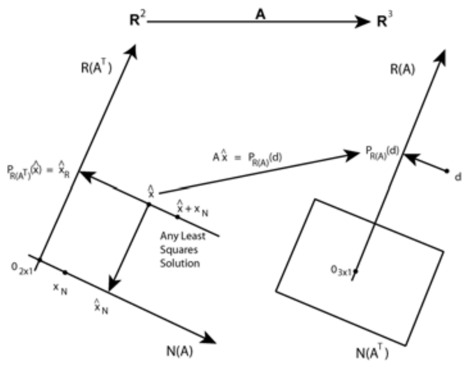
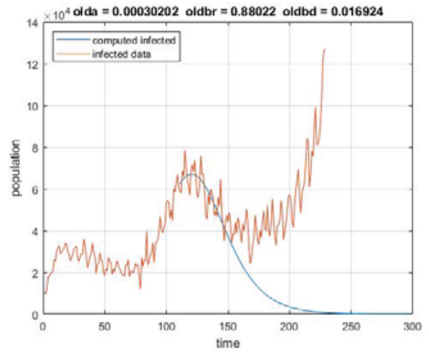
Singular Value Decomposition: An Introduction with Applications and MATLAB Computations

by

Robert E. White
Emeritus Professor of Mathematics
North Carolina State University
white@ncsu.edu
https://white.math.ncsu.edu

Draft Date is 11-15-2020





Contents

List of Figures

Preface

The singular value decomposition, $A = U\Sigma V^T$, is a deceptively simple statement. However it has very important components including least squares, fundamental subspaces, orthonormal bases and the spectral theorem. It has a number of theoretical topics, but with the advent of computation tools the SVD and its variations have an number of important applications.

An important challenge is to find time for this in an undergraduate matrix/linear algebra course. The following short chapters attempt one possible solution, which could be used as a supplement or as a special topics course. Another possibility is to create an additional course on matrices and followed by a more general vector space course; this would be a two-semester sequence on linear algebra.

Most applications involve a number of parameters and variables. The first two semesters of calculus are focused on functions of one variable. It is important to early introduce engineering/science students in their studies to more realistic applications. One way to do this is via an elementary matrix course, which could be taken concurrently with the second semester of calculus. A follow up course would have time for SVD, general vector and function spaces as well some significant applications.

These notes contain a partial description of several MATLAB codes which illustrate the mathematical concepts and applications. The full codes can be found at https://white.math.ncsu.edu/svdfiles/filename where filename is selected from:

- price_expdata.m
- qr_col.m
- svd_ex.m
- imagusa.m
  - letteru.m
  - letters.m
  - lettera.m
- svdimage.m
  - microchip.jpg
  - moon.jpg
  - pollen.jpg

sengine.m
senginesparse.m
    webmatrix_uri
Image1dsvd.m
    Kmatrix.m
    Setup1dsvd.m
hazidsvd1.m
levmarqprice.m
levmarqheat.m
sird_parid.m
    sirdid.m
    ypsirid.m
sird_paridc.m
    sirdidc.m
    ypsirdidc.m
sird_paridcuscovid2.m
    UScovid19.m
    sirdidc1.m
    ypsirdidc.m

Robert E. White, November 15, 2020

Introduction

The reader should be familiar with calculus of one variable, and basic matrix computations with row operations and inverse matrices. The first four chapters contain minimal additional materials that are needed to develop the singular value decomposition (SVD) of a $m \times n$ real matrix. These include the general least squares problem, the fundamental subspaces, basis and the eigenvectors of real symmetric matrices. The eigenvectors associated with the matrix $A^T A$ are used to construct the column vectors of V in the SVD $A = U\Sigma V^T$. The reader may find it useful to consult [14] for an introduction to matrices, [7] for a more comprehensive study of matrices/linear algebra, and [12] for a very popular treatment of matrix/linear algebra analysis.

The reader who has some knowledge of matrices, but lacks a standard course on linear algebra, should be able to master the first four chapters. Be careful to fill in some the details and any omitted proofs. The notes have been written using real and not complex numbers; a good exercise is to extend these results to complex numbers.

The last four chapters include existence of the SVD factors. Three variations of the SVD and pseudoinverse are discussed. Important applications to image compression, search engines, noise filters and hazard identifications are introduced.

Let A be a $m \times n$ real matrix and have rank$(A) = r$. The general or "full" form of the SVD of is

$$\begin{aligned} A &= U\Sigma V^T \\ &= [U_1\ U_2] \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}. \end{aligned}$$

where Σ_r is $r \times r$ nonsingular diagonal matrix. The columns of U_1, U_2, V_1 and V_2 are orthonormal bases for the fundamental subspaces $R(A), N(A^T), R(A^T)$ and $N(A)$, respectively. The "small" version is

$$A = U_1 \Sigma_r V_1^T.$$

The "truncated" version is the approximation

$$A \cong \Sigma_{k<r} U_1(:,k)\sigma_k V_1(:,k)^T.$$

In applications the matrix A may come from representations of an image, a frequency matrix associated with a search engine , or signal with noise. A forth application involves least squares problem from collected data at observation sites for an unknown hazard from source sites. The truncated SVDs are used as approximations of the matrix. Errors may appear because of the truncation or because of ill-conditioned problems and uncertain data. A fifth application is to parameter identification and epidemic models. Additional applications can be found in the current literature, for example, in [4] the SVD is used in artificial intelligence and machine learning.

There are a number of extensions. The vector spaces maybe defined over general fields and not just the real or complex numbers, see [6]. There are generalized SVD as given in section 8.7.3 of [5]. Moreover, the SVD can be extended to special Hilbert spaces and additional applications are given in [8].

Chapter 1

Least Squares and Normal Equations

The least squares solution(s) of $Ax = d$ are equivalent to finding the solution(s) of the normal equations $A^T Ax = A^T d$. If $A^T A$ is nonsingular, then there is a unique solution. An application to parameter identification is illustrated in the second section. The third and fourth sections construct multiple solutions. This is done by using the basis for the subspace $R(A)$ and the projection of d onto $R(A)$.

1.1 Normal Equations

The solution of the normal equations will be shown to be equivalent to finding the minimum of the residual. More precisely, these terms are defined as follows.

Definition 1 *Let A be $m \times n$. The least squares solution, x, of $Ax = d$ if and only if for $r(x) \equiv d - Ax$ and all y*

$$r(x)^T r(x) \leq r(y)^T r(y).$$

Definition 2 *The normal equations are $A^T Ax = A^T d$ or equivalently $A^T r(x) = 0_{n\times 1}$.*

Remark. This means the columns of A are perpendicular to $r(x)$.

Theorem 1.1.1 *x is a least squares solution of $Ax = d$ if and only if x is some solution of the normal equations.*

Proof. The proof follows from

$$\begin{aligned} r(y) &= d - Ay \\ &= d - A(x + y - x) \\ &= r(x) - A(y - x) \text{ and} \\ r(y)^T r(y) &= (r(x) - A(y - x))^T (r(x) - A(y - x)) \\ &= r(x)^T r(x) - 2(y - x)^T A^T r(x) + (A(y - x))^T A(y - x) \\ &\geq r(x)^T r(x) - 2(y - x)^T A^T r(x). \end{aligned}$$

If $A^T r(x) = 0_{n\times 1}$, then $r(y)^T r(y) \geq r(x)^T r(x)$.

If x is a least squares solution, then choose $y = x + e_i\epsilon$ where e_i a unit vector and $\epsilon > 0$. Use the least squares inequality and the above equality to get

$$\begin{aligned} 0 &\leq r(y)^T r(y) - r(x)^T r(x) \\ &= -2e_i^T \epsilon A^T r(x) + (Ae_i\epsilon)^T Ae_i\epsilon. \end{aligned}$$

Divide by $\epsilon > 0$

$$0 \leq -2e_i^T A^T r(x) + (Ae_i)^T (Ae_i)\epsilon.$$

Let $\epsilon \downarrow 0$ to get

$$0 \leq -2e_i^T A^T r(x) = -2[A^T r(x)]_i.$$

Likewise, choose $y = x + e_i(-\epsilon)$ to get the other inequality

$$0 \leq 2e_i^T A^T r(x) = 2[A^T r(x)]_i.$$

So, $A^T r(x) = 0_{n\times 1}$. ■

Theorem 1.1.2 *If A has full column rank ($Ax = 0_{m\times 1}$ implies $x = 0_{n\times 1}$), then $A^T A$ is nonsingular and the normal equations have a unique solution.*

Proof. $A^T Ax = 0_{n\times 1}$ implies $x^T (A^T Ax) = (Ax)^T (Ax) = 0$. Then $Ax = 0_{m\times 1}$ and the full column rank gives $x = 0_{n\times 1}$. So, the $n \times n$ matrix $A^T A$ is nonsingular. ■

Example 1 $A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$ *and* $d = \begin{bmatrix} 10 \\ 9 \\ 7 \end{bmatrix}$.

$$A^T A = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix} \text{ and } A^T d = \begin{bmatrix} 26 \\ 49 \end{bmatrix}.$$

$$x = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}^{-1} \begin{bmatrix} 26 \\ 49 \end{bmatrix} = \begin{bmatrix} 35/3 \\ -3/2 \end{bmatrix}.$$

Example 2 *Identify the parameters in the Newton cooling ($u(t)$ = temperature) or the Price decay ($p(t)$ = price) models:*

$$\begin{aligned} \frac{du}{dt} &= c(u_{sur} - u) \text{ or} \\ \frac{dp}{dt} &= c(p_{\min} - p). \end{aligned}$$

A discrete approximation of the Price model is

$$\frac{p^{k+1} - p^{k-1}}{2\Delta t} = (cp_{\min}) + (-c)p^k \textit{ where } p^k \simeq p(k\Delta t).$$

The unknown parameters are $(cp_{\min})$ and $(-c)$. If there are six data measurements for the past price, then A will be 4×2 with $x_1 = (-c)$ and $x_2 = (cp_{\min})$. The least squares problem is

$$\begin{bmatrix} p^2 & 1 \\ p^3 & 1 \\ p^4 & 1 \\ p^5 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{p^{2+1}-p^{2-1}}{2\Delta t} \\ \frac{p^{3+1}-p^{3-1}}{2\Delta t} \\ \frac{p^{4+1}-p^{4-1}}{2\Delta t} \\ \frac{p^{5+1}-p^{5-1}}{2\Delta t} \end{bmatrix}.$$

When this is solved, one can compute c and $p_{\min}$. The solution of the continuous price model has these values and an exponential function of time. Now predicted prices for future times can be done.

1.2 Matlab Code price_expdata.m

The code in this section is an implementation of the above example. The price data is given in line 8, and the matrix and column vectors are computed in lines 10-14. Solution of the normal equation is given in line 18. This is used in lines 19-21 to compute the two parameters and a future price. The numerical and graphical outputs are given in lines 25-31 and in Figure 1.2.1.

```
1      % Predict the price given more additional past prices.
2      % The exponential model is used.
3      % The method used is the normal equations.
4      %
5      % Input data
6      %
7      clear
8      price = [2080 2000 1950 1910 1875 1855];
9      time = 0:1:15;
10      for i = 2:5
11          d(i-1) = (price(i+1) - price(i-1))/2;
12      end
13      A = [price(2:5)' ones(4,1)]
14      d = d'
15      %
16      %  The normal equations are solved.
17      %
18      x = (A'*A)\(A'*d)               % x = A\d
19      c = -x(1)
```

```
pmin = x(2)/(-x(1))
future_price = pmin +(2080 - pmin)*exp(-c*time);
%
%  Output is in both numerical and graphical form.
%
plot(time(1:6),price,'*',time, future_price)
title('price data and predicted price curve')
display('Predicted price at time = 8')
future_price(9)
display('Residual vector')
r = price' - future_price(1:6)'
rTr = r'*r
```

```
>>    price_expdata

   A =
     2000           1
     1950           1
     1910           1
     1875           1

   d =
     -65.0000
     -45.0000
     -37.5000
     -27.5000

   x =
     -0.2920
    520.8994

   c =
     0.2920

   pmin =
     1.7839e+03

Predicted price at time = 8
   =
     1.8126e+03

Residual vector
   r =
           0
     -5.0238
```

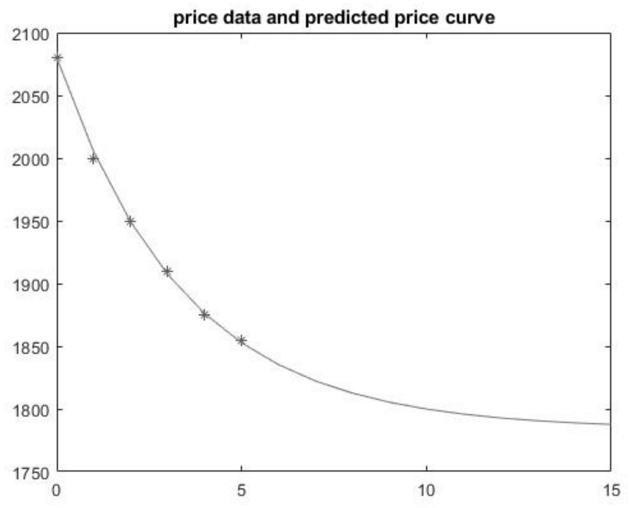


Figure 1.2.1: Parameter Identification Using Least Squares

```
     0.9662
     2.7779
    -0.9983
     2.3187

 rTr =
    40.2620
```

1.3 Basis of Subspace

In the least squares problem $Ax = d$ one can view the d to be the closest to $R(A) \equiv \{w : w = Ax \text{ and } x \in \mathbb{R}^n\}$. $R(A)$ is called the range of A, and it is a subspace of $\mathbb{R}^m$.

Definition 3 *Let $S \subset \mathbb{R}^m$ be any subspace. $\{w_1, .., w_k\}$ is called a basis of S if and only if*

(i). each $w \in S$ is a linear combination of the basis

$$w = c_1 w_1 + \cdots + c_k w_k \text{ and}$$

(ii). the set is linearly independent $c_1 w_1 + \cdots + c_k w_k = 0_{m\times1}$ implies

$$c_1 = \cdots = c_k = 0.$$

The k is called the dimension, and it is unique regardless of the basis.

Remark. Let W be the $m \times k$ matrix formed by the columns of a basis. Linear independence means $Wc = 0_{m\times 1}$ implies $c = 0_{k\times 1}$.

Example 3 *Let $S = \mathbb{R}^2$. Two bases are*

$$\begin{aligned} w_1 &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } w_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ w_1 &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } w_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}. \end{aligned}$$

Example 4 *The columns of the matrix A are linearly independent*

$$A = \begin{bmatrix} 2000 & 1 \\ 1950 & 1 \\ 1910 & 1 \\ 1875 & 1 \end{bmatrix}.$$

Theorem 1.3.1 *If $S \subset \mathbb{R}^m$ has a basis $\{w_1, .., w_k\}$, then*
1. each $w \in S$ has a unique linear combination and
2. $k = dim(S)$ is unique.

Proof. Consider the special case in the first item where $k = 2$. Let $w = c_1 w_1 + c_2 w_2$ and $w = \widehat{c}_1 w_1 + \widehat{c}_2 w_2$. Subtract these two

$$(c_1 - \widehat{c}_1)w_1 + (c_2 - \widehat{c}_2)w_2 = 0_{m\times 1}.$$

The linear independence implies $c_1 - \widehat{c}_1 = 0$ and $c_2 - \widehat{c}_2 = 0$. The general case is similar.

Consider the special case for the second item where $k = 2$ and $\widehat{k} = 3$ with bases $\{w_1, w_2\}$ and $\{\widehat{w}_1, \widehat{w}_2, \widehat{w}_3\}$. This will lead to a contradiction that $\widehat{w}_3$ is a linear combination of $\widehat{w}_1$ and $\widehat{w}_2$. Since $\{w_1, w_2\}$ is a basis,

$$\begin{aligned} \widehat{w}_1 &= c_{11}w_1 + c_{12}w_2 \\ \widehat{w}_2 &= c_{21}w_1 + c_{22}w_2 \text{ and} \\ \widehat{w}_3 &= c_{31}w_1 + c_{32}w_2. \end{aligned}$$

At least one of c_{11} and c_{12} is not zero, say $c_{11} \neq 0$. Then

$$\begin{aligned} w_1 &= \frac{\widehat{w}_1 - c_{12}w_2}{c_{11}} \text{ and} \\ \widehat{w}_2 &= \frac{c_{21}}{c_{11}}\widehat{w}_1 + \frac{-c_{21}c_{12} + c_{11}c_{22}}{c_{11}}w_2. \end{aligned}$$

If $-c_{21}c_{12} + c_{11}c_{22} = 0$, then $\widehat{w}_2 = \frac{c_{21}}{c_{11}}\widehat{w}_1$ contradicts the linear independence of $\{\widehat{w}_1, \widehat{w}_2, \widehat{w}_3\}$. If $-c_{21}c_{12} + c_{11}c_{22} \neq 0$, then one can solve w_2 as a linear combination of $\widehat{w}_1$ and $\widehat{w}_2$. So, both w_1 and w_2 are linear combinations of $\widehat{w}_1$ and $\widehat{w}_2$. From $\widehat{w}_3 = c_{31}w_1 + c_{32}w_2$, $\widehat{w}_3$ is also a linear combination of $\widehat{w}_1$ and

$\widehat{w}_2$. This contradicts the linear independence of $\{\widehat{w}_1, \widehat{w}_2, \widehat{w}_3\}$. The proof of the general case is similar.

Another variation of the above paragraph is to show there is a nontrivial solution $x \in \mathbb{R}^3$ such that

$$\begin{aligned} 0_{m\times 1} &= x_1\widehat{w}_1 + x_2\widehat{w}_2 + x_3\widehat{w}_3 \\ &= x_1\,(c_{11}w_1 + c_{12}w_2) \\ &\quad +x_2(c_{21}w_1 + c_{22}w_2) \\ &\quad +x_3(c_{31}w_1 + c_{32}w_2) \\ &= (x_1c_{11} + x_2c_{21} + x_3c_{31})w_1 \\ &\quad +(x_1c_{12} + x_2c_{22} + x_3c_{32})w_2. \end{aligned}$$

There exists a nontrivial solution to the following algebraic system with two equations and three unknowns

$$\begin{bmatrix} c_{11} & c_{21} & c_{31} \\ c_{12} & c_{22} & c_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

In the above paragraph, we argued the $(1,1)$ and $(2,2)$ pivots were not zero. ■

Example 5 *Consider a 3×2 matrix*

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}.$$

The two columns are linearly independent, and

$$R(A) = \left\{ c_1 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + c_2 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} : c_1, c_2 \in \mathbb{R} \right\}.$$

Example 6 *Consider a 3×3 matrix*

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \end{bmatrix}.$$

The third column is the sum of the first two columns. The first two columns are linearly independent, and they are a basis for $R(A)$.

Example 7 *Consider the least squares problem $Ax = d$ where*

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{bmatrix} \text{ and } d = \begin{bmatrix} 10 \\ 5 \\ 2 \end{bmatrix}.$$

The normal equations have multiple solutions

$$x = c\begin{bmatrix} 2 \\ -1 \end{bmatrix} + \begin{bmatrix} 26/14 \\ 0 \end{bmatrix}.$$

This follows from $A\begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$.

$$\begin{aligned} A^T(d - Ax) &= \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix}\left(\begin{bmatrix} 10 \\ 5 \\ 2 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - 26/14\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}\right) \\ &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \end{aligned}$$

We will return to this example in Sections 1.4, 3.2 and 7.1.

1.4 Projection to Subspace

Consider a two dimension subspace in $\mathbb{R}^3$. If a point $d \in \mathbb{R}^3$ is not in this subspace, then one can find a point in the subspace that is closest to d. The following is a generalization to $\mathbb{R}^m$. As we shall see this is related to the least squares problem.

Definition 4 *Let $d \in \mathbb{R}^m$ and $S \subset \mathbb{R}^m$ be a subspace with basis $\{w_1, .., w_k\}$. $P_S(d) \in S$ is called a projection of d onto S if and only if*

$$(d - P_S(d))^T(d - P_S(d)) \leq (d - w)^T(d - w) \text{ for all } w \in S.$$

Theorem 1.4.1 *Let $\{w_1, .., w_k\}$ be a basis for $S \subset \mathbb{R}^m$. $P_S(d) \in S$ if and only if $w_j^T(d - P_S(d)) = 0$ for all $j = 1, \cdots, k$. Moreover,*

1. *the projection is unique so that $P_S(d)$ is function,*
2. *$w^T(d - P_S(d)) = 0$ for all $w \in S$ and*
3. *$d^T d \geq P_S(d)^T P_S(d)$.*

Proof. Use $P_S(d)$ and w are a linear combination of the basis

$$\begin{aligned} P_S(d) &= \widehat{c}_1 w_1 + \cdots + \widehat{c}_k w_k \text{ and} \\ w &= c_1 w_1 + \cdots + c_k w_k. \end{aligned}$$

Let $W \equiv [w_1 \cdots w_k]$ be an $m \times k$ matrix. Then $P_S(d) = W\widehat{c}$, $w = Wc$ and the projection gives a least squares solution of $W\widehat{c} = d$

$$\begin{aligned} (d - P_S(d))^T(d - P_S(d)) &\leq (d - w)^T(d - w) \\ (d - W\widehat{c})^T(d - W\widehat{c}) &\leq (d - Wc)^T(d - Wc). \end{aligned}$$

Because $\{w_1, .., w_k\}$ is a basis, W has full column rank and W^TW is nonsingular. In this case the normal equations $W^T(d - Wc) = 0_{k\times 1}$ have a unique solution.

The proof of the inequality follows from $P_S(d) \in S$ and $P_S(d)^T(d-P_S(d)) = 0$. Write $d = (d - P_S(d)) + P_S(d)$ to get

$$\begin{aligned} d^T d &= (d - P_S(d))^T(d - P_S(d)) + \\ &\quad 2P_S(d)^T(d - P_S(d)) + P_S(d)^T P_S(d) \\ &= (d - P_S(d))^T(d - P_S(d)) + 0 + P_S(d)^T P_S(d) \\ &\geq P_S(d)^T P_S(d). \end{aligned}$$

■

The projection of d onto $S = R(A)$ can be used to identify all least squares solutions.

Theorem 1.4.2 *Let A be $m \times n$ and $S = R(A)$ be the range of A. Then $x = \widehat{x} + x_N$ are least squares solutions to $Ax = d$ where $P_{R(A)}(d) \in R(A)$, $A\widehat{x} = P_{R(A)}(d)$ and $Ax_N = 0_{m\times 1}$.*

Proof. Since $P_{R(A)}(d) \in R(A)$, there exists $\widehat{x}$ such that $A\widehat{x} = P_{R(A)}(d)$. Because $P_{R(A)}(d)$ is a projection, for all $w \in R(A)$ with $w = Ay$

$$\begin{aligned} (d - P_{R(A)}(d))^T(d - P_{R(A)}(d)) &\leq (d - w)^T(d - w) \\ (d - A\widehat{x})^T(d - A\widehat{x}) &\leq (d - Ay)^T(d - Ay). \end{aligned}$$

So, $\widehat{x}$ is a least squares solution and must satisfy the normal equations. Because $Ax_N = 0_{m\times 1}$, $\widehat{x} + x_N$ must also satisfy the normal equations

$$A^T A(\widehat{x} + x_N) = A^T(A\widehat{x} + 0_{m\times 1}) = A^T d.$$

■

Example 8 *Revisit Example 7 where*

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{bmatrix} \text{ and } R(A) = \{c \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} : c \in \mathbb{R}\}.$$

Here $m = 3, n = 2$ and $k = 1$ with

$$w_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \text{ and } d = \begin{bmatrix} 10 \\ 5 \\ 2 \end{bmatrix}.$$

Require $w_1^T(d - P_{R(A)}(d)) = 0$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}^T (\begin{bmatrix} 10 \\ 5 \\ 2 \end{bmatrix} - c \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}) = 0.$$

Then $c = 26/14$ *and*

$$\begin{aligned} A\widehat{x} &= P_{R(A)}(d) \\ \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{bmatrix} \begin{bmatrix} 26/14 \\ 0 \end{bmatrix} &= 26/14 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}. \end{aligned}$$

Choose $\widehat{x} = \begin{bmatrix} 26/14 \\ 0 \end{bmatrix}$ *and* $x_N = \widehat{c} \begin{bmatrix} 2 \\ -1 \end{bmatrix}$.

