^{n\times m}$ and $K_B \in \mathbb{R}^{p\times m}$ with the same number of columns, there exist orthogonal matrices $U \in \mathbb{R}^{n\times n}$, $V \in \mathbb{R}^{p\times p}$, $W \in \mathbb{R}^{t\times t}$, and $Q \in \mathbb{R}^{m\times m}$ such that*

$$U^T K_A Q = \Sigma_A(\underbrace{W^T R}_{t}, \underbrace{0}_{m-t}) \quad amd \quad V^T K_B Q = \Sigma_B(\underbrace{W^T R}_{t}, \underbrace{0}_{m-t}),$$

where

$$K = \begin{pmatrix} K_A \\ K_B \end{pmatrix} \quad \textit{and} \quad t = rank(K),$$

$$\Sigma_A = \begin{pmatrix} I_A & & \\ & D_A & \\ & & O_A \end{pmatrix}, \quad \Sigma_B = \begin{pmatrix} O_B & & \\ & D_B & \\ & & I_B \end{pmatrix},$$

and $R \in \mathbb{R}^{t\times t}$ *is nonsingular with its singular values equal to the nonzero singular values of* K*. The matrices*

$$I_A \in \mathbb{R}^{r\times r} \quad \textit{and} \quad I_B \in \mathbb{R}^{(t-r-s)\times(t-r-s)}$$

are identity matrices, where

$$r = rank\begin{pmatrix} K_A \\ K_B \end{pmatrix} - rank(K_B) \quad \textit{and} \quad s = rank(K_A) + rank(K_B) - rank\begin{pmatrix} K_A \\ K_B \end{pmatrix},$$

$$O_A \in \mathbb{R}^{(n-r-s)\times(t-r-s)} \quad \textit{and} \quad O_B \in \mathbb{R}^{(p-t+r)\times r}$$

are zero matrices with possibly no rows or no columns, and

$$D_A = diag(\alpha_{r+1}, \ldots, \alpha_{r+s}) \quad \textit{and} \quad D_B = diag(\beta_{r+1}, \ldots, \beta_{r+s})$$

satisfy

$$1 > \alpha_{r+1} \geq \cdots \geq \alpha_{r+s} > 0, \quad 0 < \beta_{r+1} \leq \cdots \leq \beta_{r+s} < 1, \tag{12.4}$$

and $\alpha_i^2 + \beta_i^2 = 1$ *for* $i = r+1, \ldots, r+s$.

Exercises

Chapter 13

Page Ranking for a Web Search Engine

When a search is made on the Internet using a search engine there is first a traditional text processing part, where the aim is to find all the web pages containing the words of the query. Due to the massive size of the Web, the number of hits is likely to be much too large to be handled by the user. Therefore, some measure of quality is needed to filter away pages that are likely to be less interesting.

When one uses a web search engine it is typical that the search phrase is under-specified.

Example 13.1 A Google[24] search conducted on September 29, 2005, using the search phrase *university*, gave as a result links to the following well-known universities: *Harvard, Stanford, Cambridge, Yale, Cornell, Oxford*. The total number of web pages relevant to the search phrase was more than 2 billions. ∎

Obviously Google uses an algorithm for ranking all the web pages that agrees rather well with a common-sense quality measure. Somewhat surprisingly, the ranking procedure is not based on human judgement, but on the link structure of the web. Loosely speaking, Google assign a high rank to a web page, if it has inlinks from other pages that have a high rank. We will see that this "self-referencing" statement can be formulated mathematically as an eigenvalue equation for a certain matrix.

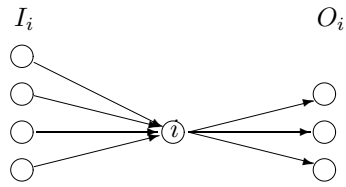
13.1 Pagerank

It is of course impossible to define a generally valid measure of relevance that would be acceptable for a majority of users of a search engine. Google uses the concept of *pagerank* as a quality measure of web pages. It is based on the assumption that the number of links to and from a page give information about the importance of a page. We will give a description of pagerank based primarily on [65] and [29]. Concerning Google, see [17].

Let all web pages be ordered from 1 to n, and let i be a particular web page. Then O_i will denote the set of pages that i is linked to, the *outlinks*. The number of outlinks is denoted $N_i = |O_i|$. The set of *inlinks*, denoted I_i, are the pages that

[24] `http://www.google.com/`

have an outlink to i.



In general, a page i can be considered as more important the more inlinks it has. However, a ranking system based only on the number of inlinks is easy to manipulate[25]: When you design a web page i that (e.g. for commercial reasons) you would like to be seen by as many as possible, you could simply create a large number of (information-less and unimportant) pages that have outlinks to i. In order to discourage this, one defines the rank of i in such a way that if a highly ranked page j, has an outlink to i, this adds to the importance of i in the following way: the rank of page i is a weighted sum of the ranks of the pages that have outlinks to i. The weighting is such that the rank of a page j is divided evenly among its outlinks. The preliminary definition of Pagerank is

$$r_i = \sum_{j \in I_i} \frac{r_j}{N_j}. \tag{13.1}$$

This definition is recursive, so pageranks cannot be computed directly. Instead a fixed point iteration might be used: Guess an initial ranking vector r^0. Then iterate

$$r_i^{(k+1)} = \sum_{j \in I_i} \frac{r_j^{(k)}}{N_j}, \quad k = 0, 1, \ldots \tag{13.2}$$

There are a few problems with this iteration: if a page has no outlinks, then in the iteration process it only accumulates rank via its inlinks, but this rank is never distributed further. Therefore it is not clear if the iteration converges. We will come back to this question later.

More insight is gained if we reformulate (13.1) as an eigenvalue problem for a matrix representing the graph of the Internet. Let Q be a square matrix of dimension n. Then define

$$Q_{ij} = \begin{cases} 1/N_j & \text{if there is a link from } j \text{ to } i \\ 0 & \text{otherwise.} \end{cases}$$

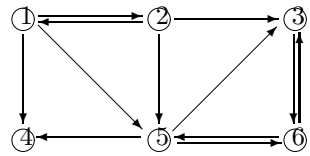
This definition means that row i has nonzero elements in those positions that correspond to inlinks of i. Similarly, column j has nonzero elements equal to N_j in those positions that correspond to the outlinks of j, and, provided that the page has

[25] For an example of attempts to fool a search engine, see [83].

outlinks, the sum of all the elements in column j is equal to one. In the following symbolic picture of the matrix Q, non-zero elements are denoted $*$:

$$
\begin{array}{ccc}
 & j & \\
 & \downarrow & \\
i \rightarrow & \begin{pmatrix} & & * & & \\ 0 & * & 0 & * & * \\ & & \vdots & & \\ & & * & & \\ & & 0 & & \\ & & * & & \end{pmatrix} & \leftarrow \text{ inlinks} \\
 & \uparrow & \\
 & \text{outlinks} &
\end{array}
$$

Example 13.2 The following link graph illustrates a set of web pages with outlinks and inlinks.



The corresponding matrix becomes

$$
Q = \begin{pmatrix}
0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\
\frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{2} \\
\frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 \\
\frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{2} \\
0 & 0 & 1 & 0 & \frac{1}{3} & 0
\end{pmatrix}.
$$

Since page 4 has no outlinks, the corresponding column is equal to zero. ∎

Obviously, the definition (13.1) is equivalent to the scalar product of row i and the vector r, which holds the ranks of all pages. We can write the equation in matrix form:

$$\lambda r = Qr, \qquad \lambda = 1, \tag{13.3}$$

i.e. r is an *eigenvector* of Q with *eigenvalue* $\lambda = 1$. It is now easily seen that the iteration (13.2) is equivalent to

$$r^{(k+1)} = Qr^{(k)}, \qquad k = 0, 1, \ldots,$$

which is the *power method* for computing the eigenvector. However, at this point it is not clear that pagerank is well-defined, as we do not know if there exists an eigenvalue equal to 1. It turns out that the theory of Markov chains is useful in the analysis.

13.2 Random Walk and Markov Chains

There is a random walk interpretation of the pagerank concept. Assume that a surfer visiting a web page, chooses the next page among the outlinks with equal probability. Then the random walk induces a Markov chain, see e.g. [63, 56].

> *A Markov chain is a random process, where the next state is determined completely from the present state; the process has no memory [63].*

The transition matrix of the Markov chain is Q^T. (Note that we use a slightly different notation than is common in the theory of stochastic processes).

The random surfer should never get stuck. In other words, our random walk model should have no pages without outlinks (such a page corresponds to a zero column in Q). Therefore the model is modified so that zero columns are replaced by a constant value in all positions. This means that there is equal probability to go to any other page in the net. Define the vectors

$$d_j = \begin{cases} 1 & \text{if } N_j = 0 \\ 0 & \text{otherwise,} \end{cases}$$

for $j = 1, \ldots, n$, and

$$e = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^n. \tag{13.4}$$

The modified matrix is defined

$$P = Q + \frac{1}{n} e d^T. \tag{13.5}$$

With this modification the matrix P is a proper *column-stochastic matrix*: It has non-negative elements and the elements of each column sum up to 1. The preceding statement can be reformulated as follows.

Proposition 13.3. *A column-stochastic matrix P satisfies*

$$e^T P = e^T, \tag{13.6}$$

where e is defined by (13.4).

Example 13.4 The matrix in the previous example is modified to

$$P = \begin{pmatrix} 0 & \frac{1}{3} & 0 & \frac{1}{6} & 0 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{6} & 0 & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{6} & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & \frac{1}{6} & \frac{1}{3} & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{6} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & \frac{1}{6} & \frac{1}{3} & 0 \end{pmatrix}.$$

∎

Now, in analogy to (13.3), we would like to define the pagerank vector as a unique eigenvector of P with eigenvalue 1,

$$Pr = r.$$

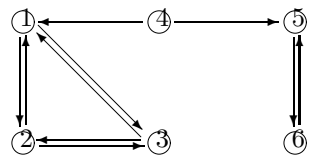
However, the existence of such a unique eigenvalue is still not guaranteed. The eigenvector of the transition matrix corresponds to a stationary probability distribution for the Markov chain: The element in position i, r_i, is the probability that after a large number of steps, the random walker is at web page i. To ensure the existence of a unique distribution, the matrix must be *irreducible*, cf. [50].

Definition 13.5. *A square matrix A is called* reducible *if there is a permutation matrix P such that*

$$PAP^T = \begin{pmatrix} X & Y \\ 0 & Z \end{pmatrix}, \tag{13.7}$$

where X and Z are both square. Otherwise the matrix is called irreducible.*,*

Example 13.6 To illustrate the concept of reducibility, we give an example of a link graph that corresponds to a *reducible* matrix.



A random walker who has entered the left part of the link graph will never get out of it, and similarly he will get stuck in the right part. The corresponding matrix is

$$P = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \tag{13.8}$$

which is of the form (13.7). Actually, this matrix has two eigenvalues equal to 1, and one equal to -1, see Example 13.10 below.

The directed graph corresponding to an irreducible matrix is *strongly connected*: given any two nodes (N_i, N_j), in the graph, there exists a path leading from N_i to N_j. ∎

The uniqueness of the largest eigenvalue of an irreducible matrix is guaranteed by the *Perron-Frobenius theorem*; we state it for the special case treated here. The inequality $A > 0$ is understood as all the elements of A being strictly positive.

Theorem 13.7. *Let A be an irreducible column-stochastic matrix. The largest eigenvalue in magnitude is equal to 1. There is a unique corresponding eigenvector r satisfying $r > 0$, and $\|r\|_1 = 1$; this is the only eigenvector that is non-negative. If $A > 0$, then $|\lambda_i| < 1$, $i = 2, 3, \ldots, n$.*

Proof. Due to the fact that A is column-stochastic we have $e^T A = e^T$, which means that 1 is an eigenvalue of A. The rest of the statement can be proved using Perron-Frobenius theory [63, Chapter 8]. □

Given the size of the Internet, we can be sure that the link matrix P is reducible, which means that the pagerank eigenvector of P is not well-defined. To ensure irreducibility, i.e. to make it impossible for the random walker to get trapped in a subgraph, one adds, artificially, a link from every web page to all the other. In matrix terms, this can be made by taking a convex combination of P and a rank one matrix,

$$A = \alpha P + (1 - \alpha)\frac{1}{n} e e^T, \tag{13.9}$$

for some α satisfying $0 \le \alpha \le 1$. It is easy to see that the matrix A is column-stochastic:

$$e^T A = \alpha e^T P + (1 - \alpha)\frac{1}{n} e^T e e^T = \alpha e^T + (1 - \alpha) e^T = e^T.$$

The random walk interpretation of the additional rank one term is that in each time step the surfer visiting a page will jump to a random page with probability $1 - \alpha$ (sometimes referred to as *teleportation*).

We now see that the pagerank vector for the matrix A is well-defined.

Proposition 13.8. *The column-stochastic matrix A defined in* (13.9) *is irreducible (since $A > 0$) and has the largest in magnitude eigenvalue $\lambda = 1$. The corresponding eigenvector r satisfies $r > 0$.*

For the convergence of the numerical eigenvalue algorithm, it is essential to know, how the eigenvalues of P are changed by the rank one modification (13.9).

Theorem 13.9 ([56]). *Assume that the eigenvalues of the column-stochastic matrix P are $\{1, \lambda_2, \lambda_3 \ldots, \lambda_n\}$. Then the eigenvalues of $A = \alpha P + (1 - \alpha)\frac{1}{n} e e^T$ are $\{1, \alpha\lambda_2, \alpha\lambda_3, \ldots, \alpha\lambda_n\}$.*

Proof. Define $\hat{e}$ to be e normalized to Euclidean length 1, and let $U_1 \in \mathbb{R}^{n \times (n-1)}$ be such that $U = \begin{pmatrix} \hat{e} & U_1 \end{pmatrix}$ is orthogonal. Then, since $\hat{e}^T P = \hat{e}^T$,

$$\begin{aligned} U^T P U &= \begin{pmatrix} \hat{e}^T P \\ U_1^T P \end{pmatrix} \begin{pmatrix} \hat{e} & U_1 \end{pmatrix} = \begin{pmatrix} \hat{e}^T \\ U_1^T P \end{pmatrix} \begin{pmatrix} \hat{e} & U_1 \end{pmatrix} \\ &= \begin{pmatrix} \hat{e}^T \hat{e} & \hat{e}^T U_1 \\ U_1^T P \hat{e} & U_1^T P^T U_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ w & T \end{pmatrix}, \end{aligned} \tag{13.10}$$

where $w = U_1^T P\hat{e}$, and $T = U_1^T P^T U_1$. Since we have made a similarity transformation, the matrix T has the eigenvalues $\lambda_2, \lambda_3, \ldots, \lambda_n$. We further have

$$U^T v = \begin{pmatrix} 1/\sqrt{n}\, e^T v \\ U_1^T v \end{pmatrix} = \begin{pmatrix} 1/\sqrt{n} \\ U_1^T v \end{pmatrix}.$$

Therefore,

$$U^T A U = U^T(\alpha P + (1-\alpha)ve^T)U = \alpha \begin{pmatrix} 1 & 0 \\ w & T \end{pmatrix} + (1-\alpha)\begin{pmatrix} 1/\sqrt{n} \\ U_1^T v \end{pmatrix}\begin{pmatrix} \sqrt{n} & 0 \end{pmatrix}$$

$$= \alpha \begin{pmatrix} 1 & 0 \\ w & T \end{pmatrix} + (1-\alpha)\begin{pmatrix} 1 & 0 \\ \sqrt{n}\, U_1^T v & 0 \end{pmatrix} =: \begin{pmatrix} 1 & 0 \\ w_1 & \alpha T \end{pmatrix}.$$

The statement now follows immediately. □

Theorem 13.9 implies that even if P has a multiple eigenvalue equal to 1, which is actually the case for the Google matrix, the second largest eigenvalue in magnitude of A is always equal to α.