Chapter 2

Fundamental Subspaces $N(A)$ and $R(A)$

Elementary matrices (row operations) will be used to construct the basis for the nullspace, $N(A)$, and the range space, $R(A)$. This construction will first be illustrated for two examples. The general case is described in the second section. These bases are not orthonormal, but they still are useful in the general least squares problem.

2.1 Examples and Bases

Definition 1 *Let A be an $m \times n$ matrix and view it as a mapping from $\mathbb{R}^n$ into $\mathbb{R}^m$.*

$$\begin{aligned} N(A) &\equiv \{x \in \mathbb{R}^n : Ax = 0_{m\times 1}\} \\ & \quad N(A) \textit{ is subspace in } \mathbb{R}^n \\ & \quad \textit{and is called a nullspace or a kernel of } A \end{aligned}$$

$$\begin{aligned} R(A) &\equiv \{y \in \mathbb{R}^m : Ax = y \textit{ where } x \in \mathbb{R}^n\} \\ & \quad R(A) \textit{ is a subspace in } \mathbb{R}^m \\ & \quad \textit{and is called a range or a column space of } A. \end{aligned}$$

Example 1 $A = \begin{bmatrix} 1 & 1 & 2 & 3 \\ 2 & 2 & 8 & 10 \\ 3 & 3 & 10 & 13 \end{bmatrix}$ *with $m = 3$ and $n = 4$.*

$Ax = d$ may be rewritten in compact form as an augmented matrix

$$[A \quad d] = \begin{bmatrix} 1 & 1 & 2 & 3 & d_1 \\ 2 & 2 & 8 & 10 & d_2 \\ 3 & 3 & 10 & 13 & d_3 \end{bmatrix}.$$

Use elementary matrices (row operations) to transform it to row echelon form

$$\begin{aligned} E_{31}(-3)E_{21}(-2)\,[A \;\; d] &= \begin{bmatrix} 1 & 1 & 2 & 3 & d_1 \\ 0 & 0 & 4 & 4 & d_2 - 2d_1 \\ 0 & 0 & 4 & 4 & d_3 - 3d_1 \end{bmatrix} \\ E_{32}(-1)E_{31}(-3)E_{21}(-2)\,[A \;\; d] &= \begin{bmatrix} 1 & 1 & 2 & 3 & d_1 \\ 0 & 0 & 4 & 4 & d_2 - 2d_1 \\ 0 & 0 & 0 & 0 & d_3 - d_1 - d_2 \end{bmatrix}. \end{aligned}$$

The pivots are at $(1,1)$ and $(2,3)$, the fixed variables are x_1, x_3 and the free variables are x_2, x_4. Let E be the product of the elementary matrices

$$E = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & -1 & 1 \end{bmatrix}.$$

$E\,[A \;\; d] = [EA \;\; Ed]$ or $EAx = Ux = Ed$. If $[Ed]_3 = 0$, then $Ux = Ed$ has a solution. Because E has an inverse, $Ux = Ed$ and $Ax = d$ have the same solution.

Definition 2 *rank$(A) \equiv$ number of pivots $= r$.*
r is number of fixed variables.
$n - r$ is the number of free variables.

In the above example $r = 2$. Use the notation $ipiv(1) = 1, jpiv(1) = 1$ for the first pivot and $ipiv(2) = 2, jpiv(2) = 3$ for the second pivot.

Find $R(A)$ by setting the free variables to zero and solving $Ux = Ed$

$$\begin{bmatrix} 1 & 2 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 - d_1 \end{bmatrix}.$$

Use the pivot columns of A in $E^{-1}Ux = Ax = E^{-1}Ed = d$.

$$R(A) = \left\{ y : y = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} x_1 + \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} x_3 \right\}.$$

Since the first and third columns are linearly independent, they are a basis and $\dim(R(A)) = 2$.

Find $N(A)$ by using the free variables. Solve the equivalent $Ax = 0_{3\times1}$ and $EAx = Ux = 0_{3\times1}$. Solve the pivot variables in terms of the free variables

$$\begin{aligned} \begin{bmatrix} 1 & 2 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} &= \begin{bmatrix} -x_2 - 3x_4 \\ -4x_4 \end{bmatrix} \text{ and} \\ \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} &= \begin{bmatrix} -1 \\ 0 \end{bmatrix} x_2 + \begin{bmatrix} -1 \\ -1 \end{bmatrix} x_4. \end{aligned}$$

$$N(A) = \left\{ x : x = \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \end{bmatrix} x_2 + \begin{bmatrix} -1 \\ 0 \\ -1 \\ 1 \end{bmatrix} x_4 \right\}.$$

Since the columns are linearly independent, they form a basis and $\dim(N(A)) = 4 - 2 = 2$.

Example 2 $A^T = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 2 & 8 & 10 \\ 3 & 10 & 13 \end{bmatrix}$. *Solve $A^T y = g$ by using elementary row operations to form $\widehat{E}A^T = \widehat{U}$ and then solve $\widehat{U}y = \widehat{E}g$.*

$$\begin{aligned} \widehat{E}A^T &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ -1 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & g_1 \\ 1 & 2 & 3 & g_2 \\ 2 & 8 & 10 & g_3 \\ 3 & 10 & 13 & g_4 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 2 & 3 & g_1 \\ 0 & 0 & 0 & -g_1 + g_2 \\ 0 & 4 & 4 & -2g_2 + g_3 \\ 0 & 0 & 0 & -g_1 - g_3 + g_4 \end{bmatrix}. \end{aligned}$$

The pivots are $(1,1)$ and $(3,2)$, the fixed variables are y_1, y_2 and the free variable is y_3. In order to solve $\widehat{U}y = \widehat{E}g$, the second and fourth components of $\widehat{E}g$ must be zero. The rank$(A^T) = \widehat{r} = 2$, and use the notation $ipiv(1) = 1, jpiv(1) = 1$ for the first pivot and $ipiv(2) = 3, jpiv(2) = 2$ for the second pivot.

Find $R(A^T)$ by the pivot columns of A^T. Set the free variable $y_3 = 0$ and solve

$$\begin{bmatrix} 1 & 2 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} g_1 \\ -2g_2 + g_3 \end{bmatrix}.$$

This solves

$$\widehat{E}A^T \begin{bmatrix} y_1 \\ y_2 \\ 0 \end{bmatrix} = \widehat{E}g = \begin{bmatrix} g_1 \\ 0 \\ -2g_2 + g_3 \\ 0 \end{bmatrix},$$

and because $\widehat{E}$ is nonsingular

$$\begin{aligned} A^T \begin{bmatrix} y_1 \\ y_2 \\ 0 \end{bmatrix} &= g \\ \begin{bmatrix} 1 \\ 1 \\ 2 \\ 3 \end{bmatrix} y_1 + \begin{bmatrix} 2 \\ 2 \\ 8 \\ 10 \end{bmatrix} y_2 &= g. \end{aligned}$$

The first two columns of A^T are linearly independent and form a basis for

$$R(A^T) = \left\{ w : w = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 3 \end{bmatrix} y_1 + \begin{bmatrix} 2 \\ 2 \\ 8 \\ 10 \end{bmatrix} y_2 \right\}.$$

Find $N(A^T)$ by solving $\widehat{U}y = 0_{4\times 1}$. Find the fixed variables y_1, y_2 in terms of the free variable y_3

$$\begin{bmatrix} 1 & 2 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} y_3 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

This gives $y_2 = -y_3$ and $y_1 = -y_3$. The nullspace has dimension equal to one and

$$N(A^T) = \left\{ y : y = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} y_3 \right\}.$$

Remark 1. $EA = U$ implies

$$A^T E^T = U^T = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 4 & 0 \\ 3 & 4 & 0 \end{bmatrix}.$$

So, the third column of E^T must be in the nullspace of A^T, which agrees with the second example.

Remark 2. $\widehat{E}A^T = \widehat{U}$ implies

$$A\widehat{E}^T = \widehat{U}^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 0 & 4 & 0 \\ 3 & 0 & 4 & 0 \end{bmatrix}.$$

So, the second and fourth columns of $\widehat{E}^T$ must be in the nullspace of A, and this agrees with the first example.

Remark 3. In the first example we required $d_3 - d_1 - d_2 = 0$. This can be written as

$$\begin{bmatrix} d_1 & d_2 & d_1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} = 0.$$

So, d is perpendicular to any element in $N(A^T)$.

Remark 4. In the second example we required $-g_1 + g_2 = 0$ and $-g_1 - g_2 + g_4 = 0$, that is,

$$g^T \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = 0 \text{ and } g^T \begin{bmatrix} -1 \\ 0 \\ -1 \\ 1 \end{bmatrix} = 0.$$

This means g must be perpendicular to any element in $N(A)$.

2.2 Construction of Bases

Consider the general $m \times n$ matrix A and use elementary matrices to find the row echelon form $E[A \;\; d] = [EA \;\; Ed] = [U \;\; Ed]$. Since E is nonsingular, $Ax = d$ and $Ux = Ed$ are equivalent. If the non fixed variable rows of $[U \;\; Ed]$ are all zeros, then $Ux = Ed$ has a solution.

Let $(ipiv, jpiv)$ be the row and column numbers of the pivots where $1 \leq k \leq r = rank(A)$ and

$$\begin{aligned} ipiv(k) &= \text{row of pivot } k \text{ and} \\ jpiv(k) &= \text{column of pivot } k. \end{aligned}$$

The fixed variables are $x_{jpiv(k)}$, and the free variables are x_j where $j \neq jpiv(k)$ with $1 \leq k \leq r$.

Theorem 2.2.1 *Let A be $m \times n$ with rank$(A) = r$, and E be a product of elementary matrices that give the row echelon form. If d is $m \times 1$ and $[Ed]_i = 0$ for all $i \neq ipiv(k)$ where $1 \leq k \leq r$, then*

1. *$d \in R(A)$.*
2. *$R(A)$ has a basis from the pivot columns of A and dim$(R(A))$ = rank$(A) = r$.*
3. *$N(A)$ has a basis by solving the fixed variables in terms of the free variables and dim$(N(A)) = n - r$.*

Proof. Since the non fixed variable rows of $[U \;\; Ed]$ are all zeros, $Ux = Ed$ has a solution and this solution also solves $Ax = d$. This means $d \in R(A)$. $Ux = Ed$ is solved by setting the free variables to zero. This means the d in $Ax = d$ is a linear combination of the r pivot columns of A. These pivot columns are linearly independent; if $d = 0_{m \times 1}$, then both the fixed and free variables are zero. Thus, $\dim(R(A)) = r$.

Find $N(A)$ by solving the $r \times r$ upper triangular system for the fixed variables, $x_{fixed} = x_{jpiv}$, in terms of the free variables, $x_{free} = x_j$ with $j \neq jpiv(k)$.

$$U(ipiv, jpiv)x_{fixed} + U(ipiv, free)x_{free} = 0_{r \times 1}.$$

$$\begin{aligned} x_{fixed} &= -U(ipiv, jpiv)^{-1}U(ipiv, free)x_{free} \\ &= -Cx_{free} \text{ and} \\ x &= \begin{bmatrix} -C \\ I_{n-r} \end{bmatrix} x_{free}. \end{aligned}$$

So, the columns of $\begin{bmatrix} -C \\ I_{n-r} \end{bmatrix}$ are linearly independent and $\dim(N(A)) = n - r$.
■

Theorem 2.2.2 *Let A be $m \times n$ with rank$(A) = r$. Then rank$(A^T) = r$, that is, rank$(A^T) =$ rank(A).*

Proof. Let rank$(A^T) = \widehat{r}$. Since A^T is $n \times m$ and by Theorem 2.2.1, $\dim(R(A^T)) = \widehat{r}$ and $\dim(N(A^T)) = m - \widehat{r}$. Let $EA = U$ have $m - r$ zero rows with rank$(A) = r$. Then $A^T E^T = U^T$ has $m - r$ zero columns and $\dim(N(A^T)) = m - r$. By Theorem 1.3.1 $\dim(N(A^T))$ is unique so that $m - r = m - \widehat{r}$ and $r = \widehat{r}$. ■

Chapter 3

Perpendicular Subspaces and Bases

If A is $m \times n$, then $A : \mathbb{R}^n \to \mathbb{R}^m$. Any vector in $\mathbb{R}^n$ may be written as a sum of vectors in $N(A)$ and in $R(A^T)$. Moreover these two vectors are perpendicular, which is part of the fundamental theorem. The bases for these two subspaces may have orthonormal vectors. This is established by using the factors $A = QR$ where the columns of Q are orthonormal and R is upper triangular.

3.1 Perpendicular Subspaces

Consider $\mathbb{R}^3$ and let W be the xy-plane. All the vectors in $\mathbb{R}^3$ that are perpendicular to the xy-plane are multiples of the unit vector in the z-direction. This is generalized by the following.

Definition 1 *Let W be a vector space in $\mathbb{R}^n$. The perpendicular or orthogonal complement is*

$$W^{\perp} \equiv \left\{ y \in \mathbb{R}^n : y^T x = 0 \textit{ for all } x \in W \right\}$$

Example 1 *Consider $\mathbb{R}^3$ and* $W = \left\{ c_1 \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + c_2 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} : c_1,\, c_2 \in \mathbb{R} \right\}$. *The cross product of the two column vectors is perpendicular to both vectors. Then*

$$W^{\perp} = \left\{ c \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} : c \in \mathbb{R} \right\}.$$

If $n = 2$ or 3, then $x^T y = \|x\|_2 \, \|y\|_2 \cos(\theta)$ where θ is the angle between the two vectors. A generalization of the "angle" between two vectors $x, y \in \mathbb{R}^n$ is

$$\cos(\theta) \equiv \frac{x^T y}{\|x\|_2 \, \|y\|_2} \text{ where } \|x\|_2 = (x^T x)^{1/2}.$$

This is based on the Cauchy inequality.

Definition 2 *The Cauchy Inequality is*

$$\left|x^T y\right| \leq \|x\|_2 \, \|y\|_2 \, .$$

The inequality follows from the quadratic function of the real parameter

$$f(t) \equiv (x+ty)^T(x+ty).$$

Let $t_0 = \frac{-x^T y}{y^T y}$, which gives the minimum of $f(t)$,

$$0 \leq f(t_0) = x^T x - \frac{(x^T y)^2}{y^T y}.$$

Then $(x^T y)^2 \leq x^T x \;\; y^T y = \|x\|_2^2 \, \|y\|_2^2$ and

$$-1 \leq \frac{x^T y}{\|x\|_2 \, \|y\|_2} \leq 1.$$

Hence, the expression x is perpendicular to y means $x^T y = 0$.

Theorem 3.1.1 *Let W be a vector space in $\mathbb{R}^n$. The following hold:*

1. $W^{\perp}$ *is a subspace of* $\mathbb{R}^n$,
2. $W^{\perp\perp} = W$,
3. $W^{\perp} \cap W = \{0_{n\times 1}\}$ *and*
4. $W^{\perp} + W = \mathbb{R}^n$.

Proof. The proof of the first item requires the $W^{\perp}$ to be closed under addition and scalar multiplication. If $y, \widehat{y} \in W^{\perp}$, then

$$(y+\widehat{y})^T x = y^T x + \widehat{y}^T x = 0 + 0 = 0.$$

If $y \in W^{\perp}$ and $c \in \mathbb{R}$, then

$$(cy)^T x = c(y^T x) = c0 = 0.$$

The proofs of the second and third items are easy.

The proof of the last item uses the projection of any $v \in \mathbb{R}^n$ into W, $P_W(v)$. By Theorem 1.4.1 for all $w \in W$

$$w^T(-P_W(v) + v) = 0.$$

This means $-P_W(v) + v \in W^{\perp}$ and $v = (v - P_W(v)) + P_W(v) \in W^{\perp} + W$. ∎

3.2 Fundamental Theorem: $\mathbb{R}^n = N(A) + R(A^T)$

An illustration of the next theorem is given in Section 2.1 in Examples 1 and 2 and Remarks 3 and 4. In the general case, A is $m \times n$ and $W = N(A) \subset \mathbb{R}^n$. The fourth part of Theorem 3.1.1 gives $\mathbb{R}^n = N(A) + N(A)^\perp$.

Theorem 3.2.1 *If $A : \mathbb{R}^n \to \mathbb{R}^m$ with rank$(A) = r$, then $A^T : \mathbb{R}^m \to \mathbb{R}^n$ and*

$$\begin{array}{ll} N(A) = R(A^T)^\perp & N(A^T) = R(A)^\perp, \\ \mathbb{R}^n = N(A) + R(A^T) & \mathbb{R}^m = N(A^T) + R(A), \\ \mathit{dim}(R(A^T)) = r & \mathit{dim}(R(A)) = r \text{ and} \\ \mathit{dim}(N(A)) = n - r & \mathit{dim}(N(A^T)) = m - r. \end{array}$$

Proof. The main step is to show $N(A) = (R(A^T))^\perp$. Suppose $x \in N(A)$ and show $x \in (R(A^T))^\perp$.

$$\begin{aligned} Ax &= 0_{n\times 1} \\ y^T(Ax) &= y^T 0_{n\times 1} \text{ for all } y \in \mathbb{R}^m \\ (A^T y)^T x &= 0. \end{aligned}$$

Suppose $x \in (R(A^T))^\perp$ and show $x \in N(A)$. Then for all $y \in \mathbb{R}^m$ $(A^T y)^T x = y^T(Ax) = 0$. Choose $y = e_i \in \mathbb{R}^m$ to be any unit vector to get

$$\begin{aligned} e_i^T(Ax) &= 0, \\ [Ax]_i &= 0 \text{ and consequently} \\ Ax &= 0_{n\times 1}. \end{aligned}$$

Apply the above argument to A^T to get $N(A^T) = R(A)^\perp$. The remaining conclusions for both A and A^T follow from Theorems 2.2.1, 2.2.2 and 3.1.1. ■

The fundamental theorem can be used to analyze least squares problems with multiple solutions. Theorem 1.4.2 showed multiple least squares solution have the form $x = \widehat{x} + x_N$ where $A\widehat{x} = P_{R(A)}(d)$ and any $x_N \in N(A)$. The above Theorem 3.2.1 gives $\widehat{x} = \widehat{x}_R + \widehat{x}_N$ with $\widehat{x}_R = P_{R(A^T)}(\widehat{x}) \in R(A^T)$ and $\widehat{x}_N \in N(A)$.

Theorem 3.2.2 *If $A\widehat{x} = P_{R(A)}(d)$, then $\widehat{x}_R = P_{R(A^T)}(\widehat{x})$ is a least squares solution of $Ax = d$ and $(\widehat{x}_R)^T\widehat{x}_R$ is the smallest of least squares solutions.*

Proof. $\widehat{x} + x_N = \widehat{x}_R + \widehat{x}_N + x_N$ is a least squares solution. Choose $x_N = -\widehat{x}_N$ to conclude $\widehat{x}_R$ is a least squares solution. By Theorem 3.2.1 $\widehat{x}_R^T\widehat{x}_N = 0$,

$$\begin{aligned} \widehat{x}^T\widehat{x} &= (\widehat{x}_R + \widehat{x}_N)^T(\widehat{x}_R + \widehat{x}_N) \\ &= \widehat{x}_R^T\widehat{x}_R + 2\widehat{x}_R^T\widehat{x}_N + \widehat{x}_N^T\widehat{x}_N \\ &= \widehat{x}_R^T\widehat{x}_R + 0 + \widehat{x}_N^T\widehat{x}_N \\ &\geq \widehat{x}_R^T\widehat{x}_R. \end{aligned}$$

■

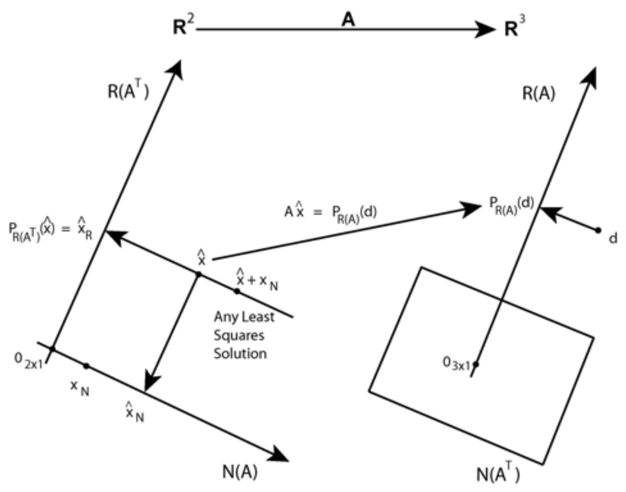


Figure 3.2.1: All Least Squares Solutions

Example 2 *Return to Examples 7 and 8 in Sections 1.3 and 1.4 and also Section 7.1.* $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{bmatrix}$, $A^T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix}$ *and* $\hat{x} = \begin{bmatrix} 26/14 \\ 0 \end{bmatrix}$. *The basis for* $R(A^T)$ *is* $w_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

$$\begin{aligned} \hat{x}_R &= c\begin{bmatrix} 1 \\ 2 \end{bmatrix} = P_{R(A^T)}(\hat{x}) \\ \begin{bmatrix} 1 & 2 \end{bmatrix} c \begin{bmatrix} 1 \\ 2 \end{bmatrix} &= \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 26/14 \\ 0 \end{bmatrix} \end{aligned}$$

This gives $c = 13/35$ *and* $\hat{x}_R$

$$\hat{x}_R = \begin{bmatrix} 13/35 \\ 26/35 \end{bmatrix}.$$

In Example 8 the general solution is

$$\hat{x} = \begin{bmatrix} 26/14 \\ 0 \end{bmatrix} + \hat{c}\begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

Choose $\hat{c}$ *so that* $\hat{x}^T\hat{x}$ *is a minimum to get* $\hat{x}_R$. *The set of all least squares solutions is depicted in Figure 3.2.1.*

3.3 A = QR Factorization

Any basis can be transformed into an orthonormal basis, whose vectors are perpendicular and unit length. This can be extended to general inner product spaces via the Gram-Schmidt method. Here we will use the QR factors of A. The QR factors are also useful in efficiently solving the normal equations.

Definition 3 *Let A be $m \times n$. $A = QR$ is a QR factorization if and only if*
(i). Q is $m \times n$ with $Q^TQ = I_n$ and
(ii). R is $n \times n$ with R being upper triangular.

The columns of Q are orthonormal. If the diagonal of R is not zero, then R is nonsingular and $AR^{-1} = Q$. In the next section, this will be useful in construction of an orthonormal basis from a given basis.

Theorem 3.3.1 *Let A be $m \times n$. If A has full column rank, rank(A) = n , then*
1. R^{-1} exists and
2. the normal equation reduces to $Rx = Q^Td$.

Proof. If $Rx = 0_{n\times 1}$, then

$$\begin{aligned} Q(Rx) &= Q0_{n\times 1} \text{ and} \\ Ax &= 0_{m\times 1}. \end{aligned}$$

Because A has full column rank, $x = 0_{n\times 1}$ and R is nonsingular.
Consider the normal equations.

$$\begin{aligned} A^TAx &= A^Td \\ (QR)^T(QR)x &= (QR)^Tx \\ R^T(Q^TQ)Rx &= R^TQ^Td \\ R^TRx &= R^TQ^Td. \end{aligned}$$

Because R is nonsingular, R^T is also nonsingular and $Rx = Q^Td$. ■

In order to find the QR factors, write $A = QR$ as a sequence of equal column vectors

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix}.$$

$$\begin{aligned} a_1 &= q_1r_{11} \\ a_2 &= q_1r_{12} + q_2r_{22} \\ a_3 &= q_1r_{13} + q_2r_{23} + q_3r_{33} \end{aligned}$$

We can find q_1 and r_{11} and then r_{12} and r_{13} as follows

$$\begin{aligned} a_1^T a_1 &= q_1^T q_1 r_{11}^2 = 1 r_{11}^2 \\ a_2^T q_1 &= (q_1 r_{12} + q_2 r_{22})^T q_1 \\ &= 1 r_{12} + 0 r_{22} \text{ and} \\ a_3^T q_1 &= (q_1 r_{13} + q_2 r_{23} + q_3 r_{33})^T q_1 \\ &= 1 r_{13} + 0 r_{23} + 0 r_{33}. \end{aligned}$$

Now reduce the dimension from m to $m-1$ by moving the first column to the left side

$$\begin{aligned} a_1 - q_1 r_{11} &= 0 \\ a_2 - q_1 r_{12} &= q_2 r_{22} \\ a_3 - q_1 r_{13} &= q_2 r_{23} + q_3 r_{33}. \end{aligned}$$

This procedure can be programmed as is illustrated in the next section.

Example 3 *Return to Example 1 in Section 1.1 where*

$$\begin{aligned} A &= \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \textit{ and } d = \begin{bmatrix} 10 \\ 9 \\ 7 \end{bmatrix}. \\ a_1 &= \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \textit{ gives } q_1 = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix} \textit{ and } r_{11} = \sqrt{3} \\ a_2 &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \textit{ gives } r_{12} = a_2^T q_1 = 6/\sqrt{3} \\ \widehat{q}_2 &= a_2 - q_1 r_{12} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix} 6/\sqrt{3} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \\ q_2 &= \begin{bmatrix} -1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix} \textit{ and } r_{22} = \sqrt{2}. \end{aligned}$$

The QR factors and the normal equations are

$$\begin{aligned} A &= QR \\ &= \begin{bmatrix} 1/\sqrt{3} & -1/\sqrt{2} \\ 1/\sqrt{3} & 0 \\ 1/\sqrt{3} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} \sqrt{3} & 6/\sqrt{3} \\ 0 & \sqrt{2} \end{bmatrix} \textit{ and} \\ Rx &= Q^T f \\ \begin{bmatrix} \sqrt{3} & 6/\sqrt{3} \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 10 \\ 9 \\ 7 \end{bmatrix} = \begin{bmatrix} 26/\sqrt{3} \\ -3/\sqrt{2} \end{bmatrix}. \end{aligned}$$

The solution of this is $x_2 = -3/2$ *and* $x_1 = 35/3$, *which agrees with the calculation in Example 1.*

3.4 Matlab Code qr_col.m

The sample inputs are given in lines 18-24. Note the matrix in line 20 will give an error because the columns are not linearly independent. The algorithm is executed in lines 25-35. This is the column version where the columns are moved to the left side in the loop starting at line 31. Lines 38-43 give the output, verify the factorization and compare the calculation with the two MATLAB intrinsic commands.

```
1    %  This code finds the qr factorization of a matrix.
2    %  The column version of the Gram-Schmidt method is
3    %  used to generate the small qr factors.
4    %  a is mxn, q is mxn and r is nxn where m>n and
5    %           q'*q = eye(n)
6    %
7    %  [a1 a2 a3 ...] = [q1 q2 q2 ...][r11 r12 r13 ...]
8    %                                     r22 r23 ...]
9    %                                         r33 ...]
10   %  OR
11   %
12   %  a1 = q1 r11
13   %  a2 = q1 r12 + q2 r22
14   %  a3 = q1 r13 + q2 r23 + q3 r33
15   %
16   clear
17   %
18   %  Input data
19   %
20   % a = [1 1 3; 1 2 4 ; 1 3 5; 1 4 6]
21   % Above fails because a3 = 2 a1 + a2!
22   a = [1 1;1 2; 1 3]
23   q = a;
24   [m n] = size(a)
25   %
26   %  Execute the column version
27   %
28   for k = 1:n
29       r(k,k) = (q(:,k)'*q(:,k))^.5;
30       q(:,k) = q(:,k)/r(k,k);
31       for j = k+1:n
32           r(k,j) = a(:,j)'*q(:,k);
33           q(:,j) = q(:,j) - q(:,k)*r(k,j);
```

```
       end
   end
   %
   %  Output Q R factors and compare with qr()
   %
   q
   r
   q*r
   [Q R] = qr(a)       % fat version
   [Q1 R1] = qr(a,0)  % skinny version
```

```
>> qr_col

   a =
     1     1
     1     2
     1     3

   m =
     3

   n =
     2

   q =
     0.5774   -0.7071
     0.5774   -0.0000
     0.5774    0.7071

   r =
     1.7321    3.4641
          0    1.4142

   A =
     1     1
     1     2
     1     3

   Q =
     -0.5774    0.7071    0.4082
     -0.5774   -0.0000   -0.8165
     -0.5774   -0.7071    0.4082

   R =
     -1.7321   -3.4641
```

```
         0   -1.4142
         0         0

Q1 =
   -0.5774    0.7071
   -0.5774   -0.0000
   -0.5774   -0.7071

R1 =
   -1.7321   -3.4641
         0   -1.4142
```

3.5 Orthonormal Basis

The QR factorization can be used to generate an orthonormal basis.

Definition 4 *A basis $\{w_1, .., w_k\}$ in $\mathbb{R}^m$ is orthonormal if and and if*
(i). $w_i^T w_i = 1$ *for all* $1 \le i \le k$ *and*
(ii). $w_i^T w_j = 0$ *when* $i \ne j$.

Remark. Let $W = \begin{bmatrix} w_1 & \cdots & w_k \end{bmatrix}$ be an $m \times k$ matrix. $W^T W = I_k$ means the columns are orthonormal.

Theorem 3.5.1 *Any basis of an k dimensional subspace $V \subset \mathbb{R}^m$ can be used to generate an orthonormal basis of V. Moreover, this basis can be extended to an orthonormal basis of $\mathbb{R}^m$.*

Proof. Let A be an $m \times k$ matrix formed by the column vectors of the basis for $V \subset \mathbb{R}^m$. Since the basis vectors are linearly independent, A has full column rank and $A = QR$ where R is nonsingular. We claim the columns of Q are an orthonormal basis. First, note the columns of Q are linearly independent. Let

$$Qx = (AR^{-1})x = A(R^{-1}x) = 0_{k\times 1}.$$

Since A has full column rank, $R^{-1}x = 0_{k\times 1}$. Then $x = 0_{k\times 1}$ and the columns are linearly independent. Second, the columns span the subspace $V \subset \mathbb{R}^m$. Let $d \in V = R(A)$ and

$$Q(Rx) = d \text{ and choose } \widehat{x} = R(x).$$

$Q(\widehat{x}) = d$ means d is a linear combination of the columns of Q. ∎

Example 4 *Consider the 3×4 matrix considered in Examples 1 and 2 in Section 2.1*

$$A = \begin{bmatrix} 1 & 1 & 2 & 3 \\ 2 & 2 & 8 & 10 \\ 3 & 3 & 10 & 13 \end{bmatrix}.$$

$$\begin{aligned} R(A) &= \left\{ c_1 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + c_2 \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} : c_1, c_2 \in \mathbb{R} \right\} \\ N(A^T) &= \left\{ c \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} : c \in \mathbb{R} \right\} \end{aligned}$$

Find the QR factorization of the 3×2 matrix from the basis of $R(A)$

$$\begin{aligned} \begin{bmatrix} a_1 & a_2 \end{bmatrix} &= \begin{bmatrix} 1 & 2 \\ 2 & 8 \\ 3 & 10 \end{bmatrix} \\ &= \begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{bmatrix} \\ &= \begin{bmatrix} 1/\sqrt{14} & -10/\sqrt{168} \\ 2/\sqrt{14} & 8/\sqrt{168} \\ 3/\sqrt{14} & -2/\sqrt{168} \end{bmatrix} \begin{bmatrix} \sqrt{14} & 48/\sqrt{14} \\ 0 & \sqrt{168}/7 \end{bmatrix}. \end{aligned}$$

The orthonormal basis is q_1, q_2. In order to extend this to all of $\mathbb{R}^3 = R(A) + N(A^T)$, choose $a_3^T = \begin{bmatrix} -1 & -1 & 1 \end{bmatrix}$. Since this is already perpendicular to both q_1, q_2, and we only need to normalize it to get a third basis vector

$$\begin{bmatrix} -1/\sqrt{3} \\ -1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix}.$$

If a_3 is not perpendicular to both q_1, q_2, then continue with the QR procedure

$$a_3 = q_1 r_{13} + q_2 r_{23} + q_3 r_{33}.$$

For example, suppose $a_3^T = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ and compute

$$\begin{aligned} a_3^T q_1 &= r_{13} = 3/\sqrt{14} \\ a_3^T q_2 &= r_{23} = -2/\sqrt{168} \\ \widehat{q}_3 &= a_3 - q_1 r_{13} - q_2 r_{23} \\ &= \begin{bmatrix} -1/3 \\ -1/3 \\ 1/3 \end{bmatrix} \quad \textit{with } r_{33} = 1/\sqrt{3} \textit{ and} \\ q_3 &= \begin{bmatrix} -1/\sqrt{3} \\ -1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix}. \end{aligned}$$

Chapter 4

Eigenvectors and Orthonormal Basis

Another way to find orthonormal vectors is to consider eigenvectors of a symmetric matrix. An important case is $A^T A$ in the normal equations, and this will form the core the singular value decomposition of an $m \times n$ matrix $A = U\Sigma V^T$. For this chapter the spectral theorem for real symmetric $n \times n$ matrices gives $A = UDU^T$ where D is the diagonal of eigenvalues and the columns of Q are the corresponding eigenvectors.

4.1 Eigenvectors of Symmetric Matrix

Let $A = [a_{ij}]$ be an $n \times n$ real symmetric matrix ($a_{ij} = a_{ji}$ for $1 \leq i, j \leq n$, or $A^T = A$). Eigenvectors play an important role in generating orthonormal bases. The matrix in least squares problem generally are not symmetric. However, the $A^T A$ in the normal equations is because $(A^T A)^T = A^T A^{TT} = A^T A$. Another important class of symmetric matrices come from differential equation. For example, consider $-u_{xx} = f$ on the interval $[0 \quad L]$ with boundary conditions $u(0) = 0 = u(L)$. A finite difference approximation for $u_i \cong u(i\Delta x)$ with $n = 4$ and $\Delta x = L/n$ is

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \Delta x^2 \begin{bmatrix} f(u_1) \\ f(u_2) \\ f(u_3) \end{bmatrix}.$$

Definition 1 *Let A be an $n \times n$ matrix. The eigenvector, $x \in \mathbb{C}^n$, associated with an eigenvalue, $\lambda \in \mathbb{C}$, is a nonzero vector such that $Ax = \lambda x$.*

An eigenvector is an element of the nullspace of $A - \lambda I_n$

$$\begin{aligned} Ax - \lambda x &= 0_{n\times 1} \\ Ax - \lambda I_n x &= 0_{n\times 1} \\ (A - \lambda I_n)x &= 0_{n\times 1}. \end{aligned}$$

Since x is a nonzero vector, $\det(A - \lambda I_n) = 0$. This allows us to solve for the eigenvalues and then the corresponding solutions of $(A - \lambda I_n)x = 0_{n\times 1}$.

Example 1 *Let* $A = \begin{bmatrix} 1 & -2 \\ 1 & 3 \end{bmatrix}$. *This matrix is not symmetric and has complex numbers for eigenvalues.*

$$\begin{aligned} \det(A - \lambda I_2) &= \det(\begin{bmatrix} 1-\lambda & -2 \\ 1 & 3-\lambda \end{bmatrix}) \\ &= \lambda^2 - 4\lambda + 5 = 0 \end{aligned}$$

and $\lambda = 2 + i, 2 - i$ *with* $i \equiv \sqrt{-1}$.

Example 2 *Let* $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$. *This matrix is symmetric and does not have an inverse.*

$$\det(A - \lambda I_2) = \det(\begin{bmatrix} 1-\lambda & 2 \\ 2 & 4-\lambda \end{bmatrix} = \lambda^2 - 5\lambda = 0$$

and $\lambda = 0, 5$.

Find the eigenvector associated with $\lambda = 0$.

$$\begin{aligned} (A - 0I_2)x &= 0_2 \\ \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ gives} \\ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 2 \\ -1 \end{bmatrix}. \end{aligned}$$

Find the eigenvector associated with $\lambda = 5$.

$$\begin{aligned} (A - 5I_2)x &= 0_2 \\ \begin{bmatrix} -4 & 2 \\ 2 & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ gives} \\ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 1 \\ 2 \end{bmatrix}. \end{aligned}$$

Note the eigenvalues are real, distinct and the eigenvectors are perpendicular. Since a nonzero multiple of an eigenvector is also an eigenvector, we normalize the eigenvectors to get an orthonormal basis of $\mathbb{R}^2$

$$q_1 = \begin{bmatrix} 2/\sqrt{5} \\ -1/\sqrt{5} \end{bmatrix} \text{ and } q_2 = \begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}.$$

Theorem 4.1.1 *If A is symmetric and has real components, then*
1. *all the eigenvalues are real and*
2. *eigenvectors with different eigenvalues are orthogonal.*

Proof. Let $\lambda \in \mathbb{C}$, $x \in \mathbb{C}^n$ and $Ax = \lambda x$. Use the notation

$$x^* \equiv \overline{x}^T \text{ and } A^* \equiv \overline{A}^T = [\overline{a}_{ji}].$$

A real and symmetric means $\overline{a}_{ji} = a_{ij}$ so that $A^* = A$.

In order to show λ is real, show $\lambda = \overline{\lambda}$. $Ax = \lambda x$ implies $x^* Ax = \lambda x^* x$. Also, $\overline{Ax} = \overline{\lambda x}$, $A\overline{x} = \overline{\lambda}\overline{x}$ and $\overline{x}^T A = \overline{\lambda}\overline{x}^T$. Hence, $x^* Ax = \overline{\lambda} x^* x$. Since x is a nonzero vector, $x^* x \neq 0$ and $\lambda = \overline{\lambda}$.

In order to show eigenvectors with different eigenvalues must be orthogonal, assume $Ax = \lambda x$, $A\widehat{x} = \widehat{\lambda}\widehat{x}$ and $\lambda \neq \widehat{\lambda}$.

$$\begin{aligned} \widehat{x}^T Ax &= \widehat{x}^T \lambda x = \lambda \widehat{x}^T x \\ x^T A\widehat{x} &= x^T \widehat{\lambda}\widehat{x} = \widehat{\lambda}\widehat{x}^T x \end{aligned}$$

Since A is symmetric, $x^T A\widehat{x} = x^T A^T \widehat{x} = (Ax)^T \widehat{x} = \widehat{x}^T Ax$. This means $\lambda \widehat{x}^T x = \widehat{\lambda}\widehat{x}^T x$ and $(\lambda - \widehat{\lambda})\widehat{x}^T x = 0$. Because $\lambda \neq \widehat{\lambda}$, $\widehat{x}^T x = 0$. ■

4.2 Spectral Theorem Factors $AQ = QD$

Example 2 in the previous section shows there exists two orthonormal eigenvectors such that

$$Aq_1 = 0q_1 \text{ and } Aq_2 = 5q_2.$$

The matrix form of this is

$$\begin{aligned} A\begin{bmatrix} q_1 & q_2 \end{bmatrix} &= \begin{bmatrix} q_1 & q_2 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 0 & 5 \end{bmatrix} \text{ with} \\ \begin{bmatrix} q_1 & q_2 \end{bmatrix}^T \begin{bmatrix} q_1 & q_2 \end{bmatrix} &= \begin{bmatrix} q_1^T q_1 & q_1^T q_2 \\ q_2^T q_1 & q_2^T q_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \end{aligned}$$

So, this 2×2 matrix has the form $A = QDQ^T$ where Q is the matrix of the orthonormal eigenvectors and the D is the diagonal matrix of the corresponding eigenvalues. We shall show this is true for any symmetric $n \times n$ matrix.

Definition 2 *Let A be an $n \times n$ matrix. A is called diagonalizable if only if there exists an orthonormal $n \times n$ matrix Q such that*

$$A = QDQ^T \textit{ and } D \textit{ is an } n \times n \textit{ diagonal matrix.}$$

The following proof is does not construct the eigenvectors, but it does show the existence of the desired eigenvectors. Numerical linear algebra is used to find the eigenvectors, and the MATLAB command eig(A) is a good implementation of these methods.

Theorem 4.2.1 *All real symmetric $n \times n$ matrices are diagonalizable.*

Proof. Mathematical induction on n will be used to prove this. Choose one eigenvector x where $Ax = \lambda x$ and $q_1 \equiv x/(x^T x)^{1/2}$. By Theorem 3.5.1 extend this to an orthonormal basis for all of $\mathbb{R}^n$

$$q_2, \cdots, q_n \text{ with } q_i^T q_i = 1,\ q_i^T q_j = 0 \text{ and } i \neq j.$$

Let $Q \equiv \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}$ so that

$$Q^T A Q = \begin{bmatrix} \lambda & q_1^T A q_2 & \cdots & q_1^T A q_n \\ q_2^T \lambda q_1 & q_2^T A q_2 & \cdots & q_2^T A q_n \\ \vdots & \vdots & \ddots & \vdots \\ q_n^T \lambda q_1 & q_n^T A q_2 & \cdots & q_n^T A q_n \end{bmatrix}.$$

Since $A = A^T$, $q_1^T A q_j = (A q_1)^T q_j = (\lambda q_1)^T q_j = 0$ for $2 \leq j \leq n$,

$$Q^T A Q = \begin{bmatrix} \lambda & 0_{1 \times (n-1)} \\ 0_{(n-1) \times 1} & \widehat{A} \end{bmatrix}.$$

Since $A = A^T$, the $(n-1) \times (n-1)$ matrix $\widehat{A}$ is also symmetric.

If $n = 2$, then $\widehat{A} = q_2^T A q_2$ and

$$Q^T A Q = \begin{bmatrix} \lambda & 0_{1 \times (n-1)} \\ 0_{(n-1) \times 1} & q_2^T A q_2 \end{bmatrix}.$$

If the proposition is true for $(n-1) \times (n-1)$ matrices, apply it to $\widehat{A}$

$$\begin{aligned} \widehat{Q}^T \widehat{A} \widehat{Q} &= \widehat{D}. \text{ Then} \\ Q^T A Q &= \begin{bmatrix} \lambda & 0_{1 \times (n-1)} \\ 0_{(n-1) \times 1} & \widehat{Q}^T \widehat{D} \widehat{Q} \end{bmatrix} \text{ and for } J \equiv \begin{bmatrix} 1 & 0_{1 \times (n-1)} \\ 0_{(n-1) \times 1} & \widehat{Q}^T \end{bmatrix} \\ &= J \begin{bmatrix} \lambda & 0_{1 \times (n-1)} \\ 0_{(n-1) \times 1} & \widehat{D} \end{bmatrix} J^T. \end{aligned}$$

Let $\widetilde{Q} \equiv QJ$ and note it is an orthonormal $n \times n$ matrix and satisfies

$$\widetilde{Q}^T A \widetilde{Q} = \begin{bmatrix} \lambda & 0_{1 \times (n-1)} \\ 0_{(n-1) \times 1} & \widehat{D} \end{bmatrix}.$$

■

4.3 Application to Nonsingular $Ax = d$

Eigenvectors are very useful in a number of problems. In this section they will be used to find solution of algebraic problems where there n unknowns, n

equations and the associated matrix is nonsingular. Provided one knows the eigenvectors and eigenvalues, this requires very few computations relative to Gauss elimination. This technique also generalizes to self adjoint boundary value problems.

Assume the matrix in the algebraic problem $Ax = d$ is nonsingular, real and symmetric. Then the eigenvalues must be real and nonzero. Choose the orthonormal eigenvectors and eigenvalues

$$Aq_i = \lambda_i q_i \text{ with } 1 \leq i \leq n \text{ and } \lambda_i \neq 0.$$

Since $d \in \mathbb{R}^n$ and the eigenvectors are a basis,

$$d = \sum_{j=1}^{n} c_j q_j.$$

Since the eigenvectors are orthonormal,

$$q_i^T d = q_i^T \sum_{j=1}^{n} c_j q_j = \sum_{j=1}^{n} c_j q_i^T q_j = c_i.$$

The solution must also be a linear combination of the eigenvectors

$$\begin{aligned} x &= \sum_{j=1}^{n} \widehat{c}_j q_j. \\ Ax &= A\sum_{j=1}^{n} \widehat{c}_j q_j \\ &= \sum_{j=1}^{n} \widehat{c}_j A q_j \\ &= \sum_{j=1}^{n} \widehat{c}_j \lambda_j q_j. \end{aligned}$$

Since $d = \sum_{j=1}^{n} c_j q_j = \sum_{j=1}^{n} (q_j^T d) q_j$ and by the linear independence, $q_j^T d = \widehat{c}_j \lambda_j$. Since $\lambda_j \neq 0$, $\widehat{c}_j = q_j^T d / \lambda_j$.

Not counting the cost of finding the eigenvalues, this requires n inner products, $q_j^T d$, and n divisions. This is in contrast to Gauss elimination, which requires about $n^3/3$ operations. If one has a large sequence of algebraic problems all with the same matrix and and number of right hand sides, then the the eigenvectors and eigenvalues only need to be found once. Also, for some special matrices, these may be easily found.

Chapter 5

Singular Value Decomposition

In this chapter we consider the more general $m \times n$ real matrix A and its factorization $U\Sigma V^T$. The factors satisfy the following: U is $m \times m$, $U^TU = I_m$, V is $n \times n$, $V^TV = I_n$ and Σ is $m \times n$. This is the "full" SVD. First, we establish the "small" version. Then the "full" and "truncated" versions are described.

Definition 1 *The ("full") SVD factorization of A is $A = U\Sigma V^T$ where*

$$\begin{aligned} U \text{ is } m \times m, U^TU &= I_m, \\ V \text{ is } n \times n, V^TV &= I_n, \\ \Sigma \text{ is } m \times n \text{ where } \Sigma &= \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix} \text{ and} \\ \Sigma_r &= \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \text{ with } \sigma_1 \geq \cdots \geq \sigma_r > 0. \end{aligned}$$

5.1 "Small" SVD

Assume $\text{rank}(A) = r$ with $U_1 \equiv U(1:m, 1:r)$ $m \times r$ and $V_1 \equiv U(1:n, 1:r)$ $n \times r$ matrices. The "small" SVD is

$$A = U_1\Sigma_r V_1^T$$

where $U_1^TU_1 = I_r$, $V_1^TV_1 = I_r$ and Σ_r is an $r \times r$ is diagonal matrix with positive diagonal components. The existence of these factors will be established in Theorem 5.1.1 and uses the existence of orthonormal eigenvectors for symmetric matrices via Theorem 4.2.1.

The columns of U_1 are a basis for $R(A)$. This follows from $AV_1 = U_1\Sigma_r$ and Σ_r^{-1}

$$AV_1\Sigma_r^{-1} = U_1.$$

So the columns of U_1 are in $R(A)$. The columns are orthonormal, $\dim(R(A)) = r$ are a basis of $R(A)$. Likewise, $A^TU_1\Sigma_r^{-1} = V_1$ and the columns of V_1 are an orthonormal basis of $R(A^T)$ and $\dim(R(A^T)) = r$.

Theorem 5.1.1 *If A is $m \times n$ with rank$(A) = r$, then U_1, V_1 and Σ_r exists where*

$$\begin{aligned} A &= U_1\Sigma_r V_1^T, \\ U_1 \textit{ is } m \times r,\ U_1^TU_1 &= I_r, \\ V_1 \textit{ is } n \times r,\ V_1^TV_1 &= I_r \textit{ and} \\ \Sigma_r &= r \times r \textit{ nonsingular diagonal matrix.} \end{aligned}$$

Remark. Multiply $A = U_1\Sigma_r V_1^T$ on the right by V_1 to get $AV_1 = U_1\Sigma_r$. In other words, $Av_i = u_i\sigma_i$ where v_i are $n \times 1$ with $V_1 = \begin{bmatrix} v_1 & \cdots & v_r \end{bmatrix}$ and u_i are $m \times 1$ with $U_1 = \begin{bmatrix} u_1 & \cdots & u_r \end{bmatrix}$.