Example 13.10 We compute the eigenvalues and eigenvectors of the matrix $A = \alpha P + (1-\alpha)\frac{1}{n}ee^T$, with P from (13.8) and $\alpha = 0.85$. The Matlab code

```
LP=eig(P)';
e=ones(6,1);
A=0.85*P + 0.15/6*e*e';
[R,L]=eig(A)
```

gives the following result:

```
LP = -0.5    1.0   -0.5    1.0   -1.0    0

R = 0.447   -0.365   -0.354    0.000    0.817    0.101
    0.430   -0.365    0.354   -0.000   -0.408   -0.752
    0.430   -0.365    0.354    0.000   -0.408    0.651
    0.057   -0.000   -0.707    0.000    0.000   -0.000
    0.469    0.548   -0.000   -0.707    0.000    0.000
    0.456    0.548    0.354    0.707   -0.000   -0.000

diag(L) = 1.0 0.85 -0.0 -0.85 -0.425 -0.425
```

It is seen that all other eigenvectors except the first one (which corresponds to the eigenvalue 1), have both positive and negative components, as stated in Theorem 13.7. ■

Instead of the modification (13.9) we can define

$$A = \alpha P + (1-\alpha)ve^T,$$

where v is a non-negative vector with $\| v \|_1 = 1$ that can be chosen in order to make the search biased towards certain kinds of web pages. Therefore, it is often referred to as a *personalization vector* [65, 44]. The vector v can also be used for avoiding manipulation by so called link farms [56].

13.3 The Power Method for Pagerank Computation

We want to solve the eigenvalue problem

$$Ar = r,$$

where r is normalized $\| r \|_1 = 1$. In this section we denote the sought eigenvector by r_1. In dealing with stochastic matrices and vectors that are probability distributions, it is natural to use the 1-norm for vectors (Section 2.3). Due to the sparsity and the dimension of A (estimated to be of the order billions), it is out of the question to compute the eigenvector using any of the standard methods for dense matrices that are based on applying orthogonal transformations to the matrix described in Chapter 16. The only viable method so far is the *power method*.

Assume that an initial approximation $r^{(0)}$ is given. The power method is given in the following algorithm.

The power method for $Ar = \lambda r$

---

for $k = 1, 2, \ldots$ until convergence

$$q^{(k)} = Ar^{(k-1)}$$
$$r^{(k)} = q^{(k)} / \| q^{(k)} \|_1$$

The purpose of normalizing the vector (making it have 1-norm equal to 1) is to avoid that the vector becomes either very large of very small, and thus unrepresentable in the floating point system. We will see later that normalization is not necessary in the pagerank computation.

In this context there is no need to compute an eigenvalue approximation, as the eigenvalue sought for is known to be equal to one.

The convergence of the power method depends on the distribution of eigenvalues. In order to make the presentation simpler, we assume that A is diagonalizable, i.e. there exists a nonsingular matrix T of eigenvectors, $T^{-1}AT = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. The eigenvalues λ_i are ordered $1 = \lambda_1 > |\lambda_2| \geq \cdots \geq |\lambda_n|$. Expand the initial approximation $r^{(0)}$ in terms of the eigenvectors,

$$r^{(0)} = c_1 t_1 + c_2 t_2 + \cdots + c_n t_n,$$

where $c_1 \neq 0$ is assumed[26], and $r = t_1$ is the sought eigenvector. Then we have

$$\begin{aligned} A^k r^{(0)} &= c_1 A^k t_1 + c_2 A^k t_2 + \cdots + c_n A^k t_n \\ &= c_1 \lambda^k t_1 + c_2 \lambda^k t_2 + \cdots + c_n \lambda^k t_n = \lambda_1^k \left(c_1 t_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1} \right)^k t_j \right). \end{aligned}$$

[26] This assumption can be expected to be satisfied in floating point arithmetic, if not at the first iteration, so after the second, due to round off.

Obviously, since for $j = 2, 3, \ldots$, we have $|\lambda_j| < 1$, the second term tends to zero and the power method converges to the eigenvector $r = t_1$. The rate of convergence is determined by the ratio $|\lambda_2/\lambda_1|$. If this ratio is close to 1, then the iteration is very slow. Fortunately this is not the case for the Google matrix, see Theorem 13.9 and below.

A stopping criterion for the power iteration can be formulated in terms of the residual vector for the eigenvalue problem. Let $\hat{\lambda}$ be the computed approximation of the eigenvalue, and $\hat{r}$ the corresponding approximate eigenvector. Then it can be shown [81], [4, p. 229] that the optimal error matrix E, for which

$$(A + E)\hat{r} = \hat{\lambda}\hat{r},$$

exactly, satisfies

$$\| E \|_2 = \| s \|_2,$$

where $s = A\hat{r} - \hat{\lambda}\hat{r}$. This means that if the residual $\|s\|_2$ is small, then the computed approximate eigenvector $\hat{r}$ is the exact eigenvector of a matrix $A + E$ that is close to A. In the case of pagerank computations it is natural to use the 1-norm instead [52], which does not make much difference, since the norms are equivalent (2.5).

In view of the huge dimension of the Google matrix, it is non-trivial to compute the matrix-vector product $y = Az$, where $A = \alpha P + (1 - \alpha)\frac{1}{n}ee^T$. First, we show that normalization of the vectors produced in the power iteration is unnecessary.

Proposition 13.11. *Assume that the vector z satisfies $\| z \|_1 = e^T z = 1$, and that the matrix A is is column-stochastic. Then*

$$\| Az \|_1 = 1. \tag{13.11}$$

Proof. Put $y = Az$. Then

$$\| y \|_1 = e^T y = e^T Az = e^T z = 1,$$

since A is column-stochastic ($e^T A = e^T$). ∎

Then recall that P was constructed from the actual link matrix Q as

$$P = Q + \frac{1}{n}ed^T,$$

where the row vector d has an element 1 in all those positions that correspond to web pages with no outlinks, see (13.5). This means that to form P, we insert a large number of full vectors in Q, each of the same dimension as the total number of web pages. Consequently, we cannot afford to store P explicitly. Let us look at the multiplication $y = Az$ in some more detail:

$$y = \alpha(Q + \frac{1}{n}ed^T)z + \frac{(1 - \alpha)}{n}e(e^T z) = \alpha Qz + \beta\frac{1}{n}\,e, \tag{13.12}$$

where

$$\beta = \alpha d^T z + (1 - \alpha) e^T z.$$

However, we do not need to compute β from this equation. Instead we can use (13.11) in combination with (13.12):

$$1 = e^T(\alpha Q z) + \beta e^T(\frac{1}{n}\, e) = e^T(\alpha Q z) + \beta.$$

Thus, we have $\beta = 1 - \| \alpha Q z \|_1$. An extra bonus is that we do not use the vector d at all, i.e., we need not know which pages lack outlinks.

The following Matlab code implements the matrix vector multiplication.

```
yhat=alpha*Q*z;
beta=1-norm(yhat,1);
y=yhat+beta*v;
residual=norm(y-z,1);
```

Here $v = (1/n)\, e$ or a personalized teleportation vector, see p. 143. In order to save memory, we even should avoid using the extra vector `yhat` and replace it by `y`.

From Theorem 13.9 we know that the second eigenvalue of the Google matrix satisfies $\lambda_2 = \alpha$. A typical value of α is 0.85. Approximately $k = 57$ iterations are needed to make the factor 0.85^k equal to 10^{-4}. This is reported [56] to be close the number of iterations used by Google.

Example 13.12 As an example we used the matrix P obtained from the domain `stanford.edu`[27]. The number of pages is 281903, and the total number of links is 2312497. Part of the matrix is displayed in Figure 13.1. We computed the page rank vector using the power method with $\alpha = 0.85$ and iterated 63 times until the 1-norm of the residual was smaller than 10^{-6}. The residual and the final pagerank vector are illustrated in Figure 13.2. ∎

If one computes the pagerank for a subset of the Internet, one particular domain, say, then the matrix P may be of a dimension for which one can use other methods than the power method. In such cases it may be sufficient to use the Matlab function `eigs`, which computes a small number of eigenvalues and the corresponding eigenvectors of a sparse matrix using a variant of the Arnoldi (or Lanczos) method, cf. Section 16.8. In view of the fact that one pagerank calculation can take several days, several enhancements of the iteration procedure have been proposed. In [50] an adaptive method is described that checks the convergence of the components of the pagerank vector and avoids performing the power iteration for those components. Up to 30 % speed up has been reported. The block structure of the web is used in [51], and speedups of a factor 2 have been reported. An acceleration method based on Aitken extrapolation is described in [52]. Aggregation methods are discussed in several papers by Langville and Meyer and in [48], and the Arnoldi method in [37].

A variant of Pagerank is proposed in [40]. Further properties of the Pagerank matrix are given in [73].

[27] `http://www.stanford.edu/~sdkamvar/research.html`

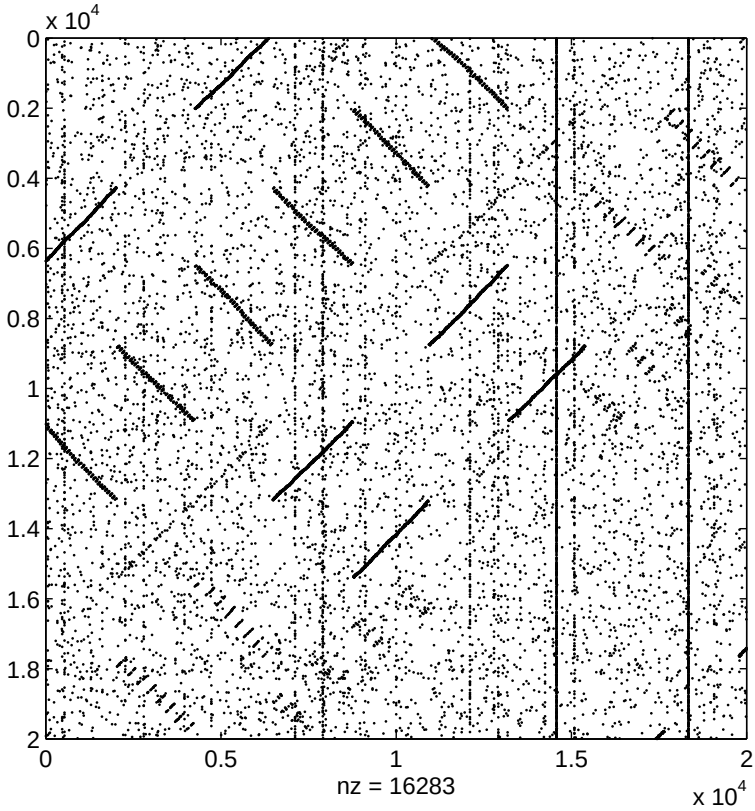


Figure 13.1. *A* 20000×20000 *submatrix of the* `stanford.edu` *matrix.*

13.4 HITS

Another method based on the link structure of the web was introduced at the same time as Pagerank [53]. It is called HITS (Hypertext Induced Topic Search), and is based on the concepts of *authorities* and *hubs*. An authority is a web page with several inlinks and a hub has several outlinks. The basic idea is: *good hubs point to good authorities and good authorities are pointed to by good hubs.* Each web page is assigned both a hub score y and an authority score x.

Let L be the adjacency matrix of the directed web graph. Then two equations are given that mathematically define the relation between the two scores, based on the basic idea:

$$x = L^T y, \qquad y = Lx. \tag{13.13}$$

The algorithm for computing the scores is the power method, which converges to the left and right singular vectors corresponding to the largest singular value of L. In the implementation of HITS it is not the adjacency matrix of the whole web that is used, but of all the pages relevant to the query.

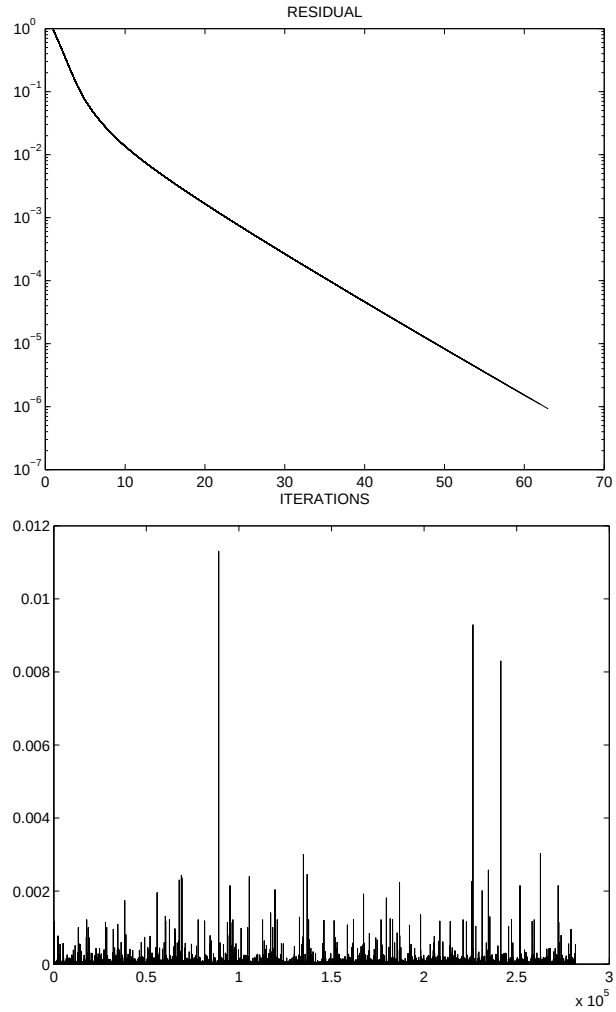


Figure 13.2. *The residual in the power iterations (top) and the pagerank vector (bottom) for the* `stanford.edu` *matrix.*

There is now an extensive literature on Pagerank, HITS and other ranking methods. For overviews, see [54, 55, 7]. A combination of HITS and Pagerank has been proposed in [59].