Proof. A^TA is a real symmetric $n \times n$ matrix. By Theorem 4.2.1 there are orthonormal eigenvectors. Let v be anyone of these $A^TAv = \lambda v$ where $v \neq 0_{n\times 1}$ and $\lambda \neq 0$. Then

$$\begin{aligned} A(A^TAv) &= A\lambda v \text{ and} \\ (AA^T)Av &= \lambda Av. \end{aligned}$$

Note $v^Tv = 1$ and $A^TAv = \lambda v$ implies

$$\begin{aligned} v^TA^TAv &= \lambda v^Tv \text{ and} \\ (Av)^T(Av) &= \lambda \neq 0. \end{aligned}$$

Thus, $\lambda > 0, Av \neq 0_{m\times 1}$ and Av is an eigenvector of AA^T. Write $\lambda_i = \sigma_i^2$ and $(Av_i)^T(Av_i) = \sigma_i^2$. Let V_1 have columns of the orthonormal eigenvectors of A^TA.

Define the columns of U_1, u_i, from the columns of Av_i as follows

$$u_i \equiv \frac{Av_i}{\sigma_i}.$$

u_i are eigenvalues of AA^T:

$$\begin{aligned} A^TAv_i &= \sigma_i^2 v_i, \\ A(A^TAv_i) &= A(\sigma_i^2 v_i), \\ AA^T(Av_i) &= \sigma_i^2(Av_i) \text{ and} \\ AA^T(\frac{Av_i}{\sigma_i}) &= \sigma_i^2(\frac{Av_i}{\sigma_i}). \end{aligned}$$

u_i are unit vectors:

$$\begin{aligned} A^TAv_i &= \sigma_i^2 v_i, \\ v_i^T(A^TAv_i) &= v_i^T(\sigma_i^2 v_i), \\ (Av_i)^T(Av_i) &= \sigma_i^2 \text{ and} \\ (\frac{Av_i}{\sigma_i})^T(\frac{Av_i}{\sigma_i}) &= 1. \end{aligned}$$

u_i are orthogonal:

$$\begin{aligned} u_i^T u_j &= (\frac{Av_i}{\sigma_i})^T(\frac{Av_j}{\sigma_j}), \\ &= \frac{v_i^TA^TAv_j}{\sigma_i\sigma_j}, \\ &= \frac{v_i^T(A^TAv_j)}{\sigma_i\sigma_j}, \\ &= \frac{v_i^T(\sigma_j^2 v_j)}{\sigma_i\sigma_j} \text{ and for } i \neq j \\ &= 0. \end{aligned}$$

Define the matrices V_1 and U_1 where $1 \le i, j \le r$

$$V_1 \equiv \left[\begin{array}{ccc} v_1 & \cdots & v_r \end{array}\right] \text{ and } U_1 \equiv \left[\begin{array}{ccc} u_1 & \cdots & u_r \end{array}\right].$$

Since $Av_i = u_i\sigma_i$, $AV_1 = U_1\Sigma_r$ and $A = U_1\Sigma_r V_1^T$. ■

Example 1 $A = \left[\begin{array}{cc} 2 & 2 \\ -1 & 1 \end{array}\right]$ *where* $rank(A) = 2, m = 2$ *and* $n = 2$. $N(A) = \{0_{2\times 1}\}$ *and* $N(A^T) = \{0_{2\times 1}\}$.

$$A^TA = \left[\begin{array}{cc} 5 & 3 \\ 3 & 5 \end{array}\right] \text{ and } det(\left[\begin{array}{cc} 5-\lambda & 3 \\ 3 & 5-\lambda \end{array}\right]) = 0.$$

This gives $\lambda_1 = 8, \sigma_1 = 2\sqrt{2}$ *and* $\lambda_2 = 2, \sigma_2 = \sqrt{2}$ *so that*

$$v_1 = \left[\begin{array}{c} 1 \\ 1 \end{array}\right]/\sqrt{2} \text{ and } v_2 = \left[\begin{array}{c} 1 \\ -1 \end{array}\right]/\sqrt{2}.$$

$$u_1 = \frac{Av_1}{\sigma_1} = \left[\begin{array}{c} 1 \\ 0 \end{array}\right] \text{ and } u_2 = \frac{Av_2}{\sigma_2} = \left[\begin{array}{c} 0 \\ -1 \end{array}\right].$$

$$\begin{aligned} U_1\Sigma_2V_1^T &= \left[\begin{array}{cc} 1 & 0 \\ 0 & -1 \end{array}\right]\left[\begin{array}{cc} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{array}\right]\left[\begin{array}{cc} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{array}\right] \\ &= \left[\begin{array}{cc} 2 & 2 \\ -1 & 1 \end{array}\right] = A. \end{aligned}$$

5.2 "Full" SVD

The "full" SVD $A = U\Sigma V^T$ has orthonormal matrices $m \times m$ U, $n \times n$ V and $m \times n$

$$\Sigma = \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix}.$$

The "full" SVD version requires the augmentation of $m \times r$ U_1, $n \times r$ V_1 in the "small" SVD.

Theorem 5.2.1 *If A is $m \times n$ with rank$(A) = r$, then the "full" SVD exists.*

Proof. In the "small" SVD the columns in U_1 and V_1 form bases of $R(A)$ and $R(A^T)$, respectively. By Theorem 3.2.1 $\mathbb{R}^n = R(A^T) + N(A)$ and $\mathbb{R}^m = R(A) + N(A^T)$. By Theorem 3.5.1 we can extend the orthonormal bases of $R(A^T)$ and $R(A)$ to all of $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. Let U be the orthonormal augmentation of U_1 so that the the columns of $U(1:m,(r+1):m)$ are an orthonormal basis of $N(A^T)$. Likewise, let V be the orthonormal augmentation of V_1 so that the the columns of $V(1:n,(r+1):n)$ are an orthonormal basis of $N(A)$.

Next we show $AV = U\Sigma$.

$$\begin{aligned} AV &= A\,[V_1 \;\; V(1:n,(r+1):n)] \\ &= \begin{bmatrix} U_1\Sigma_r & 0_{m\times(n-r)} \end{bmatrix}. \\ U\Sigma &= [U_1 \;\; U(1:m,(r+1):m)] \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix} \\ &= \begin{bmatrix} U_1\Sigma_r & 0_{m\times(n-r)} \end{bmatrix}. \end{aligned}$$

■

Example 2 $A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$ *where $n = m = 2$ and rank$(A) = 1$.*

$$\begin{aligned} A^TA &= \begin{bmatrix} 5 & 5 \\ 5 & 5 \end{bmatrix} \text{ and } \det(\begin{bmatrix} 5-\lambda & 5 \\ 5 & 5-\lambda \end{bmatrix}) = 0 \text{ implies} \\ \lambda_1 &= 10,\ \sigma_1 = \sqrt{10} \text{ and } \lambda_2 = 0,\ \sigma_2 = 0. \end{aligned}$$

The first eigenvectors are

$$\begin{aligned} v_1 &= \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \text{ and} \\ u_1 &= \frac{Av_1}{\sigma_1} = \begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}. \end{aligned}$$

This gives the "small" SVD

$$u_1\sigma_1 v_1^T = \begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} \sqrt{10} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}.$$

The "full" SVD uses the v_2 basis for $N(A)$

$$v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \text{ and}$$

the u_2 basis for $N(A^T)$

$$u_2 = \begin{bmatrix} -2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}.$$

$$\begin{aligned} U\Sigma V^T &= \begin{bmatrix} u_1 & u_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix} \\ &= \begin{bmatrix} 1/\sqrt{5} & -2/\sqrt{5} \\ 2/\sqrt{5} & 1/\sqrt{5} \end{bmatrix} \begin{bmatrix} \sqrt{10} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}. \end{aligned}$$

Example 3 $A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 4 & 5 \end{bmatrix}$ *has rank$(A) = 2 = r$, $m = 4$ and $n = 3$. The "full" SVD has the form*

$$\begin{aligned} A &= U\Sigma V^T \\ &= \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \end{bmatrix} \\ &= \begin{bmatrix} u_1\sigma_1 & u_2\sigma_2 & 0_{4\times 1} \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ v_3^T \end{bmatrix} \\ &= u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T. \end{aligned}$$

The MATLAB command svd(A) can be used to do these calculations, and this is illustrated in the next section.

5.3 Matlab Code svd_ex.m

Line 9 inputs the matrix in Example 3. The MATLAB command [U S V] = svd(A) in line 14 computes the three factors. The eigenvectors of A^TA are computed in line 24 using [VV Lam_v] = eig(A'*A). Lines 14 and 26-31 confirm the properties of the SVD.

```
% SVD of a 4x3 Example.
% The rank of this matrix is r = 2.
% Then the dim(N(A)) = 1 and dim(N(A^T)) = 2.
%
clear
%
% Input data
%
A = [ 1 1 2; 1 2 3; 1 3 4; 1 4 5]
%
% Compute the SVD of A using Matlab.
%
display('Find the svd factors via the Matlab command svd()')
[U S V] = svd(A)
%
% Check the properties of SVD.
%
display('Check A - U*S*V^T is a zero matrix.')
A - U*S*V'
display('Compute the eigenvectors and
                           eigenvalues of A^T A and A A^T.')
display('Lam_u = Lam_v.')
display('NOte, s(1,1) = sqrt(Lam_u(4,4) and
                           s(2,2) = sqrt(Lam_u(3,3) and')
display('the change in order.')
[VV Lam_v] = eig(A'*A)
[UU Lam_u] = eig(A*A')
display('The last column of V is the
                           orthonormal basis of N(A).')
display('A V = ')
A*V
display('The last two colums of U form the
                           orthonormal basis N(A^T).')
display('A^T U = ')
A'*U
```

```
>> svd_ex

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

Find the svd factors via the Matlab command svd()
```

```
   U =
     -0.2524   -0.7977    0.0985    0.5388
     -0.3990   -0.3752    0.2703   -0.7918
     -0.5457    0.0473   -0.8360   -0.0328
     -0.6923    0.4698    0.4672    0.2858

   S =
     9.3441         0         0
          0    0.8290         0
          0         0    0.0000
          0         0         0

   V =
     -0.2022   -0.7911    0.5774
     -0.5840    0.5706    0.5774
     -0.7862   -0.2204   -0.5774

Check A - U*S*V^T is a zero matrix.

     =
     1.0e-14 *
           0         0   -0.0888
           0   -0.0444   -0.0888
      0.0111   -0.0888   -0.0888
     -0.0222   -0.1776   -0.0888

Compute the eigenvectors and eigenvalues of A^T A and A A^T.
Lam_u = Lam_v.
NOte, s(1,1) = sqrt(Lam_u(4,4) and s(2,2) = sqrt(Lam_u(3,3) and
the change in order.

   VV =
     0.5774    0.7911    0.2022
     0.5774   -0.5706    0.5840
    -0.5774    0.2204    0.7862

   Lam_v =
     -0.0000         0         0
           0    0.6872         0
           0         0   87.3128

   UU =
     -0.4877    0.2494   -0.7977    0.2524
      0.8361    0.0310   -0.3752    0.3990
    -0.2092   -0.8101    0.0473    0.5457
    -0.1392    0.5297    0.4698    0.6923

  Lam_u =
    -0.0000         0         0         0
          0    0.0000         0         0
          0         0    0.6872         0
          0         0         0   87.3128

The last column of V is the orthonormal basis of N(A).
  A V =
    -2.3585   -0.6612    0.0000
    -3.7287   -0.3110    0.0000
    -5.0989    0.0392    0.0000
    -6.4690    0.3894    0.0000

The last two colums of U form the orthonormal basis N(A^T).
  A^T U =
    -1.8894   -0.6558         0   -0.0000
    -5.4568    0.4730         0   -0.0000
    -7.3462   -0.1827         0   -0.0000
```

5.4 "Truncated" SVD

The "truncated" SVD is formed from the "small" SVD by dropping the latter columns of $U_1 = U(:, 1:r)$ and $V_1 = V(:, 1:r)$

$$A \cong A^{(k)} \equiv U(:, 1:k)\Sigma_k V(:, 1:k)^T \text{ with } k < r = \text{rank}(A).$$

Σ_k is the $k \times k$ diagonal matrix with components $\sigma_1 \geq \cdots \geq \sigma_k > 0$. An alternate way of writing this is

$$\begin{aligned} A^{(k)} &= \sum_{j=1}^{k} u_j \sigma_j v_j^T \text{ where} \\ U(\ :\ &, 1:k) = \begin{bmatrix} u_1 & \cdots & u_k \end{bmatrix} \text{ and} \\ V(\ :\ &, 1:k) = \begin{bmatrix} v_1 & \cdots & v_k \end{bmatrix}. \end{aligned}$$

An important observation is the first k columns only require the first k eigenvectors of $A^T A$.

Both A and $A^{(k)}$ are $m \times n$ matrices. In order the quantify the error in using the "truncated" SVD, introduce the 2-norm of any linear operator $A : \mathbb{R}^n \to \mathbb{R}^m$ represented by an $m \times n$ matrix. The error will then be measured by $\left\| A - A^{(k)} \right\|_2^2$ where $\|*\|_2$ is the 2-norm.

Definition 2 $\|A\|_2^2 \equiv \max\limits_{x\neq 0_{n\times 1}} \frac{(Ax)^T(Ax)}{x^Tx}$
$= \max\limits_{\widehat{x}^T\widehat{x}=1} (A\widehat{x})^T(A\widehat{x})$ *where* $\widehat{x} \equiv \frac{x}{(x^Tx)^{1/2}}$.

The 2-norm of A gives an upper bound on Ax

$$\|Ax\|_2 \leq \|A\|_2 \|x\|_2 .$$

Theorem 5.4.1 *Let A be an $m \times n$ with rank$(A) = r$,*
$A = \sum\limits_{j=1}^{r} u_j\sigma_j v_j^T$ be the "small" SVD and
$A^{(k)} = \sum\limits_{j=1}^{k} u_j\sigma_j v_j^T$ be the "truncated" SVD where $k \lessdot r$.
Then $\|A\|_2^2 = \sigma_1^2$ and $\left\|A - A^{(k)}\right\|_2^2 = \sigma_{k+1}^2$.

Proof. Let $V = \{v_1, \cdots, v_n\}$ be the orthonormal basis of eigenvectors for A^TA. Order the eigenvectors so that $(A^TA)v_j = \lambda_j v_j = \sigma_j^2 v_j$.

First, we show $\|A\|_2^2 \geq \sigma_1^2$:

$$\begin{aligned}
\text{Choose } x &= v_1. \\
\|A\|_2^2 &= \max_{x^Tx=1} (Ax)^T(Ax) \\
&\geq v_1^T A^T A v_1 \\
&= v_1^T \lambda_1 v_1 \\
&= 1\lambda_1 = \sigma_1^2.
\end{aligned}$$

Second, show $\|A\|_2^2 \leq \sigma_1^2$:

$$\begin{aligned}
\text{Let } x &= \sum_{j=1}^{n} c_j v_j \text{ with } x^Tx = 1. \text{ Then} \\
1 &= x^Tx = (\sum_{i=1}^{n} c_i v_i)^T(\sum_{j=1}^{n} c_j v_j) = \sum_{j=1}^{n} c_j^2 \text{ and} \\
A^TAx &= \sum_{j=1}^{n} c_j \lambda_j v_j. \\
x^TA^TAx &= (\sum_{i=1}^{n} c_i v_i)^T(\sum_{j=1}^{n} c_j \lambda_j v_j) \\
&= \sum_{j=1}^{n} c_j^2 \lambda_j 1 \text{ and use } \lambda_j \leq \lambda_1 \\
&\leq (\sum_{j=1}^{n} c_j^2)\lambda_1 \\
&= 1\lambda_1 = \sigma_1^2. \\
\|A\|_2^2 &= \max_{x^Tx=1} (Ax)^T(Ax) \leq \max_{x^Tx=1} \sigma_1^2 = \sigma_1^2.
\end{aligned}$$

Consider $A - A^{(k)}$ where $n < r$, and note

$$\begin{aligned} A - A^{(k)} &= \sum_{j=1}^{r} u_j \sigma_j v_j^T - \sum_{j=1}^{k} u_j \sigma_j v_j^T \\ &= \sum_{j=k+1}^{r} u_j \sigma_j v_j^T . \end{aligned}$$

This is the "small" SVD of $A - A^{(k)}$ and therefore $\left\| A - A^{(k)} \right\|_2^2 = \sigma_{k+1}^2$. ■

Example 4 *Consider Example 3 and the MATLAB code svd_ ex.m. Here A is 4×3, rank$(A) = 2$ and we let the truncation be $k = 1$. The following calculations verify Theorem 5.4.1:*

```
>> norm(A) =
    9.3441
>> S(1,1)
    9.3441

>> A1 = U(:,1)*S(1,1)*V(:,1)'
    0.4769    1.3773    1.8543
    0.7540    2.1775    2.9314
    1.0310    2.9776    4.0086
    1.3081    3.7778    5.0858

>> norm(A - A1) =
    0.8290
>> S(2,2)
    0.8290
```

Chapter 6

Three Applications of SVD

The last theorem of the previous chapter suggests the "truncated" SVD can be used to approximate the original matrix. Three important applications will be illustrated for image compression, search engines and noise filters. Additional applications will be given to the general least squares problem in Chapter 7, to hazard identification in Chapter 8, and to epedimic models in Chapter 9.

6.1 Image Compression

Grayscale images are associated with $m \times n$ matrices whose components are integers. For 8-bit images the integers range from 0 to $255 = 2^8 - 1$, and for 16-bit images they range from 0 to $65535 = 2^{16} - 1$. The black image pixel is associated with 0, and the white image pixel is associated with 255 (8-bit) or 65535 (16-bit). In MATLAB one can "view" the image in several ways. First, just inspect the matrix components. Second, use the MATLAB command mesh() to generate a surface of the image where the indices of the matrix are on the xy-plane and the intensity of the image is on the z-axis. Third, one can map the matrix into a standard image file such as a *.jpg file. This can be done by the MATLAB command imwrite(). The inverse of the imwrite() is imread(), which generates a 8-bit integer matrix from an image file.

Additional material on image compression and image processing can be found in [3] and in the last two chapters in [14]

Example 1 *We will create alphabetical letters from* 50×40 *matrices. The letter "U" can be created by defining a* 50×40 *matrix to initially be zero and then nonzero for some components to form the letter. This will produce a letter with black background and with lighter regions to form the letter. The following MATLAB function defines the letter "U" where the lighter region has an input value equal to* g.

```
function letu = letteru(g)
  letu = zeros(50,40);
  letu(10:40,4:8) = g;
  letu(10:40,32:36) = g;
  letu(10:14,4:36) = g;
```

The general scheme is to convert the image to a matrix, modify the matrix and then to convert the modified matrix back to a new image. The operation of light/dark corresponds to multiplying the matrix by a constant less/larger than one. The operation of cropping/panorama corresponds to deleting/augmenting rows or columns of the matrix. In image compression we use the "truncated" SVD of the image matrix. A "truncated" SVD of and $m \times n$ image matrix requires $km + kn + k$ storage of the k eigenvectors and singular values. If $k << m, n$, then this is a lot less than mn for the full image matrix.

Example 2 *In this example three letters are created by functions similar to the letter function letteru(g) to form the matrix usa and then the image USA. The mesh plot of the matrix is given in Figure 6.1.1, the image USA with black background is in Figure 6.1.2, and the "negative" of the image USA with white background is in Figure 6.1.3. The "negative" is created by replacing the components in the matrix usa by* $255 - usa(i, j)$*, that is,* $negusa(i, j) = 255 - usa(i, j)$*. The following MATLAB code imagusa.m was used to generate these figures.*

```
usa = [letteru(6) letters(9) lettera(12)];
mesh(usa)
newusa = 20*usa;
newusa = newusa(50:-1:1,:);
negusa = 255*ones(size(newusa)) - newusa;
newusa1 = uint8(newusa);
imwrite(newusa1, 'usa.jpg');
negusa1 = uint8(negusa);
imwrite(negusa1, 'negusa.jpg');
```

6.2 Matlab Code svdimage.m

Line 16 converts the image to an image matrix. Lines 22, 23 and 27 create the scaled image in the upper left in Figure 6.2.1. The "full" SVD of the image matrix is computed in line 35, and the "truncated" SVD with $k = 20$ is computed in line 39. This is displayed in the upper right in Figure 6.2.1. The loop in lines 50-59 varies the $k = 1 : 2 : 40$ and displays the images in the lower left in Figure 6.2.1 (requires one to hit the space bar). The lines 64-68 illustrate how to use portions of the SVD to "sharpen" the resulting image matrix. The reader will find it interesting to experiment with this and compare the images in the lower right in Figure 6.2.1.

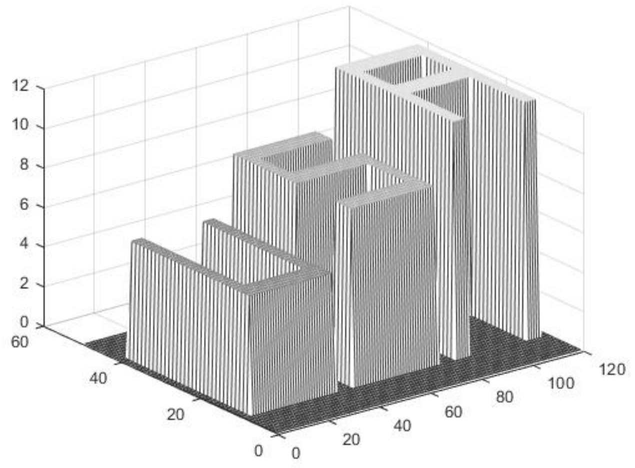


Figure 6.1.1: USA Matrix via mesh()

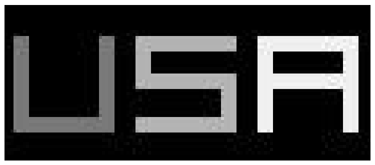


Figure 6.1.2: USA jpg Picture

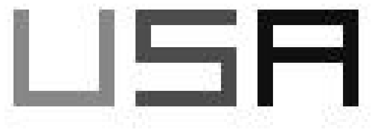


Figure 6.1.3: Negative of USA jpg Picture

```
% This code illustrates several images, their associated
% matrices,SVD expansions and image enhancements. There are
% four modified images, which are obtained by converting the
% image (*.jpg) to a matrix of 64-bit double precision
% numbers, modifying the matrix by using the SVD expansion,
% and then converting the new matrix into a new image (*.jpg).
%
clear; clf
%
% Input data
%
% Select an image. X will be an mxn matrix with 8-bit
% components rangingfrom 0 (dark) to 255 = 2^8 - 1 (light).
% load detail
% X = imread('moon.jpg');
X = imread('microchip.jpg');
% X = imread('pollen.jpg');
% The command double(X) converts the components to double
% precision (64-bit) numbers. The scale factor adjusts
% the darkness of the image.
%
scale = 0.3;
X = scale*double(X);
%
subplot(2,2,1)
% This is the new image.
image(X)
colormap(gray(64))
axis image, axis off
r = rank(X)
title(['rank = ' int2str(r)])
%
% Compute the SVD factors of X = u*s*v'.
%
[u s v] = svd(X,0);
%
subplot(2,2,2)
% This is the first 20 terms of the SVD expansion.
X20 = u(:,1:20)*s(1:20,1:20)*v(:,1:20)';
image(X20)
colormap(gray(64))
axis image, axis off
r20 = rank(X20)
title(['rank = ' int2str(r20)])
% 46     subplot(2,2,3)
```

```
%
% This loop presents a sequence of partial SVD expansions.
%
for rnew = 1:2:40
    Xnew = u(:,1:rnew)*s(1:rnew,1:rnew)*v(:,1:rnew)';
    image(Xnew)
    colormap(gray(64))
    axis image, axis off
    rnew
    title(['rank = ' int2str(rnew)])
    disp('Hit the space bar to move to the next image.')
    pause
end
%
subplot(2,2,4)
% This modifies the last image by adding additional parts
% of the SVD expansion.
% Xnew = X;
for rr = 39-2:39
    Xnew = Xnew + u(:,rr)*s(rr,rr)*v(:,rr)';
end
image(Xnew)
colormap(gray(64))
axis image, axis off
title(['sharpen image'])
```

6.3 Search Engines

Consider searching for item i in document j. Let m be the number of items and n be the number of documents.

$$\begin{aligned} A &= m \times n \text{ frequency matrix} \\ &= [a_{i,j}] \text{ where} \\ a_{i,j} &= \text{the number of times item } i \text{ appears in document } j. \end{aligned}$$

Example 3 *Suppose there are four items and two documents with*

	doc. 1	*doc. 2*
vector	*4*	*1*
matrix	*0*	*7*
real	*3*	*2*
complex	*1*	*4*

and define $A = \begin{bmatrix} 4 & 1 \\ 0 & 7 \\ 3 & 2 \\ 1 & 4 \end{bmatrix}$.