Obviously the ideas underlying Pagerank and HITS are not restricted to web applications, but can be applied to other network analyses. For instance, a variant of the HITS method was recently used in a study of Supreme Court Precedent [32]. A generalization of HITS is given in [16], which also treats synonym extraction.

Exercises

13.1. Prove Theorem 13.9 for the case when $A = \alpha P + (1 - \alpha)ve^T$, where v is a personalization vector.

Chapter 14

Automatic Key Word and Key Sentence Extraction

Due to the explosion of the amount of textual information available, there is a pressing need to develop automatic procedures for text summarization. One typical situation is when a web search engines presents a small amount of text from each document that matches a certain query. Another relevant area is the summarization of news articles.

Automatic text summarization is an active research field with connections to several other research areas such as information retrieval, natural language processing, and machine learning. Informally, the goal of text summarization is to *extract content from a text document, and present the most important content to the user in a condensed form and in a manner sensitive to the user's or application's need* [60]. In this chapter we will have a considerably less ambitious goal: we will present a method for automatically extracting key words and key sentences from a text. There will be connections to the vector space model in information retrieval, and to the concept of page rank. We will also use clustering. The presentation is based on [102].

14.1 Saliency score

Consider a text, from which we want to extract key words and key sentences. As an example we will take Chapter 13 from this book. As one of the proprocessing steps, one should perform stemming so that the same word stem with different endings is represented by one token only. Stop words (cf. Chapter 12) occur frequently in texts, but since they do not distinguish between different sentences, they should be removed. Similarly, if the text carries special symbols, e.g. mathematics, or mark-up language tags (html, LaTeX), it may be necessary to remove those.

Since we want to compare word frequencies in different sentences, we must consider each sentence as a separate document (in the terminology of information retrieval). After the preprocessing has been done, we parse the text, using the same type of parser as in information retrieval. This way a term-document matrix is prepared, which in this chapter we will refer to as a *term-sentence* matrix. Thus we have a matrix $A \in \mathbb{R}^{m \times n}$, where m denotes the number of different terms, and n the number of sentences. The element a_{ij} is defined as the frequency[28] of term i in document j.

Obviously, the column vector $(a_{1j}\, a_{2j}\, \ldots a_{mj})^T$ is nonzero in the positions

[28] Naturally, a term and document weighting scheme, see [11], should be used.

corresponding to the terms occuring in document j. Similarly, the row vector $(a_{i1}\, a_{i2}\, \ldots a_{in})$ is nonzero in the positions corresponding to sentences containing term i.

The basis of the procedure in [102] is the simultaneous, but separate *ranking* of the terms and the sentences. Thus, term i is given a nonnegative *saliency score*, denoted u_i. The higher the saliency score, the more important the term. The saliency score of sentence j is denoted v_j.

The assignment of saliency scores is made based on the *mutual reinforcement principle* [102]:

> A term should have a high saliency score if it appears in many sentences with high saliency scores. A sentence should have a high saliency score if it contains many words with high saliency scores.

More precisely, we assert that the saliency score of term i is proportional to the sum of the scores of the sentences where it appears; in addition, each term is weighted by the corresponding matrix element,

$$u_i \propto \sum_{j=1}^{n} a_{ij} v_j, \qquad i = 1, 2, \ldots, m.$$

Similarly, the saliency score of sentence j is defined to be proportional to the scores of its words, weighted by the corresponding a_{ij},

$$v_j = \propto \sum_{i=1}^{m} a_{ij} u_i, \qquad j = 1, 2, \ldots, n.$$

Obviously, collecting the saliency scores in two vectors, $u \in \mathbb{R}^m$, and $v \in \mathbb{R}^n$, these two equations can be written

$$\sigma_u u = Av, \tag{14.1}$$
$$\sigma_v v = A^T u, \tag{14.2}$$

where σ_u and σ_v are proportionality constants. In fact, the constants must be equal: Inserting one equation into the other, we get

$$\sigma_u\, u = \frac{1}{\sigma_v} AA^T u,$$
$$\sigma_v\, v = \frac{1}{\sigma_u} A^T A v,$$

which shows that u and v are eigenvectors of AA^T and $A^T A$, respectively, with the same eigenvalue, cf. Exercise 6.9. It follows that u and v are singular vectors corresponding to the same singular value.

If we choose the largest singular value, then we are guaranteed that the components of u and v are non-negative (Exercise 2).

In summary, the saliency scores of the terms are defined as the components of u_1, and the saliency scores of the sentences are the components of v_1.

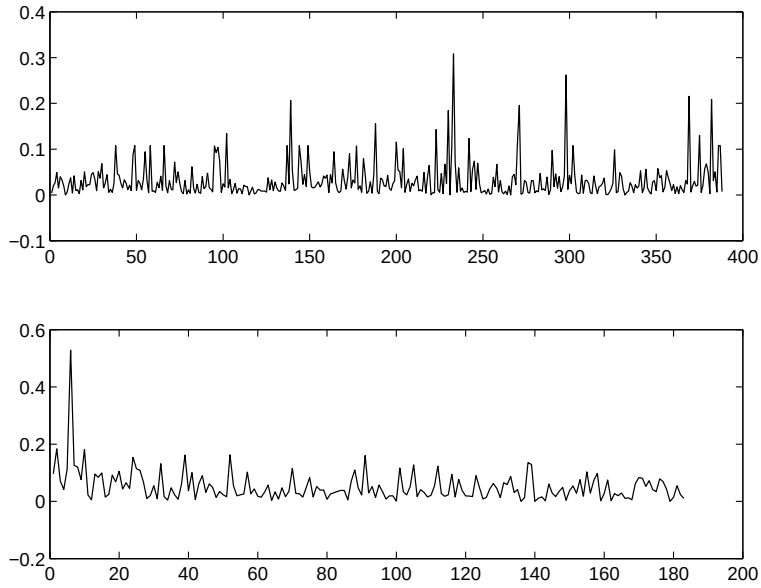


Figure 14.1. *The saliency scores for Chapter 13.*

Example 14.1 We created a term-sentence matrix based on Chapter 13. Since the text is written using LaTeX, we first had to remove all LaTeX typesetting commands. This was done using a lexical scanner called `detex`[29]. Then the text was stemmed and stop words were removed. A term-sentence matrix A was constructed using the text parser `TMG` [101]: there turned out to be 388 terms in 191 sentences. The first singular vectors were computed in Matlab, `[u,s,v]=svds(A,1)`. (The matrix is sparse, so we used the SVD function for sparse matrices.) The singular vectors are plotted in Figure 14.1.

By finding the ten largest components of u_1, and using the dictionary produced by the text parser, we found that the following words are the most important in the chapter. The words are ordered by importance.

page, search, university, web, Google, rank, outlink, link, number, equal

The six most important sentences are in order,

1. A Google (`http://www.google.com/`) search conducted on September 29, 2005, using the search phrase university, gave as a result links to the following well-known universities: Harvard, Stanford, Cambridge, Yale, Cornell, Oxford.

2. When a search is made on the Internet using a search engine there is first a traditional text processing part, where the aim is to find all the web pages containing the words of the query.

[29] `http://www.cs.purdue.edu/homes/trinkle/detex/`

3. Loosely speaking, Google assign a high rank to a web page, if it has inlinks from other pages that have a high rank.

4. Assume that a surfer visiting a web page, chooses the next page among the outlinks with equal probability.

5. Similarly, column j has nonzero elements equal to N_j in those positions that correspond to the outlinks of j, and, provided that the page has outlinks, the sum of all the elements in column j is equal to one.

6. The random walk interpretation of the additional rank one term is that in each time step the surfer visiting a page will jump to a random page with probability $1 - \alpha$ (sometimes referred to as *teleportation*).

It is apparent that this method prefers long sentences. On the other hand, these sentences a undeniably key sentences for the text. ∎

The method described above can also be thought of as a rank-1 approximation of the term-sentence matrix A, illustrated symbolically in Figure 14.2.

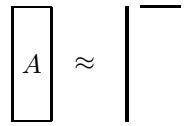


Figure 14.2. *Symbolic illustration of a rank-1 approximation:* $A \approx \sigma_1 u_1 v_1^T$.

In this interpretation the vector u_1 is a basis vector for the subspace spanned by the columns of A and the row vector $\sigma_1 v_1^T$ holds the coordinates of the columns of A in terms of this basis. Now we see that the method based on saliency scores has a drawback: if there are, say, two "top sentences" that contain the same high saliency terms. Then their coordinates will be approximately the same, and both sentences will be extracted as key sentences. This is unnecessary, since they are very similar. We will next see that this can be avoided if we base the key sentence extraction on a rank-k approximation.

14.2 Key Sentence Extraction from a Rank-k Approximation

Assume that we have computed an low rank approximation of the term-sentence matrix:

$$A \approx CD, \qquad C \in \mathbb{R}^{m \times k}, \quad D \in \mathbb{R}^{k \times n}, \tag{14.3}$$

illustrated in Figure 14.3. This approximation can be be based on the SVD or on clustering [102], spectral clustering, for instance, see Section 10.2.

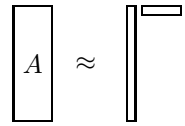


Figure 14.3. *Symbolic illustration of low rank approximation:* $A \approx CD$.

The dimension k is greater than or equal to the number of key sentences that we want to extract. C is a rank-k matrix of basis vectors, and each column of D holds the coordinates of the corresponding column in A in terms of the basis vectors.

The key observation is that *the basis vectors represent the most important directions in the "sentence space", i.e. the column space.* However, the low rank approximation does not immediately give a selection of the most important sentences. Such a selection can be found if we determine the column of A that is the "heaviest" in terms of the basis, i.e. the column in D with the largest 2-norm. This defines one new basis vector. Then we proceed by determining the column of D that is the "heaviest" in terms of the remaining $k-1$ basis vectors, and so on.

To derive the method we note that in the approximate inequality 14.3 we may introduce any nonsingular matrix T and its inverse between C and D, and we may multiply the relation with any permutation from the right, without changing the relation:

$$AP \approx CDP = (CT)(T^{-1}DP),$$

where $T \in \mathbb{R}^{k\times k}$. Now find the column of largest norm in D and permute it to the first column; at the same time move the corresponding column of A to the first position. The determine a Householder transformation Q_1 that zeros the elements in the first column below the element in position $(1,1)$, and apply the transformation both to C and D:

$$AP_1 \approx (CQ_1)(Q_1^T DP_1),$$

It is seen that this is actually the first step in the QR decomposition with column pivoting of D. Assume for illustration that $m = 6$, $n = 5$, and $k = 3$. After this step the matrices have the structure (we have also assumed that column 4 of D had almost the same coordinates as the column that was moved to the first position),

$$(CQ_1)(Q_1^T DP_1) = \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix} \begin{pmatrix} \kappa_1 & \times & \times & \times & \times \\ 0 & \times & \times & \epsilon_1 & \times \\ 0 & \times & \times & \epsilon_2 & \times \end{pmatrix},$$

where κ is the Euclidean length of the first column of DP_1. Since column 4 was similar to the one that is now in position 1, it has small entries in row 2 and 3.

Then we introduce the diagonal matrix

$$T_1 = \begin{pmatrix} \kappa_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

between the factors:

$$C_1 D_1 := (CQ_1T_1)(T_1^{-1}Q_1^T DP_1) = \begin{pmatrix} * & \times & \times \\ * & \times & \times \\ * & \times & \times \\ * & \times & \times \\ * & \times & \times \\ * & \times & \times \\ * & \times & \times \end{pmatrix} \begin{pmatrix} 1 & * & * & * & * \\ 0 & \times & \times & \epsilon_1 & \times \\ 0 & \times & \times & \epsilon_2 & \times \end{pmatrix}.$$

This changes only column 1 of the left factor and column 1 of the right factor (marked with $*$). From the relation

$$AP_1 \approx C_1 D_1,$$

we now see that the first column in AP_1 is approximately equal to the first column in C_1. Remembering that the columns of the original matrix C are the dominating directions in the matrix A, *we have now identified the "dominating column" of A.*

Next we continue the procedure and determine the second most dominating column of A. To this end we compute the norms of the columns of D_1, excluding the first row (because that row holds the coordinates in terms of the first column of C_1). The column with largest norm is moved to position 2, and reduced by a Householder transformation in a similar manner as above. After this step we have

$$C_2 D_2 := (C_1 Q_2 T_2)(T_2^{-1}Q_2^T D_1 P_2) = \begin{pmatrix} * & * & \times \\ * & * & \times \\ * & * & \times \\ * & * & \times \\ * & * & \times \\ * & * & \times \\ * & * & \times \end{pmatrix} \begin{pmatrix} 1 & * & * & * & * \\ 0 & 1 & * & \epsilon_1 & * \\ 0 & 0 & \times & \epsilon_2 & \times \end{pmatrix}.$$

Therefore the second column of

$$AP_1P_2 \approx C_2 D_2,$$

now holds the second most dominating column.

Continuing this process the final result is

$$AP = C_k D_k, \qquad D_k = \begin{pmatrix} I & S \end{pmatrix}$$

wkich shows that the first k columns of AP hold the dominating columns of the matrix, and the rank-k approximations of these columns are in C_k.

The algorithm described above is equivalent to Computing the QR decomposition with column pivoting (see Section 7.1),

$$DP = Q \begin{pmatrix} R & \hat{S} \end{pmatrix}$$

where Q is orthogonal, $R \in \mathbb{R}^{k \times k}$ is upper triangular (and assumed to be non-singular), $\hat{S} \in \mathbb{R}^{k \times (n-k)}$, and P is a permutation matrix. Note that if we are only interested in finding the top k sentences, we need not apply any transformations to the matrix of basis vectors, and the algorithm for finding the top k sentences can be implemented in Matlab as follows:

```
% C * D is a rank k approximation of A
[Q,RS,P]=qr(D);
p=[1:n]*P;
pk=p(1:k);    % Indices of the first k columns of AP
```

Example 14.2 We performed a clustering of the preprocessed text of Chapter 13, using the spectral clustering method (i.e., the initial low rank approximation $A \approx CD$ is computed using the SVD), giving six clusters. Then we computed the six top sentences using the method described above. The sentences 1,3,5, and 6 in Example 14.1 were selected, and in addition the following two.

1. Due to the sparsity and the dimension of A (estimated to be of the order billions), it is out of the question to compute the eigenvector using any of the standard methods for dense matrices that are based on applying orthogonal transformations to the matrix described in Chapter 16.

2. In [50] an adaptive method is described that checks the convergence of the components of the pagerank vector and avoids performing the power iteration for those components.

■

Chapter 15

Face Recognition Using Tensor SVD

Human beings are very skillful at recognizing faces even when the face expression, the illumination, the viewing angle, etc. vary. To develop automatic procedures for face recognition that are robust with respect to varying conditions is a challenging research problem that has been investigated using several different approaches. Principal component analysis (i.e. SVD) is a popular technique that often goes under the name "Eigenfaces" [20, 76, 87]. However, this method is best when all pictures are taken under similar conditions, and it does not perform well when several "environment factors" are varied. Also more general bilinear models have been investigated, see e.g. [82].

Recently [90, 91, 92, 93] methods for multi-linear analysis of image ensembles were studied. In particular, the face recognition problem was considered using a tensor model. By letting the modes of the tensor represent a different viewing condition, e.g. illumination or face expression, it became possible to improve the precision of the recognition algorithm compared to the PCA approach.

In this chapter we will describe the "TensorFaces" approach[30] [92]. Since we are dealing with images, which are often stored as $m \times n$ arrays, with m and n of the order 50-300, the computations for each face to identify are quite heavy. We will discuss how the tensor SVD (HOSVD) can also be used for dimensionality reduction to reduce the flop count.

15.1 TensorFaces

Assume that we have a collection of images of n_p persons, where each image is an $m_{i_1} \times m_{i_2}$ array with $m_{i_1} m_{i_2} = n_i$. We will assume that the columns of the images are stacked so that each image is represented by a vector in $\mathbb{R}^{n_i}$. Further assume that each person has been photographed with n_e different face expressions[31]. Often one can have $n_i \geq 5000$, and usually n_i is considerably larger than n_e and n_p. The collection of images is then stored as a tensor

$$\mathcal{A} \in \mathbb{R}^{n_i \times n_e \times n_p}. \tag{15.1}$$

We will refer to the different modes as the image mode, the expression mode, and the person mode, respectively.

[30] See also [90], where tensor methods are used for computing and classifying human motion signatures.

[31] For simplicity we here refer to different illuminations etc. as expressions.

If, for instance we also had photos of each person in different illuminations, viewing angles, etc., then we could represent the image collection by a tensor of higher degree [92]. For simplicity we here only consider the case of a 3-mode tensor. The generalization to higher order tensors is easy.

Example 15.1 We preprocessed images of 10 persons from the Yale Face Database by cropping and decimating each image to 112×78 pixels stored in a vector of length 8736. Five images are illustrated in Figure 15.1.

Figure 15.1. *Person 1 with five different expressions (from the Yale Face Database).*

Altogether each person is photographed in 11 different expressions. ∎

The ordering of the modes is arbitrary, of course; for definiteness and for illustration purposes we will assume the ordering of (15.1). However, in order to (somewhat) emphasize the ordering arbitrariness, will use the notation $\times_e$ for multiplication of the tensor by matrix along the expression mode, and similarly for the other modes. We now write the HOSVD (Theorem 9.3)

$$\mathcal{A} = \mathcal{S} \times_i F \times_e G \times_p H, \tag{15.2}$$

where $\mathcal{S}$ is the core tensor, with the same dimensions as $\mathcal{A}$, and the matrices $F \in \mathbb{R}^{n_i \times n_i}$, $G \in \mathbb{R}^{n_e \times n_e}$, and $H \in \mathbb{R}^{n_p \times n_p}$ are orthogonal.

Example 15.2 We computed the HOSVD of the tensor of face images of ten persons with ten different expressions. The singular values are plotted in Figure 15.2. All ten singular values, respectively, in the expression and person modes are significant, which means that it should be relatively easy to distinguish between epxressions and persons. ∎

The HOSVD can be interpreted in different ways depending on the what it is to be used for. We first illustrate the relation

$$\mathcal{A} = \mathcal{D} \times_e G \times_p H,$$

where $\mathcal{D} = \mathcal{S} \times_i F$.

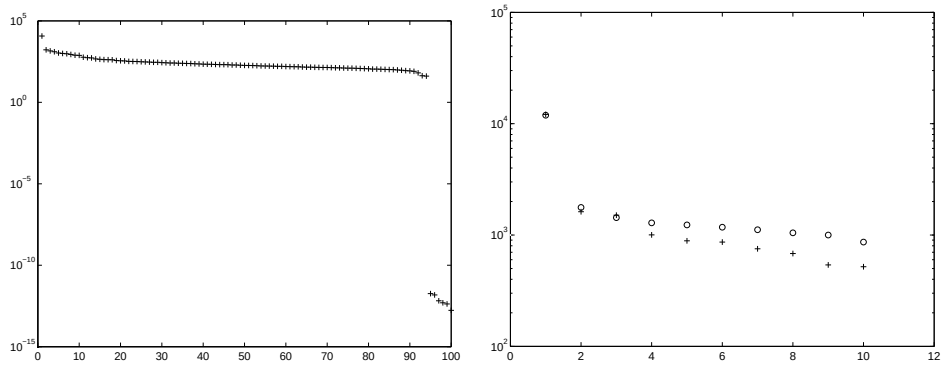
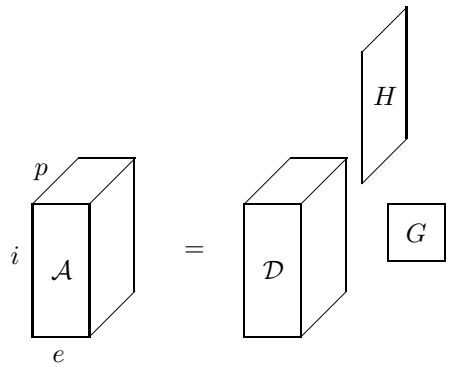


Figure 15.2. *The singular values in the image mode (left), the expression mode (right, +), and the person mode (right, o).*



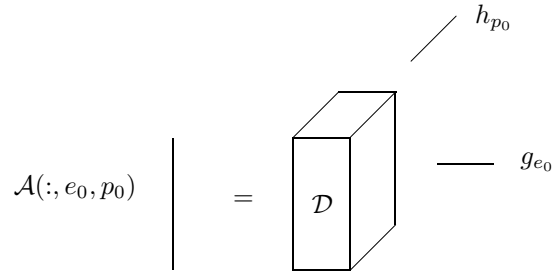
At this point, let us recapitulate the definition of tensor-matrix multiplication (Section 9.2). For definiteness we consider 2-mode, i.e. here e-mode, multiplication,

$$(\mathcal{D} \times_e G)(i_1, j, i_3) = \sum_{k=1}^{n_e} g_{j,k}\, d_{i_1,k,i_3}.$$

We see that fixing a particular value of the expression parameter, i.e. putting $j = e_0$, say, corresponds to using only the e_0'th row of G. By doing the analogous choice in the person mode, we get

$$\mathcal{A}(:, e_0, p_0) = \mathcal{D} \times_e g_{e_0} \times_p h_{p_0}, \tag{15.3}$$

where g_{e_0} denotes the e_0'th row vector of G and h_{p_0} the p_0'th row vector of H. We illustrate (15.3) in the following figure.



We summarize in words:

> The image of person p_0 in expression e_0 can be synthesized by multiplication of the tensor $\mathcal{D}$ by h_{p_0} and g_{e_0} in their respective modes. Thus person p_0 is uniquely characterized by the row vector h_{p_0} and expression e_0 is uniquely characterized by g_{e_0}, via the bilinear form
>
> $$\mathcal{D} \times_e g \times_p h.$$

Example 15.3 The matlab code

```
a=tmul(tmul(D,Ue(4,:),2),Up(6,:),3);
```

gives person six in expression four (happy), see Figure 15.3. Recall that the function

Figure 15.3. *Person 6 in expression four (happy).*

`tmul(A,X,i)` multiplies the tensor `A` by the matrix `X` in mode `i`. ∎

15.2 Face Recognition

We will now consider the classification problem

> Given a image of an unknown person, represented by a vector in $\mathbb{R}^{n_i}$, determine which of the n_p persons it represents, or decide that the unknown person is not in the database.

For the classification we write the HOSVD (15.2) in the following form

$$\mathcal{A} = \mathcal{C} \times_p H, \qquad \mathcal{C} = \mathcal{S} \times_i F \times_e G. \tag{15.4}$$

For a particular expression e we have

$$\mathcal{A}(:, e, :) = \mathcal{C}(:, e, :) \times_p H. \tag{15.5}$$

Obviously we can identify the tensors $\mathcal{A}(:, e, :)$ and $\mathcal{C}(:, e, :)$ with matrices, which we denote A_e and C_e. Therefore, for all the expressions, we have linear relations

$$A_e = C_e H^T, \qquad e = 1, 2, \ldots, n_e. \tag{15.6}$$

Note that the same (orthogonal) matrix H occurs in all n_e relations. We can interpret (15.6) as follows.

> Column p of A_e contains the image of person p in expression e. The columns of C_e are *basis vectors for expression* e, and row p of H, i.e. h_p, holds the *coordinates* of the image of person p in this basis. Furthermore, *the same* h_p *holds the coordinates of the image of person* p *in* **all** *expression bases.*

Next assume that $z \in \mathbb{R}^{n_i}$ is an image of an unknown person in an unknown expression (out of the n_e), and that we want to classify it. We refer to z as a test image. Obviously, if it is an image of person p in expression e, then the coordinates of z in that basis are equal to h_p. Thus we can classify z by computing its coordinates in all the expression bases and checking, for each expression, whether the coordinates of z coincide (or almost coincide) with the elements of any row of H.

The coordinates of z in expression basis e can be found by solving the least squares problem

$$\min_{\alpha_e} \| C_e \alpha_e - z \|_2. \tag{15.7}$$

The algorithm is summarized:

Classification Algorithm Preliminary version.

% z is a test image.

for $e = 1, 2, \ldots, n_e$

  Solve $\min_{\alpha_e} \| C_e \alpha_e - z \|_2$.

  for $p = 1, 2, \ldots, n_p$

    If $\| \alpha_e - h_p \|_2 <$ tol, then classify as person p and **stop**.

  end

end

The amount of work in this algorithm is high: for each test image z we must solve n_e least squares problems (15.7), with $C_e \in \mathbb{R}^{n_i \times n_p}$. We can roughly estimate the amount of work to $O(n_i n_e n_p)$ flops.

However, recall from (15.4) that $\mathcal{C} = \mathcal{S} \times_i F \times_e G$, which implies

$$C_e = FB_e,$$

where $B_e \in \mathbb{R}^{n_e n_p \times n_p}$ is the matrix identified with $(\mathcal{S} \times_e G)(:, e, :)$. Note that $F \in \mathbb{R}^{n_i \times n_e n_p}$; we assume that n_i is considerably larger than $n_e n_p$. Then, for the analysis only, enlarge the matrix so that it becomes square and orthogonal,

$$\hat{F} = \begin{pmatrix} F & F^\perp \end{pmatrix}, \qquad \hat{F}^T \hat{F} = I.$$

Now insert $\hat{F}^T$ inside the norm,

$$\begin{aligned} \| C_e\alpha_e - z \|_2^2 = \| \hat{F}^T (FB_e\alpha_e - z) \|_2 &= \left\| \begin{pmatrix} B_e\alpha_e - F^T z \\ -(F^\perp)^T z \end{pmatrix} \right\|_2^2 \\ &= \| B_e\alpha_e - F^T z \|_2^2 + \| -(F^\perp)^T z \|_2^2. \end{aligned}$$

It follows that we can solve the n_e least squares problems by first computing $F^T z$ and then solving

$$\min_{\alpha_e} \| B_e\alpha_e - F^T z \|_2, \qquad e = 1, 2, \ldots, n_e. \tag{15.8}$$

The matrix B_e has dimension $n_e n_p \times n_p$, so it is much cheaper to solve (15.8) than (15.7). It is also possible to precompute a QR decomposition of each matrix B_e to further reduce the work. Thus we arrive at the following algorithm.

Classification Algorithm

Preprocessing step. Compute and save the thin QR decompositions of all the B_e matrices, $B_e = Q_e R_e$, $e = 1, 2, \ldots, n_e$.

% z is a test image.

Compute $\hat{z} = F^T z$.

for $e = 1, 2, \ldots, n_e$

  Solve $R_e\alpha_e = Q_e^T \hat{z}$ for α_e.

  for $p = 1, 2, \ldots, n_p$

    If $\| \alpha_e - h_p \|_2 <$ tol, then classify as person p and **stop**.

  end

end

In a typical application it is likely that even if the test image is an image of a person in the database, it is taken with another expression that is not represented in the database. However, the above algorithm works well also in such cases, as reported in [92].

Example 15.4 For each of the ten persons in the Yale database there is an image, where the person winks. We took those at test images, and computed the closest image in the database, essentially by the algorithm above. In all cases the correct person was identified, see Figure 15.4. ∎

Figure 15.4. *The upper row shows the images to be classified, the bottom row the corresponding closest image in the database.*

15.3 Face Recognition with HOSVD Compression

Due to the ordering properties of the core, with respect to the different modes (Theorem 9.3), we may be able to truncate the core in such a way that the truncated HOSVD is still a good approximation of $\mathcal{A}$. Define $F_k = F(:, 1:k)$, for some value of k that we assume is much smaller than n_i, but larger than n_p. Then, for the analysis only, enlarge the matrix so that it becomes square and orthogonal,

$$\widetilde{F} = (F_k \, \widetilde{F}_\perp), \qquad \widetilde{F}^T \widetilde{F} = I.$$

Then truncate the core tensor similarly, i.e. put

$$\widehat{\mathcal{C}} = (\mathcal{S} \times_e G)(1:k, :, :) \times_i F_k. \tag{15.9}$$

It follows from Theorem 9.3 and the fact that the multiplication by G in the e-mode does not affect the HOSVD ordering properties in the i-mode, that

$$\|\widehat{\mathcal{C}} - \mathcal{C}\|_F^2 = \sum_{\nu=k+1}^{n_i} \sigma_\nu^{(i)}.$$

Therefore, if the rate of decay of the singular values is fast enough, it should be possible to obtain good recognition precision, in spite of the compression. So if we use $\widehat{\mathcal{C}}$ in the algorithm of the preceding section, we will have to solve least squares problems

$$\min_{\alpha_e} \| \, \widehat{C}_e \alpha_e - z \, \|_2,$$

with the obvious definition of $\widehat{C}_e$. Now, from (15.9) we have $\widehat{C}_e = F_k \widehat{B}_e$, where $\widehat{B}_e \in \mathbb{R}^{k \times n_p}$. Multiplying by $\widetilde{F}$ inside the norm sign we get

$$\| \, \widehat{C}_e \alpha_e - z \, \|_2^2 = \| \, \widehat{B}_e \alpha_e - F_k^T z \, \|_2^2 + \| \, \widetilde{F}_\perp^T z \, \|_2^2.$$

In this "compressed" variant of the recognition algorithm we only perform one operation with matrix of first dimension n_i (as opposed to the uncompressed variant, where n_e such operations are performed).

Example 15.5 We used the same data as in the previous example, but truncated the orthogonal basis in the image mode to rank k. With $k = 10$ all the test images were correctly classified, but with $k = 5$ two out of ten images were incorrectly classified. Thus a substantial rank reduction (from 100 to ten) was possible without sacrificing classification accuracy. ∎

In our illustrating example, the number of persons and different expressions are so small that it is not necessary to further compress the data. However, in a realistic application, in order to be able to classify images in a reasonable time, one can truncate the core tensor in the expression and person modes, and thus solve much smaller least squares problems than in the uncompressed case.