The frequency matrix can be sparse and very very large. In order to search for one or more items, we use a query vector

$$q = [q_i] \in \mathbb{R}^m \text{ where } q_i = 1 \text{ for a single item } i.$$

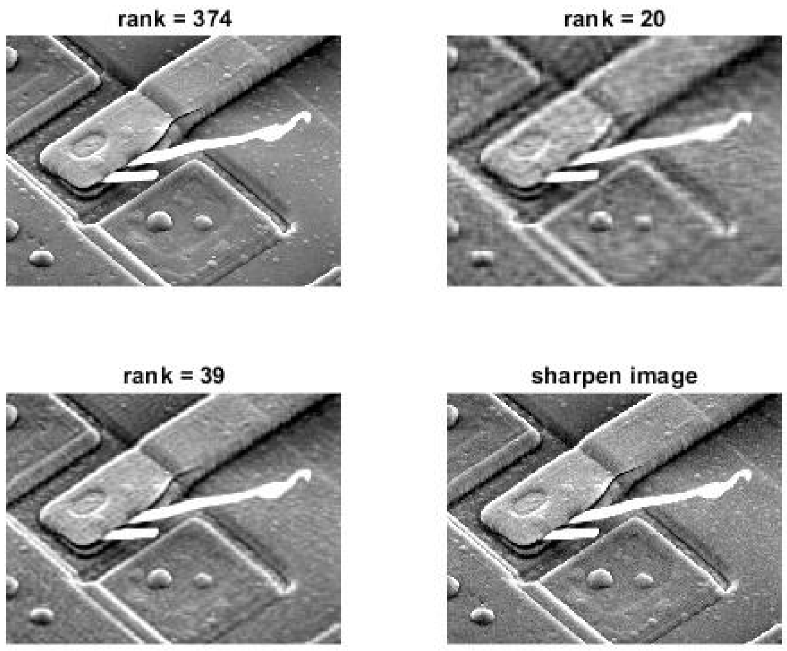


Figure 6.2.1: Image Compression Using SVD

Search for item i in document d_j (column j in A) by computing $q^T d_j$, and then normalize it by using the Cauchy inequality

$$-1 \le \frac{q^T d_j}{\|q\|_2 \, \|d_j\|_2} \le 1.$$

Definition 1 *Let $d_j = Ae_j$ be column j in the $m \times n$ frequency matrix A.*

$$\cos(\theta_j) \equiv \frac{q^T A e_j}{\|q\|_2 \, \|Ae_j\|_2}.$$

Example 4 *Return to Example 3 and consider four query vectors.*
$q = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$ *gives* $\cos(\theta_1) = \frac{4}{\sqrt{26}}$ *and* $\cos(\theta_2) = \frac{1}{\sqrt{70}}$,
$q = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$ *gives* $\cos(\theta_1) = \frac{0}{\sqrt{26}}$ *and* $\cos(\theta_2) = \frac{7}{\sqrt{70}}$,
$q = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T$ *gives* $\cos(\theta_1) = \frac{3}{\sqrt{26}}$ *and* $\cos(\theta_2) = \frac{2}{\sqrt{70}}$ *and*
$q = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}^T$ *gives* $\cos(\theta_1) = \frac{5}{\sqrt{2}\sqrt{26}}$ *and* $\cos(\theta_2) = \frac{5}{\sqrt{2}\sqrt{70}}$.

Because the components of A and q are nonnegative, $0 \leq \cos(\theta_j) \leq 1$. If $\cos(\theta_j)$ is close to 1, then one can make a judgement that document j has the items in the q. Because the frequency matrix is very very large, we will approximate it by the "truncated" SVD

$$A \approx A^{(k)} = U^{(k)}\Sigma_k(V^{(k)})^T \text{ with } k < \text{rank}(A) = r.$$

This gives a new $\cos(\theta_j)$

$$\frac{q^T U^{(k)}(\Sigma_k(V^{(k)})^T e_j)}{\|q\|_2 \left\|U^{(k)}(\Sigma_k(V^{(k)})^T e_j)\right\|_2}.$$

Use the orthonormal property and define $S_j \equiv \Sigma_k(V^{(k)})^T e_j$.

Definition 2 *First search engine approximation is*

$$\cos(\theta 1_j) \equiv \frac{q^T U^{(k)} S_j}{\|q\|_2 \|S_j\|_2}.$$

This approximation may have negative values, and the choice of k is a judgement to be made. A variation is to project the query q to the range of $A^{(k)}$. In order to find this projection, use the orthonormal basis of $R(A^{(k)})$ given by the columns of $U^{(k)}$. Theorem 1.4.1 yields

$$P_{R(A^{(k)})}(q) = U^{(k)}c \text{ where } (U^{(k)})^T q = c.$$

In the first search engine replace q by $P_{R(A^{(k)})}(q) = U^{(k)}c = U^{(k)}(U^{(k)})^T q$. Use the orthonormal property and Theorem 1.4.1 to conclude

$$\begin{aligned} P_{R(A^{(k)})}(q)^T P_{R(A^{(k)})}(q) &= ((U^{(k)})^T q)^T((U^{(k)})^T q) \leq q^T q \text{ and} \\ P_{R(A^{(k)})}(q)^T A^{(k)} e_j &= q^T U^{(k)} S_j. \end{aligned}$$

This gives the second version of search engine.

Definition 3 *Second search engine approximation is*

$$\cos(\theta 2_j) \equiv \frac{q^T U^{(k)} S_j}{\left\|(U^{(k)})^T q\right\|_2 \|S_j\|_2} \geq \cos(\theta 1_j).$$

Both search engines will be illustrated in the next section. The first code is a simple example with small dimensions. The second code uses a $11,390 \times 1,265$ frequency matrix, and there can be, for example, a significant difference in $\left\|(U^{(k)})^T q\right\|_2 = 71.5542$ and $\left\|q^T q\right\|_2 = 2.3287$.

The search engine literature is extensive, but the reader may find the following to be useful: [2], [1] and [10].

6.4 Matlab Codes sengine.m, senginesparse.m

The first code, sengine.m, illustrates the effect of using variable, k, truncation of the SVD. The sample calculations are for smaller frequency matrices. One case is for a 5×6 matirx whose rank is 5. It is interesting to experiment with different query vectors, q.

Line 13 defines (it is not used) the frequency matrix for Example 1. In lines 14-18 the larger frequency matrix is defined and uses in the subsequent calculations. A query vector is defined in line 19, and the user of the code may want to experiment with this. The SVD is calculated in line 24. The outer loop in lines 29-43 varies the document number, j. The inner loop varies the truncation number, k. The two search engine values for $cos(\theta 1)$ and $cos(\theta 2)$ are computed inside these nested loops in lines 33 and 34.

```
1     %
2     % Search Engine
3     %
4     % The code uses the SVD to search an mxn frequency matrix
5     % where the the j-column contains the frequency of items in
6     % document j. This trival example illustrates two search
7     % methods and variable truncation of the SVD.
8     %
9     clear
10     %
11     %Input data
12     %
13     %  A = [ 4 1;0 7;3 2;1 4];            % frequency matrix
14     A = [ 4 1 0 0 1 0;
15         0 7 0 1 2 0;
16         3 2 0 0 0 1;
17         1 4 1 0 3 2;
18         0 1 0 0 1 0];
19     q = [ 1 0  0 0 1]';                  % query
20     r = rank(A);                         % rank of A
21     %
22     % Compute the SVD
23     %
24     [U S V] = svd(A);
25     [m n]   = size(A);
26     S
27     % For each document (j) compute uss all the truncated SVDs.
28     % Compute both the versions of the cos theta.
29     for j = 1:n
30        for k = 1:r
31           Sv = S(1:k,1:k)*V(:,1:k)';
32           SV = Sv(:,j);
```

```
33          costhjone(k) = q'*U(:,1:k)*SV/(norm(q)*norm(SV));
34          costhjtwo(k) = q'*U(:,1:k)*SV/...
                                    (norm(U(:,1:k)'*q)*norm(SV));
35       end
36    %
37    % Output all possible cos theta
38    %
39       display('document = ' )
40       j
41       display('variable truncation for cos theta one and two =')
42       [costhjone' costhjtwo']
43    end
```

```
>> sengine

S =
   Columns 1 through 6
     9.4508         0         0         0         0         0
          0    4.7367         0         0         0         0
          0         0    2.3673         0         0         0
          0         0         0    1.2675         0         0
          0         0         0         0    0.1883         0

   document j =
     1

   variable truncation for cos theta one and two =
     0.2609    1.0000
     0.5461    0.9998
     0.5459    0.9989
     0.5556    0.6364
     0.5547    0.5547
```

The second code, senginesparse.m, uses a larger frequency matrix. It is $11,390 \times 1,265$ and has $109,056$ nonzero components. The rank of the frequency matrix is 988, and the SVD is truncated after $k = 20$. The graphs of the two search engines, $\cos(\theta 1)$ and $\cos(\theta 2)$, are given in Figure 6.4.2. For the particular query in the code, four documents are identified has having the query.

The frequency matrix is imported in lines 14-17. Truncated SVD with $k = 20$ is computed in line 24, and the query is defined in line 27. The singular values are graphed by the command in line 25, see Figure 6.4.1 The loop in lines 35-39 is over all documents, and both $\cos(\theta 1)$ and $\cos(\theta 2)$ are computed for each document, j. The output is given graphical form in Figure 6.4.2.

```
%
% Search Engine Sparse
%
% The code uses the SVD to search an mxn frequency matrix
% where the the j-column contains the frequency of items in
% document j. This sparse example two search methods
% and k = 20 truncation of the SVD.
%
clear
%
% Input data
%
% download webmatrix_uri            % frequency matrix
svdmatrix = uiimport('webmatrix_uri');
SS = svdmatrix.data;
SS(1,:)
A = sparse(SS(2:109057,1), SS(2:109057,2), SS(2:109057,3));
[m n] = size(A);
display('number of search items = ')
m
display('number of documents = ')
n
r = rank(full(A))                   % rank of full A
sig = svds(A,r);
plot(sig); title('all singular values')
% choose query q
q = zeros(m,1); q(2)= 4; q(55) = 2; q(60:110) = 10;
%
% For truncated SVD with k = 20 compute the cos theta for
% all documents.
%
for k = 20                          % truncate svd after k
   [U sig V] = svds(A,k);
   Sv = sig(1:k,1:k)*V(:,1:k)';
   for j = 1:n;
      SV = Sv(:,j);
      costhjone(j) = q'*U(:,1:k)*SV/(norm(q)*norm(SV));
      costhjtwo(j) = q'*U(:,1:k)*SV/...
                          (norm(U(:,1:k)'*q)*norm(SV));
   end
end
[costhjone' costhjtwo'];
%
% Graph the cos theta verse the documents
%
```

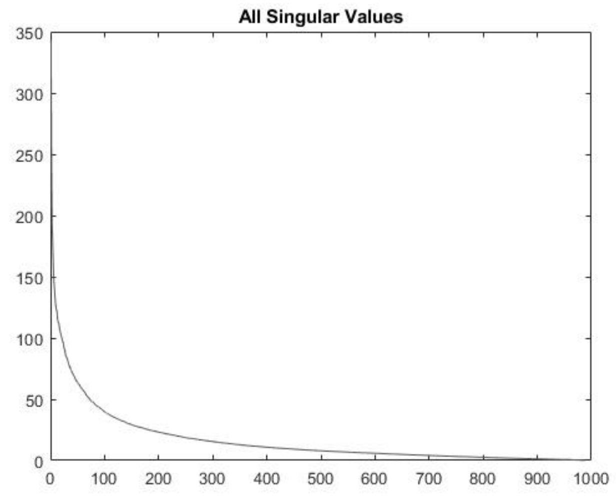


Figure 6.4.1: All Singular Values

```
figure(2)
subplot(2,1,1) , plot(1:n, costhjone')
axis([ 0 1400 -0.1 0.1])
title('cos theta one')
subplot(2,1,2) , plot(1:n, costhjtwo')
axis([ 0 1400 -0.3 1.0])
title('cos theta two')
```

```
>> senginesparse

   ans =
       11390        1265       109056

   m =
     11390

   n =
     1265

   r =
     988
```

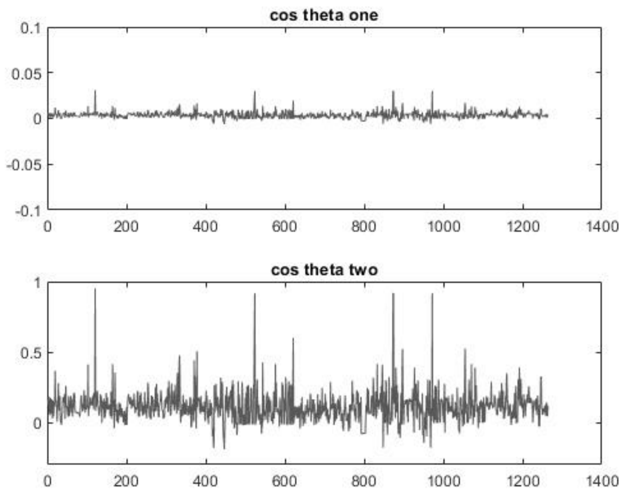


Figure 6.4.2: Search Using Truncated SVD

6.5 Noise Filter

Consider a discrete signal $f \in \mathbb{R}^n$. There is some blurring (K an $n \times n$ matrix) and some signal noise ($\eta \in \mathbb{R}^n$). The data received ($d \in \mathbb{R}^n$) is given by

$$d = Kf + \eta.$$

The objective is to approximate f from the data d. There are two problems: the noise is not known, and the matrix inversion tends to be sensitive to variations in the data.

The matrix K has exponential components

$$K = \left[hCe^{-((i-j)h)^2/2\gamma^2}\right] \text{ with } h = 1/n.$$

It is nonsingular, but it has very small singular values, for example, K in the next section has $\sigma_n = 4.2275\ 10^{-7}$. The SVD of $K = U\Sigma_n V^T$ has three $n \times n$ matrices and $\sigma_i > 0$. Use the orthonormal properties to get

$$\begin{aligned} K^{-1} &= V\Sigma_n^{-1}U^T \text{ and} \\ K^{-1}d &= f + K^{-1}\eta \\ V(\Sigma_n^{-1}U^T d) &= f + V(\Sigma_n^{-1}U^T\eta). \end{aligned}$$

Observe the latter column vectors of V have high frequencies (see the next section). Since the noise vector η has higher frequency components, one can drop some of the latter terms in the SVD, that is, use the truncated SVD $K^{(k)} = U^{(k)}\Sigma_k(V^{(k)})^T$ with $k < \text{rank}(K) = n$. The computed signal is

$$\begin{aligned}\widehat{f} &= V^{(k)}(\Sigma_k^{-1}(U^{(k)})^T d) \\ &= \sum_{j=1}^{k} v_j \frac{1}{\sigma_j} u_j^T d \\ &= \sum_{j=1}^{k} (\frac{1}{\sigma_j} u_j^T d) v_j.\end{aligned}$$

Depending in the choice the truncation, the σ_j may still be small so that variations in the data can cause large errors. This is more carefully discussed in Chapter 8 on ill-conditioned matrices. One possible remedy is to replace

$$\frac{1}{\sigma_j} \text{ by } \frac{1}{\sigma_j}\frac{\sigma_j^2}{\sigma_j^2 + \alpha} \text{ where } \alpha > 0.$$

When α is near 0.0, then this approximates $1/\sigma_j$. When α gets large, then this approximates 0.0.

Definition 4 *Let K be $n \times n$ and choose $\alpha > 0$ and $k < n$. The Tikhonov-Phillips regularization is*

$$(K^T K + \alpha I_n)^{-1} K^T d.$$

The truncated Tikhonov-Phillips regularization is

$$\begin{aligned}\widetilde{f} &= ((K^{(k)})^T K^{(k)} + \alpha I_k)^{-1}(K^{(k)})^T d \\ &= \sum_{j=1}^{k} (\frac{\sigma_j}{\sigma_j^2 + \alpha} u_j^T d) v_j.\end{aligned}$$

Matrix and summation versions of the truncated Tikhonov-Phillips regularization are the same, and this can be established by using the orthonormal properties in the SVD. The choice of $\alpha > 0$ and $k < n$ can be judgemental, and this is illustrated in the next section.

For additional information and analysis the reader should consult [13].

6.6 Matlab Code image1dsvd.m

The matrix K is 100×100 and is defined by Setup1dsvd.m. The following MATLAB code Kmatrix.m uses this and computes the SVD of K. The singular values vary from largest to very very small

$$\sigma_1 = 0.9985, \sigma_{40} = 0.0944 \text{ and } \sigma_{100} = 4.2267\ 10^{-7}.$$

The eigenvectors u_j and v_j have more oscillations as j increases. This is illustrated in Figure 6.6.1.

```
1     Setup1dsvd;
2     [U S V] = svd(K);
3     S = diag(S);
4     S(1)
5     S(40)
6     S(100)
7     figure(3)
8     subplot(1,2,1)
9     plot(V(:,2));
10     hold on
11     plot(V(:,10));
12     subplot(1,2,2)
13     mesh(U(:,10)*S(10)*V(:,10)');
```

The matrix K and the signal with noise is defined in line 8 by Setup1dsvd.m. This is illustrated in the upper left graph in Figure 6.6.2. The full SVD of the matrix is computed in line 9. The truncation $k = 40$ is used in line 11, and the user may want to experiment with the truncation number. Three values for α are used as indicated by the loop starting at line 16. The truncated Tikhonov-Phillips approximations are computed in the inner loop in lines 18-21. The graphs of the approximated signal are given by lines 25-26 and are displayed in the upper right, lower left and lower right graphs in Figure 6.6.2.

```
1    % This code illustrates noise filters using the SVD.
2    % The 1D data is given by Setup1dsvd.
3    %
4    clear;
5    %
6    % Input data from Setup1dsvd.m
7    %
8    Setup1dsvd
9    [U S V] = svd(K);
10    S = diag(S);
11    k = 40                  % truncation of SVD
12    %
13    % Compute SVD filter using variable parameter alpha.
14    %
15    num = 2;
16    for alpha = [0.001 0.034 0.900]
17       newimage = zeros(n,1);
18       for j = 1:k
19          newimage = newimage + ...
20             (S(j)*(U(:,j)'*d(:))/(S(j)*S(j) + alpha))*V(:,j);
21       end
22    %
23    % Graphaical output is given for each alpha.
```

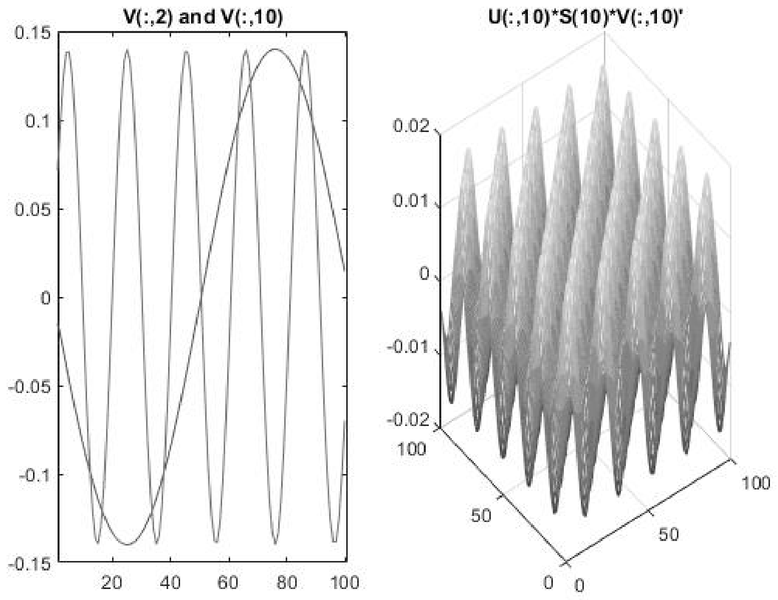


Figure 6.6.1: Vectors From SVD of K

```
%
   subplot(2,2,num)
   plot(x,f_true,'-',x,newimage)
   axis([0 1 -.2 1.6])
   title([ 'alpha = ' num2str(alpha)])
   num = num + 1;
end
```

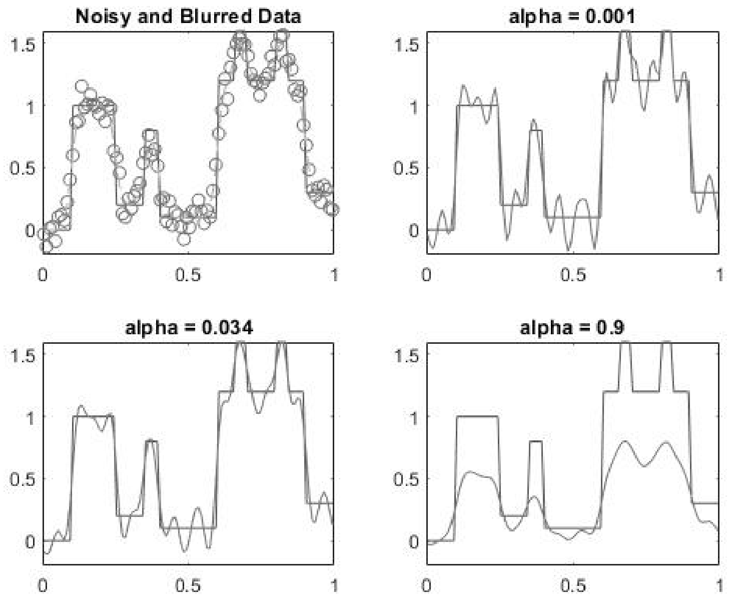


Figure 6.6.2: Noise Filter with Truncation k = 40

Chapter 7

Pseudoinverse of A

The pseudoinverse of an $m \times n$ matrix is an $n \times m$ matrix $A^\dagger$. It is important because $A^\dagger d$ is a minimal 2-norm solution to the general least squares problem. Moreover, when $A^\dagger$ is restricted to $R(A)$, it is a right inverse, that is, $A(A^\dagger d) = d$ when $d = A\widetilde{x}$.