Part III

Computing the Matrix Decompositions

Chapter 16

Computing the Schur and Singular Value Decompositions

In MATLAB and other modern programming environments eigenvalues and singular values are obtained using high-level functions, e.g. `eig(A)` and `svd(A)`. These functions implement algorithms from the LAPACK subroutine library [1]. In order to give an orientation about what is behind such high-level functions, we will in this chapter briefly describe some methods for computing eigenvalues and singular values of dense matrices and large sparse matrices. For more extensive treatments, see e.g. [36, 4, 69].

The functions `eig` and `svd` are used for dense matrices, i.e. matrices where most the elements are non-zero. Eigenvalue algorithms for a dense matrix have two phases: (1) For efficiency it is necessary to first transform the matrix to compact form: tridiagonal in the symmetric case and Hessenberg in the non-symmetric case. This part consists of a sequence of orthogonal transformations. (2) Then there is an iterative part, the QR algorithm. For large, sparse matrices it is usually not possible (or even interesting) to compute all the eigenvalues. Here there are special methods that take advantage of the sparsity.

As background material for the give some theoretical results concerning perturbation theory for the eigenvalue problem. In addition, we briefly describe the power method for computing egienvalues, and its cousin, inverse iteration.

In linear textbooks the eigenvalue problem for the matrix $A \in \mathbb{R}^{n\times n}$ is often introduced as the solution of the polynomial equation

$$\det(A - \lambda I) = 0.$$

For general problems, this approach is useless for two reasons: (1) for matrices of interesting dimensions it is too costly to compute the determinant, and (2) even if the determinant and the polynomial could be computed, the eigenvalues are extrememly sensitive to perturbation in the coefficients of the polynomial. Instead, the basic tool in the numerical computation of eigenvalues are *similarity transformations*: Let X be a nonsingular matrix.

$$A \longrightarrow X^{-1}AX, \tag{16.1}$$

It is obvious that the eigenvalues are preserved under this transformation,

$$Ax = \lambda x \quad \Leftrightarrow \quad X^{-1}AXy = \lambda y, \tag{16.2}$$

where $y = X^{-1}x$. An important special case is an *orthogonal similarity transformation*:

$$A \longrightarrow V^T AV, \tag{16.3}$$

for an orthogonal matrix V.

16.1 Perturbation Theory

The QR algorithm for computing eigenvalues is based orthogonal similarity transformations (16.3), and it computes a sequence of transformations such that the final result is diagonal (in the case of symmetric A) or triangular (for non-symmetric A). Since the algorithm is iterative, it is necesary to consider the problem of deciding when a floating point number is small enough to be considered as zero numerically. To have a sound theoretical basis for this decision, we must know how sensitive the eigenvalues and eigenvectors are to small perturbations of the data, i.e. the coefficients of the matrix.

In this section we will first give a couple of perturbation results, without proofs[32], for a symmetric matrix $A \in \mathbb{R}^{n\times n}$. Assume that eigenvalues of $n \times n$ matrices are ordered

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n.$$

We will consider a perturbed matrix $A + E$, and ask the question how far the eigenvalues and eigenvectors of $A + E$ are from those of A.

Example 16.1 Let

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0.5 & 0 \\ 0 & 0.5 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad A + E = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0.5 & 0 \\ 0 & 0.5 & 2 & 10^{-15} \\ 0 & 0 & 10^{-15} & 1 \end{pmatrix}.$$

How much can the eigenvalues of A and $A+E$ deviate? This is the typical situation in the QR algorithm. ∎

Theorem 16.2. *Let $A \in \mathbb{R}^{n\times n}$ and $A + E$ be symmetric matrices. Then*

$$\lambda_k(A) + \lambda_n(E) \leq \lambda_k(A + E) \leq \lambda_k(A) + \lambda_1(E), \qquad k = 1, 2, \ldots, n.$$

From the theorem we see that if we perturb the matrix elements by ϵ, then the eigenvalues are also perturbed by $O(\epsilon)$. Thus, in Example 16.1 the eigenvalues of the two matrices differ by 10^{-15} at the most.

The sensitivity of the eigenvectors depends on the separation of eigenvalues.

Theorem 16.3. *Let $[\lambda, q]$ be an eigenvalue-eigenvector pair of the symmetric matrix A, and assume that the eigenvalue is simple. Form the orthogonal matrix $Q = \begin{pmatrix} q & Q_1 \end{pmatrix}$ and partition the matrices Q^TAQ and Q^TEQ*

$$Q^TAQ = \begin{pmatrix} \lambda & 0 \\ 0 & A_2 \end{pmatrix}, \qquad Q^TEQ = \begin{pmatrix} \epsilon & e^T \\ e & E_2 \end{pmatrix}.$$

[32] For proofs, we refer, e.g. to [36, Chapters 7-8].

Define

$$d = \min_{\lambda_i(A) \neq \lambda} |\lambda - \lambda_i(A)|,$$

and assume that $\| E \|_2 \leq d/4$. *Then there exists an eigenvector* $\hat{q}$ *of* $A + E$ *such that the distance between* q *and* $\hat{q}$, *measured as the sine of the angle between the vectors, is bounded*

$$\sin(\theta(q, \hat{q})) \leq \frac{4 \| e \|_2}{d}.$$

The theorem is only meaningful if the eigenvalue is simple. It shows that eigenvectors corresponding to close eigenvalues can be sensitive to perturbations, and are therefore more difficult to compute to high accuracy.

Example 16.4 The eigenvalues of the matrix A in Example 16.1 are

```
0.8820, 1.0000, 2.0000, 3.1180.
```

The deviation between the eigenvectors of A and $A+E$ corresponding to the smallest eigenvalue can be estimated by

$$\frac{4\|e\|_2}{|0.8820 - 1|} \approx 1.07 \cdot 10^{-14}.$$

Since the eigenvalues are well separated the eigenvectors are rather insensitive to perturbations in the data. ∎

In order to formulate perturbation results for non-symmetric matrices we first introduce the *Schur decomposition*.

Theorem 16.5. *For any (symmetric or non-symmetric) matrix* $A \in \mathbb{R}^{n \times n}$ *there exists an orthogonal matrix* V *such that*

$$V^T A V = R, \tag{16.4}$$

where R *is upper triangular. The diagonal elements of* R *are the eigenvalues of* A.

If A is symmetric, then R is diagonal, and the Schur decomposition is the same as the eigenvalue-eigenvector decomposition. If A is non-symmetric, then some or all its eigenvalues may be complex.

Example 16.6 The Schur decomposition is a standard function in Matlab:

```
>> a=randn(3)
a = -0.4326    0.2877    1.1892
    -1.6656   -1.1465   -0.0376
     0.1253    1.1909    0.3273

>> [Q,R]=schur(a)
```

```
Q = 0.2827    0.2924    0.9136
    0.8191   -0.5691   -0.0713
   -0.4991   -0.7685    0.4004

R = -1.6984    0.2644   -1.2548
          0    0.2233    0.7223
          0   -1.4713    0.2233
```

If the matrix is real then a real variant of the deomposition is computed, where two-by-two blocks along the diagonal hold the possible complex-conjugate eigenvalues. If we compute the eigenvalue decomposition, we get:

```
>> [U,D]=eig(a)

U = 0.2827             0.4094 - 0.3992i   0.4094 + 0.3992i
    0.8191            -0.0950 + 0.5569i  -0.0950 - 0.5569i
   -0.4991             0.5948             0.5948

D =  -1.6984               0                  0
           0          0.2233+1.0309i          0
           0               0            0.2233-1.0309i
```

■

It is straightforward to show that the diagonal elements of R are the eigenvalues of A. Consider the *characteristic equation*

$$\det(R - \lambda I) = 0.$$

Since the diagonal elements of $R-\lambda I$ are $r_{ii}-\lambda$, and the determinant of a triangular matrix is the product of the diagonal elements, we see that $\det(R - \lambda I) = 0$ if and only if $\lambda = r_{ii}$ for some i, which means that r_{ii} is an eigenvalue.

The sensitivity of the eigenvalues of a non-symmetric matrix depends on how much the matrix deviates from being symmetric.

Theorem 16.7. *Let $Q^T AQ = D + N$ be the Schur decomposition of A, and let τ denote an eigenvalue of $A + E$. Further, let p be the smallest integer such that $N^p = 0$. Then*

$$\min_{\lambda_i(A)} |\lambda_i(A) - \tau| \leq max(\eta, \eta^{1/p}),$$

where

$$\eta = \| E \|_2 \sum_{k=0}^{p-1} \| N \|_2^k.$$

The degree of non-symmetry of A can be measured by $\| N \|_2$. The theorem shows that the eigenvalues of a highly non-symmetric matrix can be considerably more sensitive to perturbations than the eigenvalues of a symmetric matrix, cf. Theorem 16.2.

Example 16.8 The matrices

$$A = \begin{pmatrix} 2 & 0 & 10^3 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \qquad B = A + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 10^{-10} & 0 & 0 \end{pmatrix},$$

have the eigenvalues

```
2, 2, 2,
```

and

```
2.00031622776602, 1.99968377223398, 2.00000000000000,
```

respectively. The relevant quantity for the perturbation is $\eta^{1/2} \approx 3.164 \cdot 10^{-04}$. ∎

The non-symmetric version of Theorem 16.3 is similar: again the angle between the eigenvectors depends on the separation of the eigenvalues. We give a simplified statement below, where we disregard the possibility of a complex eigenvalue.

Theorem 16.9. *Let $[\lambda, q]$ be an eigenvalue-eigenvector pair of A, and assume that the eigenvalue is simple. Form the orthogonal matrix $Q = \begin{pmatrix} q & Q_1 \end{pmatrix}$ and partition the matrices $Q^T A Q$ and $Q^T E Q$,*

$$Q^T A Q = \begin{pmatrix} \lambda & v^T \\ 0 & A_2 \end{pmatrix}, \qquad Q^T E Q = \begin{pmatrix} \epsilon & e^T \\ \delta & E_2 \end{pmatrix}.$$

Define

$$d = \sigma_{min}(A_2 - \lambda I),$$

and assume $d > 0$. If the perturbation E is small enough, then there exists an eigenvector $\hat{q}$ of $A + E$ such that the distance between q and $\hat{q}$ measured as the sine of the angle between the vectors is bounded by

$$\sin(\theta(q, \hat{q})) \leq \frac{4 \,\|\, \delta \,\|_2}{d}.$$

The theorem says essentially that if we perturb A by ϵ, then the eigenvector is perturbed by ϵ/d.

Example 16.10 Let the tridiagonal matrix be defined

$$A_n = \begin{pmatrix} 2 & -1.1 & & & \\ -0.9 & 2 & -1.1 & & \\ & \ddots & \ddots & \ddots & \\ & & -0.9 & 2 & -1.1 \\ & & & -0.9 & 2 \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

For $n = 100$ its smallest eigenvalue is 0.01098771, approximately. The following Matlab script computes the quantity d in Theorem (16.9).

```
% xn is the eigenvector corresponding to
% the smallest eigenvalue
[Q,r]=qr(xn);
H=Q'*A*Q; lam=H(1,1);
A2=H(2:n,2:n);
d=min(svd(A2-lam*eye(size(A2))));
```

We get $d = 1.6207 \cdot 10^{-4}$. Therefore, if we perturb the matrix by 10^{-10}, say, this may change the eigenvector by a factor $4 \cdot 10^{-6}$, approximately. ∎

16.2 The Power Method

The power method is a classical iterative method for computing the largest (in magnitude) eigenvalue and the corresponding eigenvector of a matrix. Its convergence can be very slow, depending on the distribution of eigenvalues. Therefore it should never be used for dense matrices. Usually for sparse matrices one should use a variant of the Lanczos method or the Jacobi-Davidson method, see [4] and Section 16.8. However, there are applications where the dimension of the problem is so huge that no other method is viable, see Chapter 13.

In spite of its limited usefulness for practical problem, the power method is important from a theoretical point of view. In addition, there is a variation of the power method that is of great practical importance, see Section 16.2.1.

The power method for computing the largest eigenvalue

```
% Initial approximation x
for k=1:maxit
  y=A*x;
  lambda=y'*x;
  if norm(y-lambda*x) < tol*abs(lambda)
    break  % stop the iterations
  end
  x=1/norm(y)*y;
end
```

One strength of the power method is that it only uses the matrix A in the multiplication $y = Ax$. In addition, only two vectors of dimension n need be stored. This important in appplications, where A is very large and sparse, cf. Chapter 13. The main weakness of the method is that in many cases the convergence is very slow.

Example 16.11 Consider again the tridiagonal matrix

$$A_n = \begin{pmatrix} 2 & -1.1 & & & \\ -0.9 & 2 & -1.1 & & \\ & \ddots & \ddots & \ddots & \\ & & -0.9 & 2 & -1.1 \\ & & & -0.9 & 2 \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

The two largest eigenvalues of A_{20} are 3.9677 and 3.9016, approximately. As initial

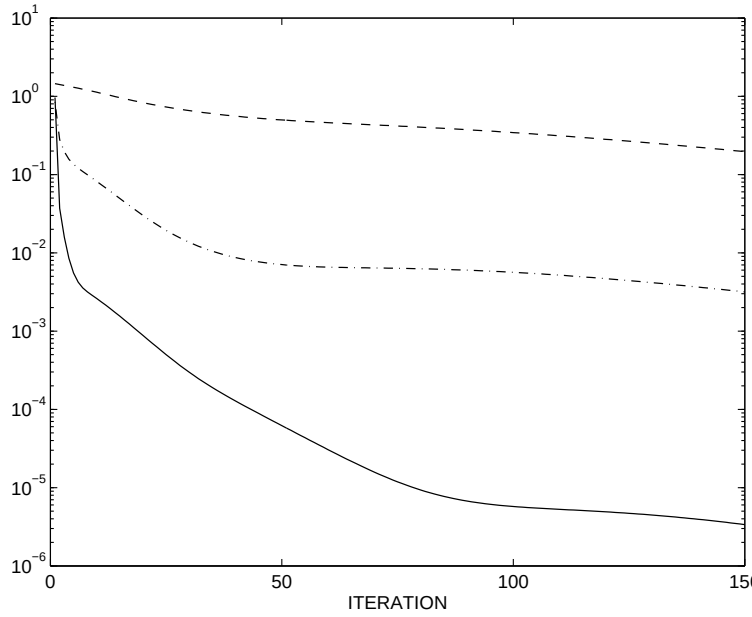


Figure 16.1. *Power iterations for A_{20}: The relative residual $\| y^{(k)} - \lambda^{(k)} x^{(k)} \| / \lambda^{(k)}$ (solid), the absolute error in the eigenvalue approximation (dashed-dotted) and the angle (in radians) between the exact eigenvector and the approximation (dashed).*

approximation we chose a random vector. In Figure 16.1 we plot different error measures during the iterations: the relative residual $\| Ax^{(k)} - \lambda^{(k)} x^{(k)} \| / \lambda_1$, ($\lambda^{(k)}$ denotes the approximation of λ_1 in the kth iteration), the error in the eigenvalue approximation and the angle between the exact and the approximate eigenvector. After 150 iterations the relative error in the computed approximation of the eigenvalue is 0.0032. ∎

The convergence of the power method depends on the distribution of eigenvalues. Assume that A is diagonalizable, i.e. there exists a nonsingular matrix X of eigenvectors, $X^{-1}AX = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. Further, assume that the largest eigenvalue in magnitude is simple, and that λ_i are ordered $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$. Expand the initial approximation $x^{(0)}$ in terms of the eigenvectors,

$$x^{(0)} = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n,$$

where $c_1 \neq 0$ is assumed[33]. Then we have

$$A^k x^{(0)} = \lambda_1^k \left(c_1 x_1 + \sum_{j=2}^{n} c_j \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j \right). \tag{16.5}$$

Obviously, since for $j = 2, 3, \ldots$, we have $|\lambda_j| < |\lambda_1|$, the second term tends to zero and the power method converges to the eigenvector x_1. The rate of convergence is determined by the ratio $|\lambda_2/\lambda_1|$. If this ratio is close to 1, then the iteration is very slow.