7.1 $\Sigma^\dagger$and $A^\dagger = V\Sigma^\dagger U^T$

As motivation for the definition of the pseudoinverse of an $m \times n$ matrix with rank$(A) = r$, consider the least squares problem. Let $A = U_1\Sigma_r V_1^T$ be the "small" SVD where the columns of U_1 and V_1 are orthonormal bases for $R(A)$ and $R(A^T)$, respectively. Theorem 3.2.1 gives the minimal 2-norm least squares solution based on two projections

$$A\widehat{x} = P_{R(A)}(d) \text{ and } x_R = P_{R(A^T)}(\widehat{x}).$$

Because the bases are orthonormal, the projections are easily computed

$$P_{R(A)}(d) = U_1(U_1^T d) \text{ and } P_{R(A^T)}(\widehat{x}) = V_1(V_1^T \widehat{x}).$$

Combine these equations to get

$$\begin{aligned} A\widehat{x} &= P_{R(A)}(d) \\ U_1\Sigma_r V_1^T \widehat{x} &= U_1(U_1^T d) \text{ and} \\ V_1^T\widehat{x} &= \Sigma_r^{-1}U_1^T d. \end{aligned}$$

The minimal 2-norm solution is

$$x_R = P_{R(A^T)}(\widehat{x}) = V_1(V_1^T\widehat{x}) = (V_1\Sigma_r^{-1}U_1^T)d.$$

Definition 1 *Let $A = U\Sigma V^T$ be the SVD of an $m \times n$ matrix with rank(A) $= r$. The pseudoinverse is an $n \times m$ matrix*

$$\begin{aligned} A^\dagger &\equiv V\Sigma^\dagger U^T \text{ with} \\ \Sigma^\dagger &\equiv \begin{bmatrix} \Sigma_r^{-1} & 0_{r\times(m-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(m-r)} \end{bmatrix} \text{ is an } n \times m \text{ matrix.} \end{aligned}$$

The proofs of the following identities are routine, but they do lead to the fundamental properties of the general pseudoinverse $A^\dagger$.

Theorem 7.1.1 *Identities with Σ and $\Sigma^\dagger$.*

1. $\Sigma^T\Sigma\Sigma^\dagger = \Sigma^T$.
2. $\Sigma^\dagger\Sigma\Sigma^T = \Sigma^T$.
3. $\Sigma\Sigma^\dagger\Sigma = \Sigma$.
4. *If A has full column rank, rank$(A) = r = n$ with $m > n$, then* $\Sigma = \begin{bmatrix} \Sigma_r \\ 0_{(m-r)\times r} \end{bmatrix}$, $\Sigma^\dagger\Sigma = I_n$ *and* $\Sigma^T\Sigma = \Sigma_n^2$.
5. *If A has full row rank, rank$(A) = r = m$ with $m < n$, then* $\Sigma = \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \end{bmatrix}$, $\Sigma\Sigma^\dagger = I_m$ *and* $\Sigma\Sigma^T = \Sigma_m^2$.
6. $(\Sigma^\dagger)^T\Sigma^\dagger = \begin{bmatrix} \Sigma_r^{-2} & 0_{r\times(m-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(m-r)} \end{bmatrix}$ *is $m \times m$.*
7. $\Sigma^T\Sigma = \begin{bmatrix} \Sigma_r^2 & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix}$ *is $n \times n$.*

Proof. Show the third identity is true.

$$\begin{aligned} \Sigma^\dagger\Sigma &= \begin{bmatrix} \Sigma_r^{-1} & 0_{r\times(m-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(m-r)} \end{bmatrix} \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix} \\ &= \begin{bmatrix} I_r & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix} \text{ is } n \times n. \\ \Sigma(\Sigma^\dagger\Sigma)^T &= \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix} \begin{bmatrix} I_r & 0_{r\times(n-r)} \\ 0_{(n-r)\times r} & 0_{(n-r)\times(n-r)} \end{bmatrix} \\ &= \begin{bmatrix} \Sigma_r & 0_{r\times(n-r)} \\ 0_{(m-r)\times r} & 0_{(m-r)\times(n-r)} \end{bmatrix} \\ &= \Sigma. \end{aligned}$$

■

Use the above identities and the orthonormal properties to establish the following theorem.

Theorem 7.1.2 *Let A be $m \times n$ matrix with rank$(A) = r$. Then*

1. $(A^TA)A^\dagger = A^T$.
2. $A^\dagger(AA^T) = A^T$.
3. $AA^\dagger A = A$.
4. $A(A^\dagger d) = d$ *when $d \in R(A)$.*

5. *If A has full column rank, then A^TA is nonsingular and*

$$A^\dagger = (A^TA)^{-1}A^T.$$

6. *If A has full row rank, then AA^T is nonsingular and*

$$A^\dagger = A^T(AA^T)^{-1}.$$

Proof. The first and second items follow from the first and second identities in Theorem 7.1.1 and the orthonormal properties. In order to show the third item, use $A = U\Sigma V^T$ and $A^\dagger = V\Sigma^\dagger U^T$.

$$\begin{aligned} AA^\dagger A &= (U\Sigma V^T)\left(V\Sigma^\dagger U^T\right)(U\Sigma V^T) \\ &= U\Sigma I_n \Sigma^\dagger I_m \Sigma V^T \\ &= U(\Sigma\Sigma^\dagger\Sigma)V^T \\ &= U\Sigma V^T \text{ by identity 3 in Theorem 7.1.1} \\ &= A. \end{aligned}$$

The proof of the fourth part follows from $AA^\dagger A = A$. $d \in R(A)$ means there is an $\widehat{x} \in \mathbb{R}^n$ such that $A\widehat{x} = d$.

$$\begin{aligned} A(A^\dagger d) &= A(A^\dagger(A\widehat{x})) \\ &= (AA^\dagger A)\widehat{x} \\ &= A\widehat{x} \\ &= d. \end{aligned}$$

■

Example 1 *Return to Examples 7,8 in Sections 1.3, 1.4 and Example 2 in Section 3.2.*

$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{bmatrix}$. *Compute $A = U\Sigma V^T$, $A^\dagger = V\Sigma^\dagger U^T$ and use $A^\dagger$ to solve the least squares problem $Ax = d = \begin{bmatrix} 10 & 5 & 2 \end{bmatrix}^T$.*

First, compute $A = U\Sigma V^T$ by finding the eigenvalues of

$$\begin{aligned} A^TA &= 14\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}. \\ \det(14\begin{bmatrix} 1-\lambda & 2 \\ 2 & 4-\lambda \end{bmatrix}) &= 0 \textit{ implies } \lambda_1 = 70 \textit{ and } \lambda_0 = 0. \end{aligned}$$

This gives $\sigma_1 = \sqrt{70}$, $\sigma_2 = 0$ and

$$v_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}/\sqrt{5} \textit{ and } u_1 = \frac{Av_1}{\sigma_1} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}/\sqrt{14}.$$

The "small" SVD is $A = u_1\sigma_1 v_1^T$. In order to find the "full" SVD, we need to extend the bases to $N(A)$ and $N(A^T)$. $Av_2 = 0_{3\times 1}$ and $A^T u_2 = 0_{2\times 1}$ gives

$$v_2 = \begin{bmatrix} -2 \\ 1 \end{bmatrix} /\sqrt{5} \text{ and } u_2 = \begin{bmatrix} -5 \\ 1 \\ 1 \end{bmatrix} /\sqrt{27}.$$

The "full" SVD is

$$A = \begin{bmatrix} u_1 & u_2 \end{bmatrix} \begin{bmatrix} \sqrt{70} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \end{bmatrix}.$$

Second, compute $A^\dagger$ and $A^\dagger d$

$$\begin{aligned} A^\dagger &= \begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} 1/\sqrt{70} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1^T \\ u_2^T \end{bmatrix} \text{ and} \\ A^\dagger d &= \begin{bmatrix} 1/\sqrt{5} & -2/\sqrt{5} \\ 2/\sqrt{5} & 1/\sqrt{5} \end{bmatrix} \begin{bmatrix} 1/\sqrt{70} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}/\sqrt{14} \\ \begin{bmatrix} -5 & 1 & 1 \end{bmatrix}/\sqrt{27} \end{bmatrix} \begin{bmatrix} 10 \\ 5 \\ 2 \end{bmatrix} \\ &= \begin{bmatrix} 1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix} ((1/\sqrt{70}) \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} /\sqrt{14} \begin{bmatrix} 10 \\ 5 \\ 2 \end{bmatrix}) \\ &= \begin{bmatrix} 13/35 \\ 26/35 \end{bmatrix}. \end{aligned}$$

Third, relate this to the general least squares solution

$$x = \begin{bmatrix} 26/14 \\ 0 \end{bmatrix} + c \begin{bmatrix} 2 \\ -1 \end{bmatrix}.$$

Let $f(c) = x^T x = (2c + 26/14)^2 + c^2$. This has a minimum at $c = -26/35$ so that

$$\begin{aligned} x &= \begin{bmatrix} 26/14 \\ 0 \end{bmatrix} + (-26/35) \begin{bmatrix} 2 \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} 13/35 \\ 26/35 \end{bmatrix} = A^\dagger d. \end{aligned}$$

7.2 $A^\dagger$ and Least Squares

Example 1 generalizes to any matrix. $A^\dagger d$ can be written in summation form

$$A^\dagger d = \sum_{j=1}^{r} v_j (\frac{1}{\sigma_j}) u_i^T d \text{ where rank}(A) = r.$$

Since $1 \le j \le r$, this means $A^\dagger d \in R(A^T)$.

Theorem 7.2.1 *Let A be $m \times n$ matrix with rank$(A) = r$.*

1. $x^+ = A^\dagger d$ is a solution of the normal equations.

2. Moreover, it is a minimal solution of the normal equations, that is,

$$(x^+)^T(x^+) \le \min_{x \in N(A)} (x^+ + x)^T(x^+ + x).$$

Proof. $A^\dagger d$ is a solution of the normal equations follows from $A^T A A^\dagger = A^T$ in the previous theorem.

$$A^T A(A^\dagger d) = (A^T A A^\dagger)d = A^T d.$$

The proof that $A^\dagger d$ is a minimal least squares solution uses $x^+ = A^\dagger d = V(\Sigma^\dagger U^T d) \in R(A^T)$. If $x \in N(A)$, then $x^+ + x$ is also a solution of the normal equations. By Theorem 3.2.1 $N(A) = R(A^T)^\perp$ and note

$$\begin{aligned}(x^+ + x)^T(x^+ + x) &= (x^+)^T x^+ + 2x^T x^+ + x^T x \\ &= (x^+)^T x^+ + 0 + x^T x \\ &\ge (x^+)^T x^+ \text{ for all } x \in N(A).\end{aligned}$$

■

Example 2 *Return to Example 3 in Sections 5.2 and 5.3*

$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 4 & 5 \end{bmatrix}$ *has rank$(A) = 2$. The first two columns of A are a basis for $R(A)$, and the vector $\begin{bmatrix} -1 & -1 & 1 \end{bmatrix}^T$ is a basis for $N(A)$. The general least squares solution of $Ax = d = \begin{bmatrix} 1 & 2 & 5 & 4 \end{bmatrix}^T$ is*

$$\begin{aligned} x &= \begin{bmatrix} 0 \\ 1.2 \\ 0 \end{bmatrix} + c \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \text{ where} \\ A\begin{bmatrix} 0 \\ 1.2 \\ 0 \end{bmatrix} &= P_{R(A)}(d) = \begin{bmatrix} 1.2 \\ 2.4 \\ 3.6 \\ 4.8 \end{bmatrix}. \end{aligned}$$

$P_{R(A)}(d) = c_1 w_1 + c_2 w_2$ where w_1 and w_2 are the first and second columns of A. The $c_1 = 0.0$ and $c_2 = 1.2$ are found by solving

$$\begin{aligned} w_1^T(d - c_1 w_1 - c_2 w_2) &= 0 \text{ and} \\ w_2^T(d - c_1 w_1 - c_2 w_2) &= 0. \end{aligned}$$

The smallest $x^T x$ is given by $c = 0.4$ and $x = \begin{bmatrix} -.4 & .8 & .4 \end{bmatrix}^T$.

This agrees with the following MATLAB computations.

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

   d = [ 1 2 5 4]'
       1
       2
       5
       4

   [U S V] = svd(A);

>> pinv(A)
     =
      0.7667    0.3667   -0.0333   -0.4333
     -0.5333   -0.2333    0.0667    0.3667
      0.2333    0.1333    0.0333   -0.0667

>> V*pinv(S)*U'
     =
      0.7667    0.3667   -0.0333   -0.4333
     -0.5333   -0.2333    0.0667    0.3667
      0.2333    0.1333    0.0333   -0.0667

>> pinv(A)*d
     =
     -0.4000
      0.8000
      0.4000
```

Example 3 *This example exposes some possible difficulties with the pseudoinverse. Let ϵ be a small positive number and consider the 3×2 matrix*

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

Since A has full column rank,

$$A^{\dagger} = (A^T A)^{-1} A^T = \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/\epsilon & 0 \end{bmatrix}.$$

First, note A converges as $\epsilon \to 0$, but $A^{\dagger}$ does not. Second, a small variation in *the right side of* $Ax = d$ *can give a large variation in the least squares solution*

$$A^\dagger d = \begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/\epsilon & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} (d_1 + d_3)/2 \\ d_2/\epsilon \end{bmatrix}.$$

If $\epsilon = 10^{-6}$ *and* d_2 *varies by* 1.0, *the second component varies by* 10^6.

Chapter 8

Ill-conditioned Least Squares

Ill-conditioned matrices have the unpleasant property that small perturbations in the $n \times n$ matrix or right side can give large variations to the solution to the unperturbed system. This will be generalized to certain $m \times n$ matrices and the least squares problem. An interesting application is to hazard identification such as a pollutant in a river with point sources along the river. Can one find the location and intensities of the point sources from observations at other locations along the river? In this application we restrict the location and find the intensities. The location of the observations can generate ill-conditioned least squares problems.

8.1 Condition Number

Consider a nonsingular $n \times n$ matrix and use any norm. The classical analysis with $Ax = d$ and $A(x + \Delta x) = d + \Delta d$ estimates Δx relative to x.

$$\begin{aligned} A\Delta x &= \Delta d \text{ and } \Delta x = A^{-1}\Delta d. \\ \|d\| &\leq \|A\| \, \|x\| \text{ or } \frac{1}{\|x\|} \leq \|A\| \frac{1}{\|d\|}. \\ \|\Delta x\| &\leq \left\|A^{-1}\right\| \|\Delta d\| . \end{aligned}$$

Combine these to get

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A\| \left\|A^{-1}\right\| \frac{\|\Delta d\|}{\|d\|}.$$

The classical condition number is $\|A\| \left\|A^{-1}\right\|$, but it is dependent on the choice of the norm. If the 2-norm is used, then by Theorem 5.4.1 $\|A\| \left\|A^{-1}\right\| = \sigma_1/\sigma_r$. For general $m \times n$ use this ratio as condition number.

Definition 1 *Let A be $m \times n$ matrix with rank$(A) = r$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$.*

$$\mathit{cond}(A) \equiv \sigma_1/\sigma_r.$$

Example 1 $A = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$ *or, more generally,* $A = H(n) = \left[\frac{1}{1+i+j}\right]$ *where $i, j = 0, ..., n-1$. The condition numbers increases rapidly with n:*

$$\begin{aligned} \mathit{cond}(H(3)) &= 524.0568, \\ \mathit{cond}(H(4)) &= 1.5514\ 10^4 \text{ and} \\ \mathit{cond}(H(5)) &= 4.7661\ 10^5. \end{aligned}$$

If the right sides of $Ax = d$ are close, then the solutions may not be so close. For example,

$$\begin{aligned} H(4)\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} &= \begin{bmatrix} 2.0837 \\ 1.2833 \\ 0.9500 \\ 0.7595 \end{bmatrix} \text{ and} \\ H(4)\begin{bmatrix} -8 \\ 114 \\ -288 \\ 196 \end{bmatrix} &= \begin{bmatrix} 2.0000 \\ 1.2000 \\ 0.9000 \\ 0.8000 \end{bmatrix}. \end{aligned}$$

This is the Hilbert matrix, and it evolves from approximation of general function by a polynomial. Choose the coefficients of the polynomial to minimize the mean square integral of the difference.

$$\begin{aligned} F(c) &\equiv \int_0^1 (f(x) - \sum_{j=0}^{n-1} c_i x^j)^2 dx \\ 0 &= \frac{\partial F}{\partial c_i} = 2\int_0^1 (f(x) - \sum_{j=0}^{n-1} c_j x^j)^{2-1} x^i dx \\ \int_0^1 f(x) x^i dx &= \sum_{j=0}^{n-1} c_j \int_0^1 x^j x^i dx = \sum_{j=0}^{n-1} c_j \frac{1}{1+j+i}. \end{aligned}$$

Example 2 *Let $A = K$ be the blurring matrix in the noise filter application in Sections 6.5 and 6.6. In the calculations in Section 6.6 the K matrix is* 100×100 *with* $\sigma_1 = 0.9985$, $\sigma_{100} = 4.2267\ 10^{-7}$ *and cond*$(K) = 2.3620\ 10^6$.

Theorem 8.1.1 *Let A be an $m \times n$ matrix with rank$(A) = r$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. $A^\dagger \equiv V\Sigma^\dagger U^T$ is almost in the SVD form and*

$$\|A\|_2 = \sigma_1 \text{ and } \left\|A^\dagger\right\|_2 = 1/\sigma_r.$$

Proof. The eigenvectors in $A^\dagger$ need to be reordered so that the singular values in $\Sigma^\dagger$ have decreasing order. The proof of $\|A\|_2 = \sigma_1$ was given in Theorem 5.4.1. The proof of $\left\|A^\dagger\right\|_2 = 1/\sigma_r$ is similar. Let $x \in \mathbb{R}^m$ be a unit vector and

$$\begin{aligned}(A^\dagger x)^T A^\dagger x &= x^T(V\Sigma^\dagger U^T)^T V\Sigma^\dagger U^T x \\ &= x^T U(\Sigma^\dagger)^T V^T V \Sigma^\dagger U^T x \\ &= x^T U(\Sigma^\dagger)^T \Sigma^\dagger U^T x.\end{aligned}$$

Use identity 6 in Theorem 7.1.1 to note the largest diagonal component in $(\Sigma^\dagger)^T\Sigma^\dagger$ is $1/\sigma_r^2$. Since the columns of U are an orthonormal basis, $x = Uc$ with $c = U^T x$ and $c^T c = x^T x = 1$. Thus

$$(A^\dagger x)^T A^\dagger x \le x^T x(1/\sigma_r^2) \text{ and } \left\|A^\dagger\right\|_2 \le 1/\sigma_r.$$

In order to obtain the inequality in the other direction, choose $x = u_r$ to be column r in U. Then $U^T x = e_r$ and $(A^\dagger x)^T A^\dagger x = 1/\sigma_r^2$. ■

Consider $x^+ = A^\dagger d$ where there is a variation in the right side

$$\begin{aligned}(x + \Delta x)^+ &= A^\dagger(d + \Delta d) \\ &= A^\dagger d + A^\dagger \Delta d \\ &= x^+ + A^\dagger \Delta d \\ \Delta(x^+) &\equiv (x + \Delta x)^+ - x^+ = A^\dagger \Delta d.\end{aligned}$$

Theorem 8.1.2 *If A is an $m \times n$ matrix with rank$(A) = r$ and $d, \Delta d \in R(A)$, then for $x^+ = A^\dagger d$*

$$\frac{1}{cond(A)}\frac{\|\Delta d\|_2}{\|d\|_2} \le \frac{\|\Delta(x^+)\|_2}{\|x^+\|_2} \le cond(A)\frac{\|\Delta d\|_2}{\|d\|_2}.$$

Proof. Apply the 2-norm to $\Delta(x^+) = A^\dagger \Delta d$

$$\left\|\Delta(x^+)\right\|_2 \le \left\|A^\dagger\right\|_2 \|\Delta d\|_2 \le (1/\sigma_r)\|\Delta d\|_2.$$

Use $d \in R(A)$ and Theorem 7.1.2 to write $Ax^+ = A(A^\dagger d) = d$. Apply the 2-norm

$$\begin{aligned}\|d\|_2 &\le \|A\|_2 \left\|x^+\right\|_2 \le \sigma_1 \left\|x^+\right\|_2 \text{ or} \\ \frac{1}{\|x^+\|_2} &\le \frac{\sigma_1}{\|d\|_2}.\end{aligned}$$

Combine these to get the desired upper inequality.

Apply the 2-norm to $x^+ = A^\dagger d$

$$\begin{aligned}\left\|x^+\right\|_2 &\le \left\|A^\dagger\right\|_2 \|d\|_2 \le (1/\sigma_r)\|d\|_2 \text{ or} \\ \sigma_r/\|d\|_2 &\le 1/\left\|x^+\right\|_2\end{aligned}$$

Use $\Delta d \in R(A)$ and Theorem 7.1.2 to write $A\Delta(x^+) = A(A^\dagger \Delta d) = \Delta d$. Apply the 2-norm

$$\begin{aligned} \|\Delta d\|_2 &\leq \|A\|_2 \left\|\Delta(x^+)\right\|_2 \leq \sigma_1 \left\|\Delta(x^+)\right\|_2 \text{ or} \\ (1/\sigma_1)\|\Delta d\|_2 &\leq \left\|\Delta(x^+)\right\|_2 . \end{aligned}$$

These two inequalities give the lower inequality. ■

The above result for the least squares solution does not include variations in the least squares solution because of the possible changes in the matrix. The following result can be established for nonsingular $n \times n$ matrices. Consider $Ax = d$ and $(A + \Delta A)(x + \Delta x) = d + \Delta d$ and assume

$$\left\|A^{-1}\right\|_2 \|\Delta A\|_2 < 1.$$

Then $A + \Delta A$ has an inverse and one can eventually show

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leq \frac{cond(A)}{1 - \|A^{-1}\|_2 \|\Delta A\|_2}\left(\frac{\|\Delta d\|_2}{\|d\|_2} + \frac{\|\Delta A\|_2}{\|A\|_2}\right).$$

A possible extension of this to the general least squares problem is interesting!

8.2 Application to Hazard Identification

Consider a governing differential equation for the hazard, and discretize it to obtain a linear system

$$Au = d.$$

The coefficient matrix A is derived using finite differences with upwind finite differences on the velocity term. The coefficient matrix will be $n \times n$, and the sites or nodes will be partitioned into three ordered sets, whose order represents a reordering of the nodes,

osites	observe sites
ssites	source sites and
rsites	remaining sites.

The *hazard identification* problem is given data $data$ at $osites$, find $d(ssites)$ such that

$$Au = d \text{ and } u(osites) = data.$$

In general, the reordered matrix has the following 2×2 block structure with $other = [ssites \quad rsites]$

$$\begin{aligned} A &= \begin{bmatrix} a & e \\ f & b \end{bmatrix} \text{ where} \\ a &= A(osites, osites), \\ e &= A(osites, other), \\ f &= A(other, osites) \text{ and} \\ b &= A(other, other). \end{aligned}$$

Assume the following are true:

(i). A and a have inverses.

(ii). $\#(osites) = k$, $\#(ssites) = l < k$ and $\#(rsites) = r$ so that $n = k + l + r$.

(iii). $d = \begin{bmatrix} 0 & d_1^T \end{bmatrix}^T$ and the only nonzero terms in d_1 are at the nodes in $ssites$, that is,

$$d \equiv \begin{bmatrix} 0 & d_1(ssites)^T & 0 \end{bmatrix}^T.$$

Multiply the above equation by a block elementary matrix

$$\begin{bmatrix} I_k & 0 \\ -fa^{-1} & I_{n-k} \end{bmatrix} \begin{bmatrix} a & e \\ f & b \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} = \begin{bmatrix} I_k & 0 \\ -fa^{-1} & I_{n-k} \end{bmatrix} \begin{bmatrix} 0 \\ d_1 \end{bmatrix}$$

$$\begin{bmatrix} a & e \\ 0 & \widehat{b} \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} = \begin{bmatrix} 0 \\ d_1 \end{bmatrix} \text{ and } \widehat{b} \equiv b - fa^{-1}e. \quad (8.2.1)$$

Solve for u_1 and then $u_0 = -a^{-1}e(\widehat{b})^{-1}d_1$. Define

$$d_1 \equiv [d_1(ssites)^T \quad 0]^T \text{ and } C \equiv -a^{-1}e(\widehat{b})^{-1}.$$

The computed approximations of the observed *data* is in u_0 and the source data is in $z = d_1(ssites)$. This gives a *least squares* problem for z

$$C(:, ssites)z = data.$$

Let $d_1(ssites) = z$ and use the notation $d(z) = [0 \quad z^T \quad 0]^T$.

Since a is $k \times k$, e is $k \times (n-k)$ and $\widehat{b}$ is $(n-k) \times (n-k)$, the matrix C is $k \times (n-k)$ and the *least squares matrix* $C(:, ssites)$ is $k \times l$ where $k > l$. The least squares problem will have a unique solution if the columns of $C(:, ssites)$ are linearly independent ($C(:, ssites)z = 0$ implies $z = 0$). The condition number will depend on the parameters in the differential equation as well as the position of the observation sites relative to the source sites. Here both the least squares matrix and the right side have variations. If there are insufficient source sites, $k > l$, then the pseudoinverse of $C(:, ssites)$ can be used to find the minimal 2-norm sources.

A more general discussion of the topic can be found in [15].

8.3 Matlab Code hazidsvd1.m

Lines 1-33 define the 100×100 matrix associated with the differential equation. Line 34 locates the three source sites, and lines 35-38 give three possible observation sites and intensities. Lines 39-53 computes the new source numbers, which are caused by the reordering. The reordered matrix is computed in line 57-61. The least squares matrix is computed in lines 62-64. The loop in lines 71-76 solves the least square problem 100 times with variable noise in the data. Lines 82 and 83 display the mean and standard deviation of the computed intensities at the source sites.