Example 16.12 In the Example 16.11 we have $\lambda_2(A_{20})/\lambda_1(A_{20}) = 0.9833$. It follows that

$$0.9833^{150} \approx 0.0802,$$

which indicates that the convergence is quite slow, as seen in Figure 16.1. This is comparable to the reduction of the angle between the exact and the approximate eigenvector during 150 iterations: from 1.2847 radians to 0.0306. ∎

A stopping criterion for the power iteration can be formulated in terms of the residual vector for the eigenvalue problem. Let $\hat{\lambda}$ be the computed approximation of the eigenvalue, and $\hat{x}$ the corresponding approximate eigenvector. Then it can be shown [81], [4, p. 229] that the optimal error matrix E, for which

$$(A + E)\hat{x} = \hat{\lambda}\hat{x},$$

exactly, satisfies

$$\| E \|_2 = \| r \|_2,$$

where $r = A\hat{x} - \hat{\lambda}\hat{x}$. Thus, if we iterate until the residual r is smaller in norm than some tolerance δ, then we know that our eigenvalue approximation is the *exact* eigenvalue of a matrix $\hat{A} = A + E$ such that $\| A - \hat{A} \|_2 \leq \delta$.

16.2.1 Inverse Iteration

If we iterate with A^{-1} in the power method,

$$x^{(k)} = A^{-1} x^{(k-1)},$$

then, since the eigenvalues of A^{-1} are $1/\lambda_i$, the sequence of eigenvalue approximations converges toward $1/\lambda_{min}$, where λ_{min} is the eigenvalue of smallest absolute value. Even better, if we have a good enough approximation of one of the eigenvalues, $\tau \approx \lambda_k$, then the shifted matrix $A - \tau I$ has the smallest eigenvalue $\lambda_k - \tau$. Thus, we can expect very fast convergence in the "inverse power method". This method is called *inverse iteration*.

Inverse iteration

[33] This assumption can be expected to be satisfied in floating point arithmetic, if not at the first iteration, so definitely after the second, due to round off.

```
% Initial approximation x and eigenvalue approximation tau
[L,U]=lu(A - tau*I);
for k=1:maxit
  y=U\(L\x);
  theta=y'*x;
  if norm(y-theta*x) < tol*abs(theta)
    break  % stop the iteration
  end
  x=1/norm(y)*y;
end
lambda=tau+1/theta; x=1/norm(y)*y;
```

Example 16.13 The smallest eigenvalue of the matrix A_{100} from Example 16.10 is $\lambda_{100} = 0.01098771187192$ to 14 decimals accuracy. If we use the approximation $\lambda_{100} \approx \tau = 0.011$ and apply inverse iteration, we get fast convergence, see Figure 16.2. In this example the convergence factor is

$$\left| \frac{\lambda_{100} - \tau}{\lambda_{99} - \tau} \right| \approx 0.0042748,$$

which means that after 4 iterations, say, the error is reduced by a factor of the order $3 \cdot 10^{-10}$. ∎

To be efficient, inverse iteration requires that we have good approximation of the eigenvalue. In addition, we must be able to solve linear systems $(A - \tau I)y = x$ (for y) cheaply. If A is a band matrix, then the LU decomposition can be obtained easily and in each iteration the system can be solved by forward and back substitution (as in the code above). The same method can be used for other sparse matrices if a sparse LU decomposition can be computed without too much fill-in.

16.3 Similarity Reduction to Tridiagonal Form

The QR algorithm that we will introduce in Section 16.4, is an iterative algorithm, where in each step a QR decomposition is computed. If it is applied to a dense matrix $A \in \mathbb{R}^{n \times n}$, then the cost of a step is $O(n^3)$. This prohibitively high cost can be reduced substantially by first transforming the matrix to compact form, by an orthogonal similarity transformation, see (16.1),

$$A \longrightarrow V^T A V,$$

for an orthogonal matrix V. We have already seen in (16.2) that the eigenvalues are preserved under this transformation,

$$Ax = \lambda x \quad \Leftrightarrow \quad V^T A V y = \lambda y,$$

where $y = V^T x$.

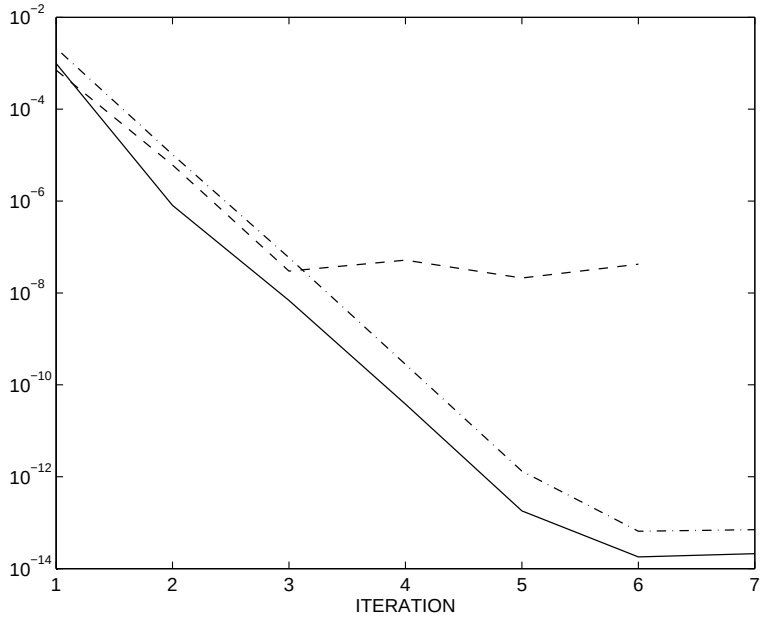


Figure 16.2. *Inverse iterations for A_{100} with $\tau = 0.011$: The relative residual $\| Ax^{(k)} - \lambda^{(k)} x^{(k)} \| / \lambda^{(k)}$ (solid), the absolute error in the eigenvalue approximation (dashed-dotted) and the angle (in radians) between the exact eigenvector and the approximation (dashed).*

Consider first a symmetric matrix $A \in \mathbb{R}^{n \times n}$. By a sequence of Householder transformations it can be reduced to tridiagonal form. We illustrate the procedure using an example with $n = 6$. First we construct a transformation that zeros the elements in positions 3 through n in the first column when we multiply A from the left.

$$H_1 A = H_1 \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{pmatrix} = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{pmatrix}.$$

Elements that are changed in the transformation are denoted by $*$. Note that the elements in the first row are not changed. In an orthogonal similarity transformation we shall multiply by the same matrix transposed from the right[34].Since in the left

[34]The Householder matrices are symmetric, so we need not not transpose.

multiplication the first row was not touched, the first column will remain unchanged:

$$H_1 A H_1^T = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \end{pmatrix} H_1^T = \begin{pmatrix} \times & * & 0 & 0 & 0 & 0 \\ \times & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{pmatrix}.$$

Due to symmetry, elements 3 through n in the first row will be equal to zero.

In the next step we zero the elements in the second column in positions 4 through n. Since this affects only rows 3 through n and the corresponding columns, this does not destroy the zeros that we created in the first step. The result is

$$H_2 H_1 A H_1^T H_2^T = \begin{pmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & * & 0 & 0 & 0 \\ 0 & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & 0 & * & * & * & * \end{pmatrix}.$$

After $n-2$ such similarity transformations the matrix is in tridiagonal form,

$$V^T A V = \begin{pmatrix} \times & \times & 0 & 0 & 0 & 0 \\ \times & \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{pmatrix},$$

where $V = H_1 H_2 \cdots H_{n-2}$.

In summary, we have in this section demonstrated how a symmetrix matrix can be reduced to tridiagonal form by a sequence of $n-2$ Householder transformations

$$A \longrightarrow T = V^T A V, \qquad V = H_1 H_2 \cdots H_{n-2}, \tag{16.6}$$

Since the reduction is done by similarity transformations, the tridiagonal matrix T has the same eigenvalues as A.

The reduction to tridiagonal form requires $4n^3/3$ flops if one takes advantage of symmetry. As in the case of QR decomposition, the Householder transformations can be stored in the subdiagonal part of A. If V is computed explicitly, this takes $4n^3/3$ additional flops.

16.4 The QR Algorithm for Symmetric Tridiagonal Matrices

We will now give a sketch of the QR algorithm for a symmetric, tridiagonal matrix. We emphasize that our MATLAB codes are greatly simplified and are only intended

to demonstrate the basic ideas of the algorithm. The actual software (in LAPACK) contain numerous features for efficiency, robustness and numerical stability.

The procedure that we will describe, can be considered as a continuation of the similarity reduction (16.6), but now we will reduce the matrix T to diagonal form

$$T \longrightarrow \Lambda = Q^T T Q, \qquad Q = Q_1 Q_2 \cdots , \tag{16.7}$$

where $\Lambda = \mathrm{diag}(\lambda_1\, \lambda_2\, \ldots,\, \lambda_n)$. The matrices Q_i will be orthogonal, but here they will be constructed using plane rotations. However, the most important difference between (16.6) and (16.7) is that there does not exist a finite algorithm, for computing Λ. By a finite algorithm we here mean an algorithm for computing the diagonalization of T, *in the field of real numbers, i.e. in exact arithmetic*, using a finite number of operations. We compute a sequence of matrices,

$$T_0 := T, \qquad T_i = Q_i^T T_{i-1} Q_i, \qquad i = 1, 2, \ldots, \tag{16.8}$$

such that it converges to a diagonal matrix

$$\lim_{i \to \infty} T_i = \Lambda.$$

We will demonstrate in numerical examples that the convergence is very rapid, so that *in a floating point system the algorithm can actually be considered as finite.* Since all the transformations in (16.8) are similarity transformations, the diagonal elements of Λ are the eigenvalues of T.

We now give a provisional version of the QR algorithm for a symmetric tridiagonal matrix $T \in \mathbb{R}^{n \times n}$.

QR iteration for symmetric T: bottom eigenvalue

```
for i=1:maxit   % Provisional simplification
  mu=wilkshift(T(n-1:n,n-1:n));
  [Q,R]=qr(T-mu*I);
  T=R*Q+mu*I
end

function mu=wilkshift(T);
  % Compute the Wilkinson shift
  l=eig(T);
  if abs(l(1)-T(2,2))<abs(l(2)-T(2,2))
    mu=l(1);
  else
    mu=l(2);
  end
```

We see that the QR decomposition of a shifted matrix $QR = T - \tau I$ is computed, and that the shift is then added back $T := RQ + \tau I$. The shift is the eigenvalue of

the 2×2 submatrix in the lower right corner that is closest to t_{nn}. This is called the *Wilkinson shift*.

We applied the algorithm to the matrix

$$T_6 = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix} \tag{16.9}$$

After the first step the result was (slightly edited for readability)

```
T = 1.0000    0.7071         0         0         0         0
    0.7071    2.0000    1.2247         0         0         0
         0    1.2247    2.3333   -0.9428         0         0
         0         0   -0.9428    1.6667    0.8660         0
         0         0         0    0.8660    2.0000   -0.5000
         0         0         0         0   -0.5000    3.0000
```

We see first that the triangular structure is preserved, and that the off-diagonal elements in the lower right corner have become smaller. We perform 3 more steps and look more closely at that submatrix.

```
 2.36530292572181  -0.02609619264716
-0.02609619264716   3.24632297453998

 2.59270689576885   0.00000366571479
 0.00000366571479   3.24697960370634

 2.77097818052654   0.00000000000000
-0.00000000000000   3.24697960371747
```

Thus, after 4 iterations the off-diagonal element has become zero in working precision and therefore we have an eigenvalue in the lower right corner.

When the eigenvalue has been found, we can deflate it and continue working with the the upper $(n-1) \times (n-1)$ submatrix, which is now

```
0.4374    0.3176         0         0         0
0.3176    0.7961   -0.4395         0         0
     0   -0.4395    1.3198    0.2922         0
     0         0    0.2922    3.4288    0.5902
     0         0         0    0.5902    2.7710
```

We now apply the same algorithm to this matrix. Tracing its lower right submatrix during three subsequent steps, we have

```
 3.74629910763238  -0.01184028941948
-0.01184028941948   2.44513898239641
```

```
 3.68352336882524   0.00000009405188
 0.00000009405188   2.44504186791263

 3.54823766699472   0.00000000000000
-0.00000000000000   2.44504186791263
```

After these three iterations we have again an eigenvalue at the lower right corner. The algorithm now proceeds by deflating this eigenvalue and reducing the dimension of the active matrix by one. A preliminary implementation of the algorithm is given below.

QR iteration for symmetric T

```
function [D,it]=qrtrid(T);
  % Compute the eigenvalues of a symmetric tridiagonal
  % matrix using the QR algorithm with explicit
  % Wilkinson shift
    n=size(T,1); it=0;
    for i=n:-1:3
      while abs(T(i-1,i)) > ...
                (abs(T(i,i))+abs(T(i-1,i-1)))*C*eps
        it=it+1;
        mu=wilkshift(T(i-1:i,i-1:i));
        [Q,R]=qr(T(1:i,1:i)-mu*eye(i));
        T=R*Q+mu*eye(i);
      end
      D(i)=T(i,i);
    end
    D(1:2)=eig(T(1:2,1:2))';
```

For a given submatrix `T(1:i,1:i)` the QR steps are iterated until the stopping criterion

$$\frac{|t_{i-1,i}|}{|t_{i-1,i-1}| + |t_{i,i}|} < C\mu,$$

is satisfied, where C is a small constant and μ is the unit roundoff. From Theorem 16.2 we see that considering such a tiny element as a numerical zero, leads to a a very small (and acceptable) perturbation of the eigenvalues. In actual software, a slightly more complicated stopping criterion is used.