```
clear; clf(figure(1));
% hold on
% This code illustrates the linear least squares problem to
% identify the sources given observations. This is from
%   -(K u_x)_x + vel u_x + ru = point sources.
% The intensities at the point source sites are to be found
% from data at observation sites.
% The 1D steady state problem with fixed locations and
% unknown intensities is solved.
 %
 % Input data and system matrix
 %
 n = 100; L = 2.0; dx = L/n;
 K     = 0.005;    % diffusion
 vel   = 0.10;     % flow rate
 r     = 0.0200;   % decay rate
 noise = 0.10      % percent noise level
 A = zeros(n);
 for i = 1:n
     if i == 1
         A(i,i) = K*2/dx^2 + vel/dx + r;
         A(i,i+1) = -K/dx^2;
     end
     if i>1 && i<n
         A(i,i) = K*2/dx^2 + vel/dx + r;
         A(i,i+1) = -K/dx^2;
         A(i,i-1) = -K/dx^2 - vel/dx;
     end
     if i == n
         A(i,i) = K*2/dx^2 + 2*vel/dx + r;
         A(i,i-1) = -K*2/dx^2 - 2*vel/dx;
     end
 end
 ssites = [20 35 80 ];      % source sites
 osites = [10 30 60 90];    % observation sites
% osites = [10 30 60 70];
% osites = [10 30 ];
 p = [1 5 4]                % intensities of souces
 [ms ns] = size(ssites)
 [mo no] = size(osites)
 % Find shifted ssites
 newssites = ssites;
 for js = 1:ns
     for jo = 1:no
         if osites(jo) < ssites(js)
```

```
            newssites(js) = newssites(js) - 1;
        end
    end
end
newssites
other = setdiff(1:n, osites);
set1 = union(osites,ssites);
rsites = setdiff(1:n, set1);
%
% Computation of reordered matrix and least squares matrix.
%
a = A(osites,osites);
b = A(other,other);
e = A(osites,other);
f = A(other,osites);
AA   = [ a e; f b];
bhat = b - f*inv(a)*e;
C    = -inv(a)*e*inv(bhat);
CLS  = C(:,newssites);
d    = zeros(n,1);
[U S V] = svd(CLS)
% Dirac delta appoximations
d(ssites)= p/dx;
u = A\d;
% Simulations with noise 100 executions
for kk = 1:100
    data = u(osites);
    data = data + data*(noise).*(rand(jo,1) - .5);
    z = pinv(CLS)*data;
    zz(1:ns,kk) = z;
end
display('U^T*data = ')
U'*data
%
% Numeric and graphical output
%
meanzz = mean(zz')
stdzz  = std(zz')
display('u at ssites = ')
u(ssites)
res_error = data - CLS*z
rank_num = rank(CLS)
cond_num = cond(CLS)
x = (1:n)*dx;
plot(u)
```

```
dd = zeros(n,1);
dd(ssites) = z;
uu = A\dd;
hold on
plot(uu,'k:')
plot(osites,4,'*')
```

In the following numerical experiments there are four or two observation sites and three source sites. So the least square matrix CLS will be either 4×3 or 2×3. The pseudoinverse times $data$ can be written as

$$CLS^{\dagger}\ data = v_1(\frac{u_1^T d}{\sigma_1}) + v_2(\frac{u_2^T d}{\sigma_2}) + v_3(\frac{u_3^T d}{\sigma_3}).$$

In the second numerical experiment the fourth observation site is poorly placed. This results in small third singular value, and the third term in the above being prone to computation errors.

The first numerical experiment uses observation sites 10, 30, 60 and 90 and source sites 20, 35 and 80. The graphical output is given in Figure 8.3.1 and some of numerical output is given below. The mean computed intensities are close to the exact values of 50, 250 and 200. Moreover, the standard deviations is small.

```
>> hazidsvd1
noise =
    0.1000
ssites =
    20    35    80
osites =
    10    30    60    90
p =
     1     5     4
newssites =
    19    33    77
U =
   -0.0120    0.0203    0.0339   -0.9991
   -0.4116    0.5815    0.7006    0.0405
   -0.5921    0.4142   -0.6912   -0.0079
   -0.6927   -0.6999    0.1739    0.0000
S =
    0.3830         0         0
         0    0.1520         0
         0         0    0.0880
         0         0         0
V =
   -0.7272    0.4910    0.4797
```

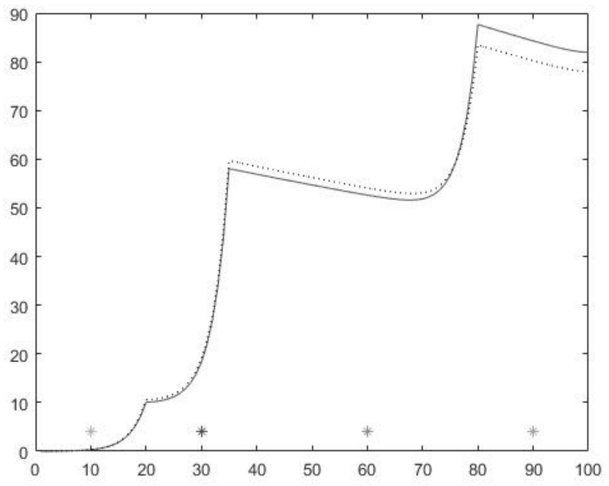


Figure 8.3.1: Hazard Id with osites = [10 30 60 90]

```
   -0.5963   -0.1056   -0.7958
   -0.3401   -0.8647    0.3695
U^T*data =
  -96.8285
  -28.6826
   -6.7047
    0.0194
meanzz =
   49.6993  250.0798  201.6292
stdzz =
    3.8101   10.8717   14.8688
u at ssites =
   10.0538
   58.0373
   87.6532
rank_num =
     3
cond_num =
    4.3520
```

The second numerical experiment uses observation sites 10, 30, 60 and 70 and source sites 20, 35 and 80. The graphical output is given in Figure 8.3.2 and some of numerical output is given below. The mean computed intensities are close to the exact values of 50, 250 and 200. However, the standard deviation is larger, and this is more pronounced in the third intensity where the third singular value is small. The condition number for this least squares matrix is larger, and the rank is still equal to 3.

```
>> hazidsvd1
noise =
    0.1000
ssites =
    20    35    80
osites =
    10    30    60    70
p =
     1     5     4
newssites =
    19    33    76
U =
   -0.0131    0.0388    0.0059   -0.9991
   -0.4497    0.8922    0.0005    0.0405
   -0.6438   -0.3237   -0.6933   -0.0082
   -0.6190   -0.3124    0.7206    0.0003
S =
    0.3741         0         0
         0    0.1012         0
         0         0    0.0045
         0         0         0
V =
   -0.7775    0.6289    0.0044
   -0.6288   -0.7772   -0.0231
   -0.0111   -0.0207    0.9997
U^T*data =
  -73.1552
  -16.6344
    2.7738
    0.0029
meanzz =
   49.6001  249.5465  214.6239
stdzz =
    4.1013   12.0458  380.1449
u at ssites =
   10.0538
   58.0373
   87.6532
rank_num =
     3
cond_num =
```

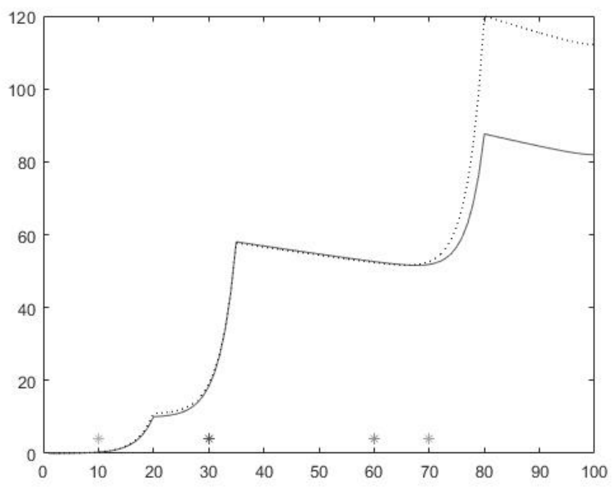


Figure 8.3.2: Hazard Id with osites = [10 30 60 70]

```
    82.5861
```

The third numerical experiment uses observation sites 10 and 30 and source sites 20, 35 and 80. The graphical output is given in Figure 8.3.3 and some of numerical output is given below. The mean computed intensities are close to the exact values of 50, 250 but not 200. The condition number of the least square matrix has increased and the rank is 2. The general least squares solution is

$$CLS^{\dagger}\ data + c\ v_3 \text{ where } CLS\ v_3 = 0_{2\times 1}.$$

The minmal 2-norm least squares solution follows from

$$z + cv_3 = \begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}^T + c\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T.$$

Then $c = -z_3$ and $z + cv_3 = \begin{bmatrix} z_1 & z_2 & 0 \end{bmatrix}^T$.

```
>>hazidsvd1
noise =
    0.1000
ssites =
    20    35    80
osites =
    10    30
p =
     1     5     4
```

```
newssites =
    19    33    78
U =
   -0.0324    0.9995
   -0.9995   -0.0324
S =
    0.1911         0         0
         0    0.0011         0
V =
   -0.9825    0.1863    0.0000
   -0.1863   -0.9825   -0.0000
   -0.0000   -0.0000    1.0000
U^T*data =
  -17.6769
   -0.2616
meanzz =
   49.6716  251.3730    0.0001
stdzz =
    1.5527   17.3989    0.0000
u at ssites =
   10.0538
   58.0373
   87.6532
rank_num =
     2
cond_num =
  168.4529
```

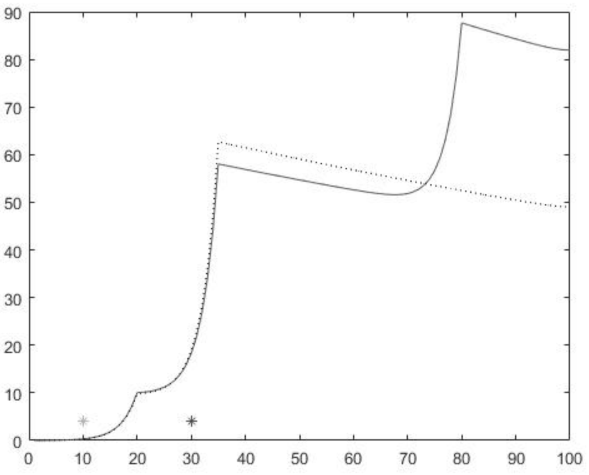


Figure 8.3.3: Hazard Id with osites = [10 30]

Chapter 9

Nonlinear LS and Epidemic Models

The Levenberg-Marquardt algorithm will be used to approximate the solutions to nonlinear least squares problems. Application to parameter identification of differential equations will be illustrated. In particular, the elementary models of epidemics will be discussed. This includes SIR and SIRD (death is allowed). The US data for the COVID-19 virus will be used in a variation of the SIRD to estimate the parameters. Two limitations of this model are non homogeneous population and time dependent parameters. Here the matrices are ill-conditioned, and the effects of uncertain data are noted. This model does seem to give a limited prediction about the aggregated US COVID-19 infection and with select time intervals.

9.1 Levenberg-Marquardt Algorithm

The objective is to identify n parameters from m measured data at time or space. The model is usually an ODE or a PDE.

$y \in \mathbb{R}^m$ measured data vector with $m > n$,
$p \in \mathbb{R}^n$ is the parameter vector,
$f(p,t)$ real solution from a DE with given parameters and
error or *residual* $\equiv y_i - f(p,t_i)$.

Nonlinear Least Squares Problem.

Find p so that the residual in a minimum

$$\min \sum_{i=1}^{m} (y_i - f(p,t_i))^2.$$

Use a Linear Approximation.

$f(p,t_i) \simeq f(p^0,t_i) + f'(p^0,t_i)(p-p^0)$ and

$f'(p^0,t_i) \equiv [f_{p_j}(p^0,t_i)]$ is an $m \times n$ matrix of partial derivatives.

$$\min \sum_{i=1}^{m} (y_i - (f(p^0, t_i) + f'(p^0, t_i)(p - p^0)))^2$$
$$= \min \sum_{i=1}^{m} ((y_i - f(p^0, t_i)) - f'(p^0, t_i)(p - p^0))^2.$$

This is a least squares problem $A\Delta p = d$ where
$d = [y_i - f(p^0, t_i)]$ and $A \equiv [f_{p_j}(p^0, t_i)]$.
Use Theorem 1.1.1 to conclude Δp is the solution of the normal equations.

Gauss-Newton Algorithm. Let $A \equiv [f_{p_j}(p^k, t_i)]$.
$p^{k+1} = p^k + \Delta p$ where $A^T A \Delta p = A^T(y - f(p^k, t_i))$.

Notation. Let $F(p) \equiv y - f(p)$. $F'(p) = 0 - f'(p)$ and so the least squares problem is
$F'(p)^T F'(p) \Delta p = -F'(p)^T (F(p))$. If $F'(p)$ has full column rank, then

$$p^{k+1} = p^k - (F'(p)^T F'(p))^{-1} F'(p)^T (F(p)).$$

This method does not always converge to the solution of nonlinear least squares problem. The following example has $m = 2$ and $n = 1$ with $y = 0_{2\times 1}$.

Example 1 *Let* $f : \mathbb{R}^1 \to \mathbb{R}^2$

$$\begin{aligned} f(p) &= \begin{bmatrix} p+1 \\ ap^2 + p - 1 \end{bmatrix}. \\ r &= y - f(p) \\ \min r^T r &= \min(\frac{1}{2}(p+1)^2 + \frac{1}{2}(ap^2 + (p-1))^2). \end{aligned}$$

Define $G(p) = r^T r$ *and note* $G'(0) = 0$ *and* $G''(0) = 0$ *for* $a < 1$. *Thus* $p = 0$ *is a solution. However, the Gauss-Newton method fails for* $a < -1$ *as is illustrated by the following calculation which oscillates between* $+1$ *and* -1.

```
clear
a = -1.1; p = .1;
for k = 1:100
   F = -[p+1; a*p^2+p-1];
   Fp = -[1; 2*a*p+1];
   newp = p - Fp'*F/(Fp'*Fp);
   p = newp;
   pp(k) = p;
end
plot(pp)
```

The Levenberg-Marquardt algorithm has two enhancements: the α and λ real parameters. These can be adjusted within the interation so to accelerate or to minimize the residual. The parameters were introduced by K. Levenberg (1944) and by D. Marquardt (1963); see [9, section 8.5]. Let $y \in \mathbb{R}^m$ be the data and let $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ be a function of the unknown parameters p. Approximate the solution of

$$\min_p (y - f(p))^T (y - f(p)).$$

Levenberg-Marquardt Algorithm. *Let $F(p) = y - f(p)$.*

```
$p^0 \in \mathbb{R}^n$ initial estimate for the parameters
for $k = 1, maxk$
    choose $\alpha_k$ and $\lambda_k$
    compute $F(p^k)$ and the $m \times n$ matrix $F'(p^k)$
    solve the least squares problem

        $(F'(p^k)^T F'(p^k) + \lambda_k I_n)\Delta p = F'(p^k)^T F(p^k)$

    $p^{k+1} = p^k - \alpha_k \Delta p$
    test for convergence
end
```

Example 2 *Return to Example 2 in Section 1.1 and 1.2. The model for price prediction based on six previous observations was a discrete model, which could be viewed as an approximation of a first order differential equation*

$$u' = c(p_{\min} - u) \text{ and } u(0) = 2080.$$

Here the parameters are c and $p_{\min}$. The solution of the differential equation is

$$\begin{aligned} u(t) &= (2080) - pmin) \; exp(ct) + pmin \\ &= (2080) - p_2) \; exp(p_1 t) + p_2. \end{aligned}$$

Here there are six measurements for $u(t_i)$ to approximate the two parameters. The matrix in the Levenberg-Marquardt algorithm is 6×2. The computer code levmarqprice.m is in the next section. One can use the results in price_ expdata.m as a starting point for the code in levmarqprice.m.

Example 3 *Consider over damped mass spring system where damping, spring constant, initial position and initial velocity are four unknown parameters. The model has the form*

$$mx^{''} + cx^{'} + kx = 0$$

If the position is measured at m time steps, find these parameters.

$$f(p, t) = p_1 \exp(p_2 t) + p_3 \exp(p_4 t).$$

The interested reader should download levmarqheat.m. Here there are eight measurements for $x(t_i)$ to approximate the four parameters. The matrix in the Levenberg-Marquardt algorithm is 8×4.

9.2 Matlab Code levmarqprice.m

Line 18 contains the initial estimate for the parameters. Here one could have used the results of the finite difference model in price_expdata.m. Line 20 contains constant values for α and λ. In more sophisticated implementations, these are adjusted within the algorithm's loop in lines 21-32. The algorithm converged in only 11 iterations, and this is in contrast to levmarqheat.m. The outputs for the residual and price at time equal to 8 are 40.2620 and 1812.6 from price_expdata.m, and 31.5225 and 1813.3 from levmarqprice.m.

```
1    % This code illustrates the Levenberg-Marquardt algorithm
2    % for curve fitting to price data (or Newton cooling).
3    % Consider the first order ODE: u' = c(pmin - u) and u(0) = 2080.
4    % Suppose c and pmin are unknown, and measured values for u(t_i)
5    % are known where i = 1:m. Approximate the two unknown
6    % parameters from the measured data.
7    %
8    % f_i(p1,p2,t) = (2080) - pmin) exp(ct) + pmin
9    %               =(2080 - p(2)) exp(p(1)t) + P(2)
10    % F(p1,p2,t_i) = udata(t_i) - f_i(p1,p2,t)
11    % FP is an mx2 matrix with components f_i,_p(1) and f_i,_p(2).
12 %
13 clear; clf;
14 % Define "data"
15 tdata = 0:1:5; tdata = tdata';
16 udata = [ 2080 2000 1950 1910 1875 1855]';
17 % Initial guess for parameters
18 p = [ -1 1500]';
19 % Begin Levenberg-Marquardt
20 lam = 0.01; alpha = 1.0;
21 for lm = 1:20
22   F = udata -((2080 - p(2))*exp(p(1)*tdata) + p(2));
23   FP = -[(2080 - p(2))*exp(p(1)*tdata).*tdata...
24        -exp(p(1)*tdata)+1];
25   newp = p - alpha*(FP'*FP + lam*eye(2))\(FP'*F);
26   error = norm(newp - p);
27   p = newp;
28   if error < 0.00001
29      break
30   end
31   % pause
32 end
33 lm
34 error
35 % Compare computed and initial parameters
36 plot(tdata,udata,'*')
```

```
hold on
time = [0:1:15];
newu = (2080 - p(2))*exp(p(1)*time) + p(2);
plot(time,newu, 'r')
title('Predicted Price Values by the ...
          Levenberg-Marquardt Algorithm')
xlabel('Time')
ylabel('Price')
pmin = p(2)
c = p(1)
display('Predicted price at 8 = ')
newu(9)
r = udata - newu(1:6)'
rTr = r'*r
cond(FP)
S = svd(FP)

>> levmarqprice
lm =
    11
error =
   3.5374e-06
pmin =
   1.7942e+03
c =
   -0.3095
Predicted price at 8 =
ans =
   1.8183e+03
r =
        0
   -3.9384
    1.8776
    2.8369
   -2.1060
   -0.0493
rTr =
   31.5225
ans =
   2.3261e+03
S =
  675.1095
    0.2902
```

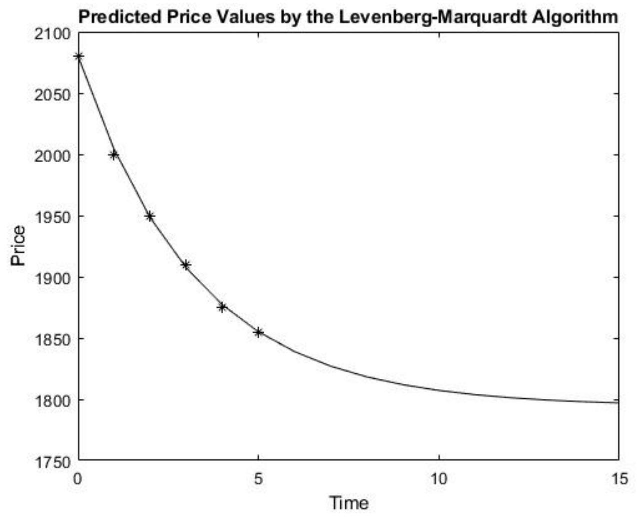


Figure 9.2.1: Levenberg-Marquardt Prediction of Price

9.3 SIRD Epidemic Models

The SIRD model has four differential equations for four unknown functions of time. The functions are $S(t)$ = susceptible, $I(t)$ = infected, $R(t)$ = recovered and $D(t)$ = death populations. If there are no deaths, this is called the SIR model. This was introduced by W. Kernmack and A. McKendrick (1927); for a more recent discussion see [11, section 1.2].

There are four primary components that govern the growth of an infected population: probability of transmission, the degree of susceptibility, number of contacts with an infected, and duration of contacts. These components determine the size of the three parameters, a, b_r and b_d, in the following system of differential equations.

SIRD Model.

$$\begin{aligned} \frac{dS}{dt} &= -aSI, \\ \frac{dI}{dt} &= aSI - b_r I - b_d I, \\ \frac{dR}{dt} &= b_r I \text{ and} \\ \frac{dD}{dt} &= b_d I. \end{aligned}$$

The parameter a is often called the "effective contact" because it has the primary components of probability, susceptibility and contacts. The parameters

b_r and b_d are the removal rates from the infected to either the recovered or death populations. Their reciprocals determine the durations of the infected. Here we assume the four primary components are fixed, and the three parameters are constants.

The populations move from S to I and then to either R or D. S is a decreasing function of time, and R and D are increasing functions of time. The total population must be constant because the sum of the derivatives is zero. The equation of the infected may be written as

$$\frac{dI}{dt} = I(aS - b_r - b_d) \text{ and } I \text{ is positive.}$$

So, if the second factor is positive (negative), then I will increase (decrease). This may also be written as

$$\frac{aS(t)}{b_r + b_d} > 1 \text{ (or } < 1\text{) for } I \text{ to increase (or decrease).}$$

Initially, the infection will increase only if $I(0) > 0$ and $aS(0) - b_r - b_d > 0$. The infected will be a maximum at time $t_{\max}$ where $aS(t_{\max}) - b_r - b_d = 0$, which follows from

$$\frac{dI}{dt}(t_{\max}) = 0 \text{ and } \frac{d^2 I}{dt^2}(t_{\max}) < 0.$$

An alternate analysis is to determine if a single infected will infect more than one susceptible. This can be broken into product of three factors:
(new infections per contact)
(contacts per time) and
(duration time per infected).
In the SIR model where $b_d = 0$, this is $a(1/b) > 1$.

Because this model is nonlinear, one must approximate the solution using numerical methods. In the next section this is done using Runge-Kutta variable step size method. The first calculation illustrates the effect of decreasing the effective contact parameter, see Figure 9.3.1. Note, the maximum of the infection decreases and moves to the right. The other calculations in the next section determine the three parameters resulting from additional observations.

9.4 Matlab Code sird_parid.m

The primary objective of this code is to illustrate how the Levenberg-Marquardt algorithm can be used to approximate the three parameters in the SIRD model. This is done in lines 117-156. For each of the four functions there are 12 data points. So, the objective function is a 48×1 column vector and the derivative matrix is 48×3. The columns in the derivative matrix are partial derivatives, and they are approximated by finite differences (can be tricky).

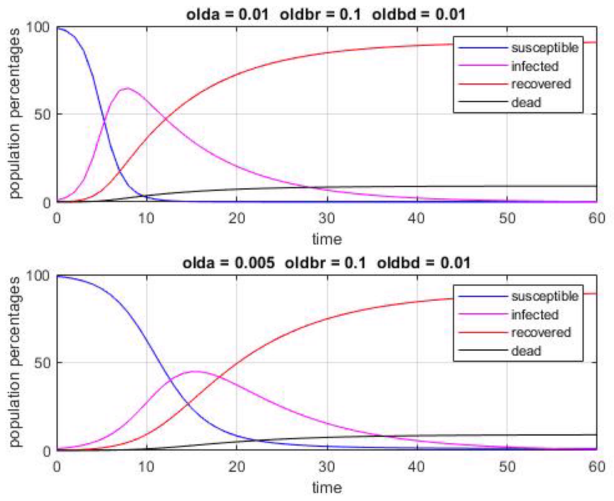


Figure 9.3.1: Variable Effective Contact Parameter

Lines 37-68 solves the SIRD model for given parameters, and this was used to generate Figure 9.3.1. Lines 70-92 define test data (would be measured values of the four functions). Lines 98-116 give a first estimate of three parameters using the first order finite difference approximation of the SIRD system. This is gives a least squares problem, which is solved in line 111. These are used as an initial estimate for the more accurate Levenberg-Marquardt algorithm.

The implementation requires many evaluations of the objective function and the derivative matrix. These are done using the higher order ode45 implementation of the variable step size Runge-Kutta method. See lines 117, 131,136, 141 and 149.

The output is given in Figure 9.4.1 where the data points are indicated by the isolated points. The three computed parameters are listed at the top of the figure. The four curves were computed using these parameters. The reader will find it interesting to experiment with different numbers of data points.