When applied to the matrix T_{100} (cf. (16.9)) with the value $C = 5$, 204 QR steps were taken, i.e. approximately 2 steps per eigenvalue. The maximum deviation between the computed eigenvalues and those computed by MATLAB'S `eig` was $2.9 \cdot 10^{-15}$.

It is of course inefficient to compute the QR decomposition of a tridiagonal matrix using the MATLAB function `qr`, which is a Householder-based algorithm.

Instead the decomposition should be computed using $n-1$ plane rotations in $O(n)$ flops. We illustrate the procedure with a small example, where the tridiagonal matrix $T = T^{(0)}$ is 6×6. The first subdiagonal element (from the top) is zeroed by a rotation from the left in the (1,2) plane, $G_1^T(T^{(0)} - \tau I)$, and then the second subdiagonal is zeroed by a rotation in (2,3), $G_2^T G_1^T(T^{(0)} - \tau I)$. Symbolically,

$$\begin{pmatrix} \times & \times & & & & \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix} \longrightarrow \begin{pmatrix} \times & \times & + & & & \\ 0 & \times & \times & + & & \\ & 0 & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

Note the fill-in (new non-zero elements, denoted +) that is created. After $n-1$ steps we have an upper triangular matrix with three non-zero diagonals:

$$R = G_{n-1}^T \cdots G_1^T (T^{(0)} - \tau I) = \begin{pmatrix} \times & \times & + & & & \\ 0 & \times & \times & + & & \\ & 0 & \times & \times & + & \\ & & 0 & \times & \times & + \\ & & & 0 & \times & \times \\ & & & & 0 & \times \end{pmatrix}.$$

We then apply the rotations from the right, $RG_1 \cdots G_{n-1}$, i.e. we start with a transformation involving the first two columns. Then follows a rotation involving the second and third columns. The result after two steps is

$$\begin{pmatrix} \times & \times & \times & & & \\ + & \times & \times & \times & & \\ & + & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \\ & & & & & \times \end{pmatrix}.$$

We see that the zeroes that we introduced below the diagonal in the transformations from the left are systematically filled in. After $n-1$ steps we have

$$T^{(1)} = RG_1G_2 \cdots G_{n-1} + \tau I = \begin{pmatrix} \times & \times & \times & & & \\ + & \times & \times & \times & & \\ & + & \times & \times & \times & \\ & & + & \times & \times & \times \\ & & & + & \times & \times \\ & & & & + & \times \end{pmatrix}.$$

But we have made a similarity transformation: With $Q = G_1G_2 \cdots G_{n-1}$ and using $R = Q^T(T^{(0)} - \tau I)$, we can write

$$T^{(1)} = RQ + \tau I = Q^T(T^{(0)} - \tau I)Q + \tau I = Q^T T^{(0)} Q, \tag{16.10}$$

so we know that $T^{(1)}$ is symmetric,

$$T^{(1)} = \begin{pmatrix} \times & \times & & & & \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

Thus we have shown the following result.

Proposition 16.14. *The QR step for a tridiagonal matrix*

$$QR = T^{(k)} - \tau_k I, \qquad T^{(k+1)} = RQ + \tau_k I,$$

is a similarity transformation

$$T^{(k+1)} = Q^T T^{(k)} Q, \tag{16.11}$$

and the tridiagonal structure is preserved. The transformation can be computed with plane rotations in $O(n)$ flops.

From (16.10) it may appear as if the shift plays no significant role. However, it determines the value of the orthogonal transformation in the QR step. Actually, the shift strategy is absolutely necessary for the algorithm to be efficient: If no shifts are performed, then the QR algorithm usually converges very slowly, in fact as slowly as the power method, cf. Section 16.2. On the other hand, it can be proved that with Wilkinson shifts the off-diagonal element in the bottom right 2×2 submatrix converges cubically to zero (Wilkinson 1968).

In actual software for the QR algorithm, there are several enhancements of the algorithm that we have outlined above. For instance, the algorithm checks all off-diagonals if they are small: when a negligable off-diagonal element is found, then the problem can be split in two. There is also a divide-and-concquer variant of the QR algorithm. For an extensive treatment, see [36, Chapter 8].

16.4.1 Implicit shifts

One important aspect of the QR algorithm is that the shifts can be performed *implicitly*. This is especially useful for the application of the algorithm to the SVD and the non-symmetric eigenproblem. This variant is based on the *Implicit Q Theorem*, which we here give in slightly simplified form.

Theorem 16.15. *Let A be symmetric, and assume that Q and V are orthogonal matrices such that $Q^T AQ$ and $V^T AV$ are both tridiagonal. Then, if the first columns of Q and V are equal, $q_1 = v_1$, then Q and V are essentially equal: $q_i = \pm v_i$, $i = 2, 3, \ldots, n$.*

For a proof, see [36, Chapter 8].

A consequence of this theorem is that if we determine and apply the first transformation in the QR decomposition of $T - \tau I$, and if we construct the rest of the transformations in such a way that we finally arrive at a tridiagonal matrix, then we have performed a shifted QR step as in Proposition 16.14. This procedure is implemented as follows.

Let the first plane rotation be determined such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} \alpha_1 - \tau \\ \beta_1 \end{pmatrix} = \begin{pmatrix} \times \\ 0 \end{pmatrix}, \tag{16.12}$$

where α_1 and β_1 are the top diagonal and subdiagonal elements of T. Define

$$G_1^T = \begin{pmatrix} c & s & & & \\ -s & c & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{pmatrix},$$

and apply the rotation to T. The multiplication from the left introduces a new non-zero element in the first row, and, correspondingly a new non-zero is introduced in the first column by the multiplication from the right:

$$G_1^T T G_1 = \begin{pmatrix} \times & \times & + & & & \\ \times & \times & \times & & & \\ + & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix},$$

where $+$ denotes a new non-zero element. We next determine a rotation in the (2,3)-plane that annihilates the new non-zero, and at the same time introduces a new non-zero further down:

$$G_2^T G_1^T T G_1 B_2 = \begin{pmatrix} \times & \times & 0 & & & \\ \times & \times & \times & + & & \\ 0 & \times & \times & \times & & \\ & + & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

In an analogous manner we "chase the bulge" downwards until we have

$$\begin{pmatrix} \times & \times & & & & \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & + \\ & & & \times & \times & \times \\ & & & + & \times & \times \end{pmatrix},$$

where by a final rotation we can zero the bulge and at the same time restore the tridiagonal form.

Note that it was only in the determination of the shift (16.12) that the shift was used. The rotations were only applied to the *unshifted* tridiagonal matrix. However, due to the Implicit QR Theorem 16.15, this is equivalent to a shifted QR step as given in Proposition 16.14.

16.4.2 Eigenvectors

The QR algorithm for computing the eigenvalues of a symmetric matrix (including the reduction to tridiagonal form) requires about $4n^3/3$ flops if only the eigenvalues are computed. Accumulation of the orthogonal transformations to compute the matrix of eigenvectors takes another $9n^3$ flops approximately.

If all n eigenvalues are needed but only a few of the eigenvectors, then it is cheaper to use inverse iteration (Section 16.2.1) to compute these eigenvectors, with the computed eigenvalues $\hat{\lambda}_i$ as shifts:

$$(A - \hat{\lambda}_i I)x^{(k)} = x^{(k-1)}, \qquad k = 1, 2, \ldots.$$

The eigenvalues produced by the QR algorithm are so close to the exact ones (see below) that usually only one step of inverse iteration is needed to get a very good eigenvector, even if the initial guess for the eigenvector is random.

The QR algorithm is ideal from the point of view of numerical stability. There exist an exactly orthogonal matrix Q and a perturbation E such that the computed diagonal matrix of eigenvalues $\hat{D}$ satisfies exactly

$$Q^T(A + E)Q = \hat{D},$$

with $\| E \|_2 \approx \mu \| A \|_2$, where μ is the unit round off of the floating point system. Then, from Theorem 16.2 we know that a computed eigenvalue $\hat{\lambda}_i$ differs from the exact one by a small amount: $\|\hat{\lambda}_i - \lambda_i\|_2 \leq \mu \| A \|_2$.

16.5 Computing the SVD

Since the singular values of a matrix A are the egenvalues squared of $A^T A$ and AA^T, it is clear that the problem of computing the SVD can be solved using similar algorithms as for the symmetric eigenvalue problem. However, it is important to avoid forming the matrices $A^T A$ and AA^T, since that would lead to loss of information (cf. also the least squares example on page 52).

Assume that A is $m \times n$ with $m \geq n$. The first step in computing the SVD of a dense matrix A is to reduce it to upper bidiagonal form by Householder transformations from the left and right,

$$A = H \begin{pmatrix} B \\ 0 \end{pmatrix} W^T, \qquad B = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & \alpha_n \end{pmatrix}. \tag{16.13}$$

For a description of this reduction, see Section 8.2.1. Since we use orthogonal transformations in this reduction, the matrix B has the same singular values as A: Let σ be a singular value of A with singular vectors u and v. Then $Av = \sigma u$, is equivalent to

$$B\tilde{v} = \sigma\tilde{u}, \qquad \tilde{v} = W^T v, \qquad \tilde{u} = H^T u,$$

from (16.13).

It is easy to see that the matrix $B^T B$ is tridiagonal. The method of choice for computing the singular values of B is the tridiagonal QR algorithm with implicit shifts applied to the matrix $B^T B$, without forming it explicitly.

Let $A \in \mathbb{R}^{m\times n}$, where $m \geq n$. The thin SVD $A = U_1 \Sigma V^T$ (cf. Section 6.1), can be computed in $6mn^2 + 20n^3$ flops.

16.6 The Non-Symmetric Eigenvalue Problem

If we perform the same procedure as in Section 16.3 on a non-symmetric matrix, then due to non-symmetry no elements above the diagonal are zeroed. Thus the final result is a *Hessenberg matrix*,

$$V^T AV = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{pmatrix}.$$

The reduction to Hessenberg form using Householder transformations requires $10n^3/3$ flops.

16.6.1 The QR Algorithm for Non-Symmetric Matrices

The "unrefined" QR algorithm for tridiagonal matrices given in Section 16.4, page 182, works equally well for non-symmetric matrices, and the result is an upper triangular matrix, i.e. is the R factor in the Schur decomposition. As in the symmetric case, for efficiency the QR decomposition in each step of the algorithm is computed using plane rotation, but here the transformation are applied to more elements.

We illustrate the procedure with a small example. Let the matrix $H \in \mathbb{R}^{6\times 6}$ be upper Hessenberg, and assume that a Wilkinson shift τ has been computed from the bottom right 2×2 matrix. For simplicity we assume that the shift is real. Denote $H^{(0)} := H$. The first subdiagonal element (from the top) in $H - \tau I$ is zeroed by a rotation from the left in the (1,2) plane, $G_1^T(H^{(0)} - \tau I)$, and then the second

subdiagonal is zeroed by a rotation in (2,3), $G_2^T G_1^T (H^{(0)} - \tau I)$. Symbolically,

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix} \longrightarrow \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ & 0 & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

After $n-1$ steps we have an upper triangular matrix:

$$R = G_{n-1}^T \cdots G_1^T (H^{(0)} - \tau I) = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ & 0 & \times & \times & \times & \times \\ & & 0 & \times & \times & \times \\ & & & 0 & \times & \times \\ & & & & 0 & \times \end{pmatrix}.$$

We then apply the rotations from the right, $RG_1 \cdots G_{n-1}$, i.e. we start with a transformation involving the first two columns. Then follows a rotation involving the second and third columns. The result after two steps is

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times & \times \\ & + & \times & \times & \times\times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \\ & & & & & \times \end{pmatrix}.$$

We see that the zeroes that we introduced in the transformations from the left are systematically filled in. After $n-1$ steps we have

$$H^{(1)} = RG_1G_2 \cdots G_{n-1} + \tau I = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times & \times \\ & + & \times & \times & \times & \times \\ & & + & \times & \times & \times \\ & & & + & \times & \times \\ & & & & + & \times \end{pmatrix}.$$

But we have made a similarity transformation: With $Q = G_1G_2 \cdots G_{n-1}$ and using $R = Q^T(H^{(0)} - \tau I)$, we can write

$$H^{(1)} = RQ + \tau I = Q^T(H^{(0)} - \tau I)Q + \tau I = Q^T H^{(0)} Q, \tag{16.14}$$

and we know that $H^{(1)}$ has the same eigenvalues as $H^{(0)}$.

The non-symmetric QR algorithm has almost as nice convergence properties as its symmetric counterpart.

Proposition 16.16. *The non-symmetric QR algorithm with Wilkinson shifts converges quadratically towards the Schur decomposition.*

As in the symmetric case there are numerous refinements of the algorithm sketched above, see e.g. [36, Chapter 7]. In particular one usually uses implicit double shifts to avoid complex arithmetic.

Given the eigenvalues, selected eigenvectors can be computed by inverse iteration with the upper Hessenberg matrix and the computed eigenvalues as shifts.

16.7 Sparse Matrices

In many applications a very small proportion of the elements of a matrix are non-zero. Then the matrix is called *sparse*. It is not unusual that less than 1% of the matrix elements are non-zero.

In the numerical solution of an eigenvalue problem for a sparse matrix, one often uses an iterative method for solving a linear system with this matrix, or computes the eigenvalues or singular values by an iterative method. This is due to the fact that the transformations to compact form described in Section 16.3 would completely destroy the sparsity, which leads to excessive storage requirements. In addition, the computational complexity of the reduction to compact form is often much too high, due to the dimension of the matrix.

In Sections 16.2 and 16.8 we describe a couple of methods for solving numerically the eigenvalue (and singular value) problem for a large sparse matrix. Here we give a brief description of one possible method for storing a sparse matrix.

In order to take advantage of sparseness the matrix only the non-zero elements should be stored. We describe briefly one possible storage scheme for sparse matrices, *compressed row storage*.

Example 16.17 Let

$$A = \begin{pmatrix} 0.6667 & 0 & 0 & 0.2887 \\ 0 & 0.7071 & 0.4082 & 0.2887 \\ 0.3333 & 0 & 0.4082 & 0.2887 \\ 0.6667 & 0 & 0 & 0 \end{pmatrix}$$

In compressed row storage the non-zero entries are stored in a vector, here called `val`, along with the corresponding column indices in a vector `colind` of equal length. The vector `rowptr` points to the positions in `val` that are occupied by the first element in each row. The matrix A is stored (we round the elements in the table in order to save space here)

val	0.67	0.29	0.71	0.41	0.29	0.33	0.41	0.29	0.67
colind	1	4	2	3	4	1	3	4	1
rowptr	1	3	6	9	10				

■

The compressed row storage scheme is convenient for multiplying $y = Ax$. The extra entry in the `rowptr` vector that points to the (nonexistent) position after the end of the `val` vector is used to make the code for multiplying $y = Ax$ simple.

Multiplication $y = Ax$ for sparse A

```
function y=Ax(val,colind,rowptr,x)
  % Compute y = A * x, with A in compressed row storage
  m=length(rowptr)-1;
  for i=1:m
    a=val(rowptr(i):rowptr(i+1)-1);
    y(i)=a*x(colind(rowptr(i):rowptr(i+1)-1));
  end
  y=y';
```

It can be seen that compressed row storage is inconvenient for multiplying $y = A^T z$. However, there is an analogous *compressed column storage* scheme that, naturally, is well suited for this.

Compressed row (column) storage for sparse matrices is relevant in programming languages like Fortran and C, where the programmer must handle the sparse storage explicitly. Matlab has a built-in (in an object-oriented way) storage scheme for sparse matrices, with overloaded matrix operations. For instance, for a sparse matrix `A`, the Matlab statement `y=A*x` implements sparse matrix-vector multiplication, and internally, Matlab executes a code analogous to the one above.

In a particular application, different sparse matrix storage schemes can influence the performance of matrix operations, depending on the structure of the matrix. In [34] a comparison is made of sparse matrix algorithms for information retrieval.

16.8 The Lanczos and Arnoldi Methods

Exercises

16.1. Let $N \in \mathbb{R}^{n\times n}$ be upper triangular with zeros on the diagonal. Prove that $N^{n-1} = 0$. (such a matrix is called *nilpotent*).

Chapter 17

Software

17.1 LAPACK

17.2 Programming Environments

Appendix A

Tensor Identities

Prove that

$$S_{(1)} = (U^{(1)})^T A_{(1)} (U^{(2)} \otimes U^{(3)}) = \Sigma^{(1)} (V^{(1)})^T (U^{(2)} \otimes U^{(3)})$$

and explain the Kronecker product.

Bibliography

[1] E. Anderson, Z. Bai, C. H. Bischof, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, S. Ostrouchov, and D. C. Sorensen. *LAPACK Users' Guide*. Soc. for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] ANSI/IEEE 754. *Binary Floating Point Arithmetic*. The Institute of Electric and Electronics Engineers, New York, 1985.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, Addison-Wesley, New York, 1999.

[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, 2000.

[5] R. Barrett, M. Berry, and T. Chan et al. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, Philadelphia, 1994.

[6] B. Bergeron. *Bioinformatics Computing*. Prentice Hall, 2002.

[7] P. Berkin. A survey on PageRank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[8] M. Berry, S. Dumais, and G. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995.

[9] M.J.A. Berry and G. Linoff. *Mastering Data Mining. The Art and Science of Customer Relationship Management*. John Wiley and Sons, New York, 2000.

[10] M.W. Berry, editor. *Computational Information Retrieval*. SIAM, Philadelphia, PA, 2001.

[11] M.W. Berry and M. Browne. *Understanding Search Engines. Mathematical Modeling and Text Retrieval*. SIAM, Philadelphia, PA, 1999.

[12] M.W. Berry and M. Browne. *Understanding Search Engines. Mathematical Modeling and Text Retrieval*. SIAM, Philadelphia, PA, second edition, 2005.

[13] Å. Björck. *Numerical Methods for Least Squares Problems*. Soc. for Industrial and Applied Mathematics, Philadelphia, PA, 1996.

[14] Å. Björck. The calculation of least squares problems. *Acta Numerical*, pages 1–51, 2004.

[15] Katarina Blom and Axel Ruhe. A Krylov subspace method for information retrieval. *SIAM J. Matrix Anal. Appl.*, 26:566–582, 2005.

[16] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. *SIAM Review*, 46:647–666, 2004.

[17] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[18] M.C. Burl, L. Asker, P. Smyth, U. Fayyad, P. Perona, L. Crumpler, and J. Aubele. Learning to recognize volcanoes on Venus. *Machine Learning*, 30:165–195, 1998.

[19] P.A. Businger and G.H. Golub. Linear least squares solutions by Householder transformations. *Numer. Math.*, 7:269–276, 1965.

[20] R. Chelappa, C.L. Wilson, and S. Sirohey. Human and machine recognition of faces: A survey. *Proceedings of the IEEE*, 83:705–740, 1995.

[21] N. Christianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[22] K.J. Cios, W. Pedrycz, and R.W. Swiniarski. *Data Mining. Methods for Knowledge Discovery*. Kluwer Academic Publishers, Boston, 1998.

[23] B. De Moor and P. Van Dooren. Generalizations of the singular value and QR decompositions. *SIAM J. Matrix Anal. Appl.*, 13:993–1014, 1992.

[24] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harsman. Indexing by latent semantic analysis. *J. Amer. Soc. Information Science*, 41:391–407, 1990.

[25] J.W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[26] I.S. Dhillon and D.S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42:143–175, 2001.

[27] R.O. Duda, P.E. Hart, and D.G. Storck. *Pattern Classification*. Wiley-Interscience, second edition, 2001.

[28] L. Eldén. Partial least squares vs. Lanczos bidiagonalization I: Analysis of a projection method for multiple regression. *Comp. Stat. Data Anal.*, 46:11–31, 2004.

[29] L. Eldén. Numerical linear algebra in data mining. *Acta Numerica*, 2006, to appear.

[30] L. Eldén, L. Wittmeyer-Koch, and H. Bruun Nielsen. *Introduction to Numerical Computation – Analysis and MATLAB Illustrations*. Studentlitteratur, Lund, 2004.

[31] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/The MIT Press, Menlo Park, CA, 1996.

[32] J.H. Fowler and S. Jeon. The authority of supreme court precedent: A network analysis. Technical report, Department of Political Science, UC Davis, 2005.

[33] J.T. Giles, L. Wo, and M.W. Berry. GTP (General Text Parser) software for text mining. In H. Bozdogan, editor, *Statistical Data Mining and Knowledge Discovery*, pages 455–471. CRC Press, Boca Raton, 2003.

[34] N. Goharian, A. Jain, and Q. Sun. Comparative analysis of sparse matrix algorithms for information retrieval. *Journal of Systemics, Cybernetics and Informatics*, 1(1), 2003.

[35] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Numer. Anal. Ser. B*, 2:205–224, 1965.

[36] G. H. Golub and C. F. Van Loan. *Matrix Computations. 3rd ed.* Johns Hopkins Press, Baltimore, MD., 1996.

[37] G.H. Golub and C. Greif. Arnoldi-type algorithms for computing stationary distribution vectors, with application to PageRank. Technical Report SCCM-04-15, Department of Computer Science, Stanford University, 2004.

[38] G.H. Golub, K. Sølna, and P. Van Dooren. Computing the SVD of a general matrix product/quotient. *SIAM J. Matrix Anal. Appl.*, 22:1–19, 2000.

[39] D. Grossman and O. Frieder. *Information Retrieval: Algorithms and Heuristics*. Kluwer Academic Press, 1998.

[40] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Databases*, pages 576–587. Morgan Kaufmann, 2004.

[41] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco, 2001.

[42] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001.

[43] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning. Data mining, Inference and Prediction*. Springer Verlag, New York, 2001.

[44] T.H. Haveliwala and S.D. Kamvar. An analytical comparison of approaches to personalizing PageRank. Technical report, Computer Science Department, Stanford University, 2003.

[45] M. Hegland. Data mining techniques. *Acta Numerica*, pages 313–355, 2001.

[46] N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996.

[47] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[48] Ilse C.F. Ipsen and Steve Kirkland. Convergence analysis of a Pagerank updating algorithm by Langville and Meyer. *SIAM J. Matrix Anal. Appl., to appear*, 2005.

[49] E.R. Jessup and J.H. Martin. Taking a new look at the latent semantic analysis approach to information retrieval. In M.W. Berry, editor, *Computational Information Retrieval*, pages 121–144, Philadelphia, PA, 2001. SIAM.

[50] S.D. Kamvar, T.H. Haveliwala, and G.H. Golub. Adaptive methods for the computation of pagerank. *Linear Algebra and its Applications*, 386:51–65, 2003.

[51] S.D. Kamvar, T.H. Haveliwala, C.D. Manning, and G.H. Golub. Exploiting the block structure of the Web for computing PageRank. Technical report, Computer Science Department, Stanford University, 2003.

[52] S.D. Kamvar, T.H. Haveliwala, C.D. Manning, and G.H. Golub. Extrapolation methods for accelerating PageRank computations. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[53] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[54] A. N. Langville and C. D. Meyer. A survey of eigenvector methods for web information retrieval. *SIAM Review*, 47:135–161, 2005.

[55] A. N. Langville and C. D. Meyer. *Understanding Web Search Engine Rankings: Google's PageRank, Teoma's HITS, and other ranking algorithms*. Princeton University Press, 2005.

[56] A.N Langville and C.D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 1:335–380, 2005.

[57] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 21:1253–1278, 2000.

[58] Y. LeCun, L. Bottou, Y. Bengiou, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, November 1998.

[59] R. Lempel and S. Moran. Salsa: the stochastic approach for link-structure analysis. *ACM Trans. Inf. Syst.*, 19:131–160, 2001.

[60] I. Mani. *Automatic Summarization*. John Benjamins Pub. Co, 2001.

[61] MathWorks. *Matlab User's Guide*. Mathworks Inc., Natick Mass., 1996.

[62] J. Mena. *Data Mining Your Website*. Digital Press, Boston, 1999.

[63] C.D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, Philadelphia, 2000.

[64] C. Moler. The world's largest matrix computation. *Matlab News and Notes*, pages 12–13, October 2002.

[65] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Stanford Digital Library Working Papers, 1998.

[66] C.C. Paige and M.A. Saunders. Towards a generalized singular value decomposition. *SIAM J. Numer. Anal.*, 18:398–405, 1981.

[67] H. Park, M. Jeon, and J. B. Rosen. Lower dimensional representation of text data in vector space based information retrieval. In M.W. Berry, editor, *Computational Information Retrieval*, pages 3–23, Philadelphia, PA, 2001. SIAM.

[68] H. Park, M. Jeon, and J. B. Rosen. Lower dimensional representation of text data based on centroids and least squares. *BIT*, 43:427–448, 2003.

[69] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, 1992.

[70] Y. Saad. *Iterative Methods for Sparse Linear Systems, second ed.* SIAM, 2003.

[71] G. Salton, C. Yang, and A. Wong. A vector-space model for automatic indexing. *Comm. ACM*, 18:613–620, 1975.

[72] B. Savas. Analyses and test of handwritten digit algorithms. Master's thesis, Mathematics Department, Linköping University, 2002.

[73] S. Serra-Capizzano. Jordan canonical form of the Google matrix: a potential contribution to the Pagerank computation. *SIAM J. Matrix Anal. Appl.*, pages 305–312, 2005.

[74] P. Simard, Y. Le Cun, and J. Denker. Efficient pattern recognition using a new transformation distance. In J.D. Cowan, S.J. Hanson, and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 50–58. Morgan Kaufmann, 1993.

[75] P.Y. Simard, Y.A. Le Cun, J.S. Denker, and B. Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. *Internat. J. Imaging System Techn.*, 11:181–194, 2001.

[76] L. Sirovich and M. Kirby. Low dimensional procedures for the characterization of human faces. *J. Optical Soc. Amer. A*, 4:519–524, 1987.

[77] M. Sjöström and S. Wold. *Pattern Recognition in Practice, SIMCA: A Pattern Recognition Method based on Principal Component Models.* North-Holland Publishing Comp., 1980.

[78] A. Smilde, R. Bro, and P. Geladi. *Multi-way Analysis: Applications in the Chemical Sciences.* Wiley, 2004.

[79] G. W. Stewart. *Matrix Algorithms I: Basic Decompositions.* SIAM, Philadelphia, 1998.

[80] G. W. Stewart. *Matrix Algorithms II: Eigensystems.* SIAM, Philadelphia, 2001.

[81] G. W. Stewart and J.-G. Sun. *Matrix Perturbation Theory.* Academic Press, Boston, 1990.

[82] J.B. Tenenbaum and W.T. Freeman. Separating style and content with bilinear models. *Neural Computation*, 12:1247–1283, 2000.

[83] M. Totty and M. Mangalindan. As Google becomes Web's gatekeeper, sites fight to get in. *Wall Street Journal*, CCXLI(39)(Feb. 26), 2003.

[84] L. N. Trefethen and D. B. Bau. *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[85] L.R. Tucker. The extension of factor analysis to three-dimensional matrices. In H. Gulliksen and N. Frederiksen, editors, *Contributions to Mathematical Psychology*, pages 109–127. Holt, Rinehart and Winston, New York, 1964.

[86] L.R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311, 1966.

[87] M.A. Turk and A.P. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3:71–86, 1991.

[88] G. van den Bergen. *Collision Detection in Interactive 3D Environments.* Morgan Kaufmann Publishers, 2004.

[89] C.F. Van Loan. Generalizing the singular value decomposition. *SIAM J. Numer. Anal.*, 13:76–83, 1976.

[90] M.A.O. Vasilescu. Human motion signatures: Analysis, synthesis, recognition. In *International Conference on Pattern Recognition (ICPR '02)*, Quebec City, Canada, 2002.

[91] M.A.O. Vasilescu and D. Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *Proc. 7th European Conference on Computer Vision (ECCV'02)*, Lecture Notes in Computer Science, Vol. 2350, pages 447–460, Copenhagen, Denmark, 2002. Springer Verlag.

[92] M.A.O. Vasilescu and D. Terzopoulos. Multilinear image analysis for facial recognition. In *International Conference on Pattern Recognition (ICPR '02)*, Quebec City, Canada, 2002.

[93] M.A.O. Vasilescu and D. Terzopoulos. Multilinear subspace analysis of image ensembles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'03)*, Madison WI, 2003.

[94] P.Å. Wedin. Perturbation theory for pseudoinverses. *BIT*, 13:344–354, 1973.

[95] J.H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, England, 1965.

[96] I.H. Witten and E. Frank. *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers, San Francisco, 2000.

[97] H. Wold. Soft modeling by latent variables; the nonlinear iterative partial least squares approach. In J. Gani, editor, *Perspectives in Probability and Statistics, Papers in honour of M.S. Bartlett*, chapter London. Academic Press, 1975.

[98] S. Wold, A. Ruhe, H. Wold, and W.J. Dunn. The collinearity problem in linear regression. The partial least squares (PLS) approach to generalized inverses. *SIAM J. Sci. Stat. Comput.*, 5:735–743, 1984.

[99] S. Wold, M. Sjöström, and L. Eriksson. PLS-regression: a basic tool of chemometrics. *Chemometrics and Intell. Lab. Systems*, 58:109–130, 2001.

[100] S. Wolfram. *The Mathematica Book, 4th ed.* Cambridge University Press, 1999.

[101] D. Zeimpekis and E. Gallopoulos. Design of a MATLAB toolbox for term-document matrix generation. In I.S. Dhillon, J. Kogan, and J. Ghosh, editors, *Proc. Workshop on Clustering High Dimensional Data and its Applications*, pages 38–48, Newport Beach, CA, 2005.

[102] H. Zha. Generic summarization and keyphrase extraction using mutual reinforcement principle and sentence clustering. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Tampere, Finland, 2002.

Index