```
1 % This code uses least squares to identify three parameters
2 % in the SIRD model:
3 %   S_t = -a SI,
4 %   I_t = a SI - (br + bd) I,
5 %   R_t = br I and
6 %   D_t = bd I where
7 %         a = "contagious" (effective contact) parameter,
8 %         br = "recovery" parameter and
9 %         bd = "death" parameter.
10 % The data is given in the vectors Sd, Id Rd and Dd,
% and they are adjusted by a random variable.
% The data is used in the finite difference approximation:
%   (S_i+1 - S_i-1)/(2 dt) = -a S_i I_i,
%   (I_i+1 - I_i-1)/(2 dt) = a S_i I_i - (br + bd) I_i,
%   (R_i+1 - R_i-1)/(2 dt) = br I_i an
%   (D_i+1 - D_i-1)/(2 dt) = bd I_i.
% Least squares is used to compute the coefficients.
% The variable data points can be used.
%
% function [t y] = sirdid
%   global olda oldbr oldbd so io ro do t0 y0 tend
%   y0 = [so io ro do];
%   to = 0;
%   tf = 60;
%   opts = odeset('RelTol',1e-8,'AbsTol',1e-8);
%   [t y] = ode45('ypsirdid',[to:1:tf],y0,opts);
%
% function ypsirdid = ypsirdid(t,y)
%   global olda oldbr oldbd so io ro do t0 y0 tend
%   ypsirdid(1) = -olda*y(1)*y(2);
%   ypsirdid(2) = olda*y(1)*y(2) - (oldbr + oldbd)*y(2);
%   ypsirdid(3) = oldbr*y(2);
%   ypsirdid(4) = oldbr*y(2);
%   ypsirdid = [ypsirdid(1) ypsirdid(2)...
%               ypsirdid(3) ypsirdid(4)]';
%
clear; clf(figure(1)); clf(figure(2));
global olda oldbr oldbd so io ro do t0 y0 tend
figure(1)
%
olda = 0.01; oldbr = 0.100; oldbd = 0.010;
io = 1;
ro = 0;
do = 0;
so = 100 - io - ro - do;
t0 = 0; tend = 60;
[t y] = sirdid;
subplot(2,1,1)
plot(t,y(:,1),'b', t,y(:,2),'m', t,y(:,3),'r', t,y(:,4),'k')
title(['olda = ' ,num2str(olda),' oldbr = ' ,num2str(oldbr),...
' oldbd = ' ,num2str(oldbd)]);
ylabel('population percentages')
xlabel('time')
legend('susceptible', 'infected', 'recovered', 'dead');
grid on
```

```
%
% Decrease "effective contact"
%
olda = 0.005; oldbr = 0.100; oldbd = 0.010;
[t y] = sirdid;
subplot(2,1,2)
plot(t,y(:,1),'b', t,y(:,2),'m', t,y(:,3),'r', t,y(:,4),'k')
title(['olda = ' ,num2str(olda),' oldbr = ' ,num2str(oldbr),...
' oldbd = ' ,num2str(oldbd)]);
ylabel('population percentages')
xlabel('time')
legend('susceptible', 'infected', 'recovered', 'dead');
grid on
%
% Test Data
%
nd = 12;
td = 2:2:(2*nd);
%td = 1:nd;
Id = [1.00 2.56 6.37 14.68 29.20 46.35 58.19 61.64 ...
      59.35 54.50 48.88 43.32 38.15];
Rd = [0.00 0.22 0.77 2.09 4.93 9.93 16.92 24.91 32.94 ...
      40.07 47.29 53.37 58.75];
Dd = [0.000 0.008 0.031 0.083 0.197 0.396 0.675 0.994 1.315 ...
      1.615 1.888 2.131 2.345];
Sd = 100 - Id - Rd - Dd;
rvec = rand(1,nd);
Id(1:nd) = Id(1:nd) + Id(1:nd).*(2*rvec - 1)/100;
rvec = rand(1,nd);
Rd(1:nd) = Rd(1:nd) + Rd(1:nd).*(2*rvec - 1)/100;
rvec = rand(1,nd);
Dd(1:nd) = Dd(1:nd) + Dd(1:nd).*(1*rvec )/50;
Sd = 100 - Id - Rd - Dd;
io = Id(1);
ro = Rd(1);
do = Dd(1);
so = 100 - io - ro - do;
%
% Parameter ID From First Order Finite Difference
%
for i = 2:1:nd-1
  ii = (i-1)*4;
  d(ii) = (Sd(i+1) - Sd(i-1))/(td(i+1) - td(i-1));
  d(ii+1) = (Id(i+1) - Id(i-1))/(td(i+1) - td(i-1));
  d(ii+2) = (Rd(i+1) - Rd(i-1))/(td(i+1) - td(i-1));
  d(ii+3) = (Dd(i+1) - Dd(i-1))/(td(i+1) - td(i-1));
  A(ii,1) = -Sd(i)*Id(i); A(ii,2) = 0; A(ii,3) = 0;
  A(ii+1,1)= Sd(i)*Id(i); A(ii+1,2) = -Id(i); ...
  A(ii+1,3) = -Id(i);
  A(ii+2,1)= 0.0; A(ii+2,2) = Id(i); A(ii+2,3) = 0;
  A(ii+3,1)= 0.0; A(ii+3,2) = 0; A(ii+3,3) = Id(i);
end
%
meas = nd - 2;
m = 4*meas + 1;
x = A(2:m,:)\d(2:m)'; % solves the least squares
olda = x(1);
oldbr = x(2);
oldbd = x(3);
display('Parameters from finite difference')
[olda oldbr oldbd]
[t y] = sirdid; % SIRD with new parameters
dd = [ Sd(1:nd)'; Id(1:nd)'; Rd(1:nd)'; Dd(1:nd)'];
sol = [y(td,1); y(td,2); y(td,3); y(td,4)]; % computed
FF = dd - sol; % residual
display('Norm of residual')
norm(FF)
%
% Parameter ID from Levenberg-Marquardt Algorithm Using ODE45
%
lam = 0.1; alpha = 4.0; p = [x(1) x(2) x(3)]';
%pause
for lm = 1:100
  da = 0.001;
  olda = p(1) + da;,oldbr = p(2); oldbd = p(3);
  [t yp] = sirdid;
  solp = [yp(td,1); yp(td,2); yp(td,3); yp(td,4)];
  FFa = (solp - sol)/da;
  dbr = 0.01;
  oldbr = p(2) + dbr; olda = p(1); oldbd = p(3);
  [t yp] = sirdid;
  solp = [yp(td,1); yp(td,2); yp(td,3); yp(td,4)];
  FFbr = (solp - sol)/dbr;
  dbd = 0.001;
  oldbd = p(3) + dbd; olda = p(1); oldbr = p(2);
  [t yp] = sirdid;
  solp = [yp(td,1); yp(td,2); yp(td,3); yp(td,4)];
  FFbd = (solp - sol)/dbd;
  FFP = -[FFa FFbr FFbd];
  newp = p - alpha*(FFP'*FFP + lam*eye(3))\FFP'*FF;
  error = norm(newp - p)/norm(p);
  p = newp;
  olda = p(1); oldbr = p(2); oldbd = p(3);
  [t y] = sirdid;
  sol = [y(td,1); y(td,2); y(td,3); y(td,4)];
  FF = dd - sol;
  norm(FF);
  if error < 0.00001
    break
  end
end
display('Parameters from Levenberg-Marquardt')
olda = p(1);
oldbr = p(2);
oldbd = p(3);
[olda oldbr oldbd]
display('Norm of residual from Levenberg-Marquardt')
norm(FF)
error;
lm
%
figure(2)
plot(td(1:1:meas+1),Sd(1:1:meas+1),'*',td(1:1:meas+1),...
Id(1:1:meas+1),'o',...
td(1:1:meas+1),Rd(1:1:meas+1),'s', td(1:1:meas+1),...
Dd(1:1:meas+1),'d')
hold on
% plot(td,Sd,'x', td,Id,'x', td,Rd,'x', td,Dd,'x')
[t y] = sirdid;
plot(t,y(:,1),'b' ,t,y(:,2),'m' ,t,y(:,3),'r' ,t,y(:,4),'k')
title(['a = ' ,num2str(olda),' br = ' ,num2str(oldbr),...
' bd = ' ,num2str(oldbd)]);
ylabel('population percentages')
xlabel('time')
legend('susceptible data', 'infected data',...
'recovered data', 'dead data');
grid on
svd(A(2:m,:));
cond(A(2:m,:));
svd(FFP);
cond(FFP);
```

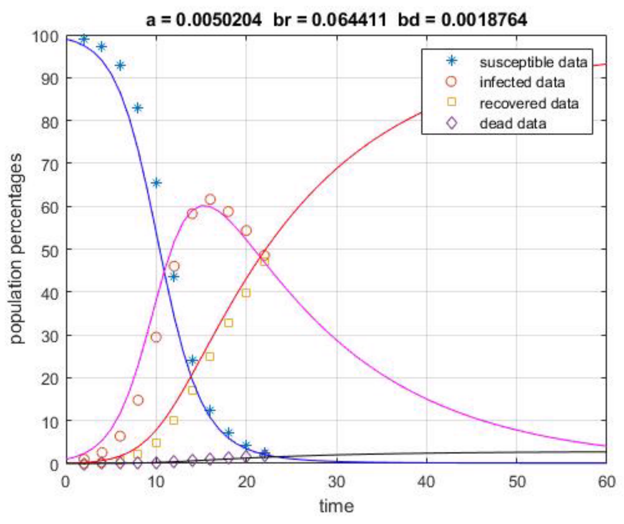


Figure 9.4.1: Parameter Identification

9.5 The Cumulated Infection Version of SIRD

The motivation for introducing the cumulated infection model is to use "smoother" data than is given by daily data for the four SIRD functions of time. Figure 9.5.1 displays the daily infected and death data for the US COVID-19. Note the jagged appearance. By taking the integral or cumulated sum of the data one obtains a continuous and strictly increasing function of time. This section shows the four functions of time can be expressed as function of the cumulated infection, which evolves from a nonlinear differential equation, CI-equation, with the three parameters of the SIRD model.

Consider the first equation in the SIRD model.

$$\begin{aligned}\frac{1}{S}\frac{dS}{dt} &= -aI\\ \int_{t_0}^{t}\frac{1}{S}\frac{dS}{dt} &= -a\int_{t_0}^{t} I = -aC(t)\\ \ln(S(t)/(S(t_0)) &= -aC(t).\end{aligned}$$

Now solve for $S(t), R(t)$ and $D(t)$ in terms of the cumulated infection $C(t)$:

$$\begin{aligned}S(t) &= S(t_0)\exp(-aC(t)),\\ R(t) &= R(t_0)+b_rC(t) \text{ and}\\ D(t) &= R(t_0)+b_rC(t).\end{aligned}$$

In order to find $C(t)$ without explicitly knowing I, use the fact that $S+I+$

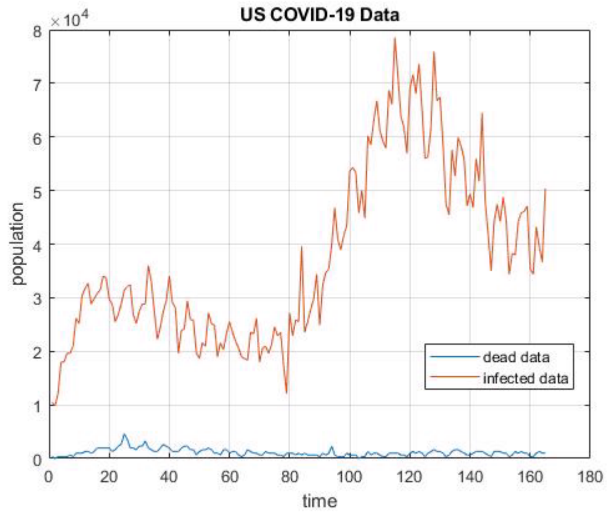


Figure 9.5.1: Daily Infected and Death Data

$R + D$ is a constant, say *popmax*. Solve for I and use the above

$$\begin{aligned} I &= popmax - S - R - D \\ &= popmax - S(t_0)\exp(-aC(t)) \\ &\quad - (R(t_0) + b_r C(t)) - (R(t_0) + b_r C(t)) \,. \end{aligned}$$

$C(t)$ is the integral of I, and therefore the derivative of $C(t)$ is I. This gives the nonlinear differential equation for the cumulative infection.

Cumulative Infection Equation, CI-equation.

$$\begin{aligned} \frac{dC}{dt} &= popmax - S(t_0)\exp(-aC(t)) \\ &\quad - (R(t_0) + b_r C(t)) - (R(t_0) + b_r C(t)) \\ &= I(t_0) + S(t_0)(1 - \exp(-aC(t))) \\ &\quad -b_r C(t) - b_d C(t) \text{ and} \\ &\quad C(t_0) \text{ is given.} \end{aligned}$$

If the initial values $I(t_0), R(t_0), D(t_0)$ and the cumulative function at $C(t_0)$ as well as the three parameters are known, then one can solve the CI-equation for $C(t)$ with $t \geq t_0$. Furthermore, by using cumulative infection and cumulative death data near to t_0, the three parameters can be identified.

9.6 Matlab Code sird_paridc.m

The following computations solve the same problem as in Section 9.4, but here the CI-equation is used. The test data is defined in lines 34-58, and in line 47 the cumulated sum of the daily infected data is defined. The initial estimate of the three parameters is given in lines 59-67 and uses least squares. The objective and initial residual are defined in lines 73-75. Note the data for the cumulated infection and cumulated death are only used in these calculations. The Levenberg-Marquardt method is implemented in lines 79-115. Here the CI-equation is solved numerous times in lines 69, 85, 91, 97 and 107. The output is given in Figure 9.6.1 where the three parameter are identified using the indicated data points.

```
1 % This code uses least squares to identify three parameters
2 % in the SIRD model:
3 %   S_t = -a SI,
4 %   I_t = a SI - (br + bd) I,
5 %   R_t = br I and
6 %   D_t = bd I where
7 %         a = "contagious" (effective contact) parameter,
8 %         br = "recovery" parameter and
9 %         bd = "death" parameter.
10 % The data is given in the vectors Sd, Id Rd and Dd,
11 % and they are adjusted by a random variable.
12 %
13 % An equivalent system is for the cumulative sum or integral
14 % of the infection, the CI-equation:
15 %   C_t = popmax - S(0) - R(0) - D(0)
16 %         + S(0) (1 - exp(-aC) - br C - bd C, C(0) = 0.
17 % This requires initial cumulative infection and death data.
18 %
19 % function [t y] = sirdidc
20 %   global olda oldbr oldbd so io ro do popmax t0 y0 tend
21 %   y0 = io;
22 %   to = 0;
23 %   tf = 60;
24 %   opts = odeset('RelTol',1e-3,'AbsTol',1e-6);
25 %   [t y] = ode45('ypsirdidc',[to:1:tf],y0,opts);
26 %
27 % function ypsirdidcout = ypsirdidc(t,C)
28 %   global olda oldbr oldbd so io ro do popmax t0 y0 tend
29 %   ypsirdidcout = popmax - ro - do - so*exp(-olda*C)...
30 %                   - oldbr *C - oldbd*C;
31 %
32 clear; clf(figure(1));
33 global olda oldbr oldbd so io ro do popmax t0 y0 tend
```

```
%
% Test Data
%
nd = 12;
td = 1:nd;
Id = [1.00 2.56 6.37 14.68 29.20 46.35 58.19 61.64 ...
      59.35 54.50 48.88 43.32 38.15]';
Rd = [0.00 0.22 0.77 2.09 4.93 9.93 16.92 24.91 32.94 ...
      40.07 47.29 53.37 58.75]';
Dd = [0.000 0.008 0.031 0.083 0.197 0.396 0.675 0.994 1.315 ...
      1.615 1.888 2.131 2.345]';
popmax = 100;
Sd = popmax - Id - Rd - Dd;
Cd = [0; cumsum(Id(1:(nd)))];
rvec = rand(nd,1);
%Id(1:nd) = Id(1:nd) + Id(1:nd).*(2*rvec - 1)/100;
rvec = rand(nd,1);
%Rd(1:nd) = Rd(1:nd) + Rd(1:nd).*(1*rvec )/100;
rvec = rand(nd,1);
%Dd(1:nd) = Dd(1:nd) + Dd(1:nd).*(1*rvec )/50;
Sd = popmax - Id - Rd - Dd;
io = Id(1);
ro = Rd(1);
do = Dd(1);
so = popmax - io - ro - do;
%
% Initial Parameter ID from CI-equation formulation
%
oldbr = Cd\(Rd - Rd(1));
oldbd = Cd\(Dd - Dd(1));
Sd = popmax - (diff(Cd)) - Rd(1:(nd)) - Dd(1:(nd));
olda = Cd(1:(nd))\(-log(Sd/Sd(1)));
display('Initial Parameters a,br,bd ')
x = [olda oldbr oldbd]'
% SIRDc with initial parameters
[t yp] = sirdidc;
C = yp;
%dd = [Cd(1:nd); Rd(1:nd); Dd(1:nd)]; % objective
%solc = [C(td); ro + oldbr*C(td); do + oldbd*C(td)];
dd = [Cd(1:nd); Dd(1:nd)];         % objective
solc = [C(td); do + oldbd*C(td)];  % initial computed
FF = dd - solc;                    % initial residual
display('Norm of initial residual')
norm(FF)
%
```

```
% Parameter ID from Levenberg-Marquardt Algorithm
%
lam = 1.0; alpha = 2.0; p = x;
for lm = 1:100
  da = 0.001;
  olda = p(1) + da; oldbr = p(2); oldbd = p(3);
  [t yp] = sirdidc;
  %solp = [yp(td); ro + oldbr*yp(td); do + oldbd*yp(td)];
  solp = [yp(td); do + oldbd*yp(td)];
  FFa = (solp - solc)/da;
  dbr = 0.001;
  oldbr = p(2) + dbr; olda = p(1); oldbd = p(3);
  [t yp] = sirdidc;
  solp = [yp(td); do + oldbd*yp(td)];
  %solp = [yp(td); ro + oldbr*yp(td); do + oldbd*yp(td)];
  FFbr = (solp - solc)/dbr;
  dbd = 0.001 ;
  oldbd = p(3) + dbd; olda = p(1); oldbr = p(2);
  [t yp] = sirdidc;
  %solp = [yp(td); ro + oldbr*yp(td); do + oldbd*yp(td)];
  solp = [yp(td); do + oldbd*yp(td)];
  FFbd = (solp - solc)/dbd;
  FFP = -[FFa FFbr FFbd];
  newp = p - alpha*(FFP'*FFP + lam*eye(3))\FFP'*FF;
  error = norm(newp - p)/norm(p);
  lm;
  p = newp;
  olda = p(1); oldbr = p(2); oldbd = p(3);
  [t y] = sirdidc;
  %solc = [y(td); ro + oldbr*y(td); do + oldbd*y(td)];
  solc = [y(td); do + oldbd*y(td)];
  FF = dd - solc;
  norm(FF);
  if error < 0.00001
    break
  end
end
display('Parameters from Levenberg-Marquardt')
[olda oldbr oldbd]'
display('Norm of residual from Levenberg-Marquardt')
norm(FF)
error;
lm
%
figure(1)
```

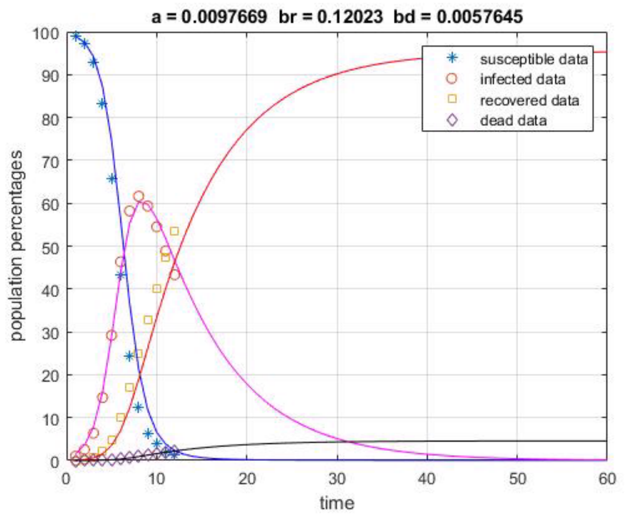


Figure 9.6.1: Parameter Identifcation and CI-equation

```
plot(td,Sd(1:nd),'*', td, Id(1:nd),'o',...
td,Rd(1:nd),'s', td, Dd(1:nd),'d')
hold on
[t y] = sirdidc;
C = y;
S = so*exp(-olda*C(1:60));
I = diff(C(1:61));
R = ro + oldbr*C(1:60);
D = do + oldbd*C(1:60);
time = t(1:60)+1;
plot(time,S,'b' ,time,I,'m' ,time,R,'r' ,time,D,'k')
title(['a = ' ,num2str(olda),' br = ' ,num2str(oldbr),...
' bd = ' ,num2str(oldbd)]);
ylabel('population percentages')
xlabel('time')
legend('susceptible data', 'infected data',...
'recovered data', 'dead data');
grid on
%svd(A(2:m,:))
%cond(A(2:m,:))
%svd(FFP)
%cond(FFP)
```

9.7 US COVID-19: An Aggregated Model

The SIRD model assumes the parameters are constant with respect to the particular population set and the time interval. This is not the case with respect to the entire US population. The COVID-19 virus varies with population location and time. This model is an attempt to "lump" or "aggregate" these local and time intervals into a more global (with respect to US) model. One approach would be to approximate the three SIRD parameters for all locations and time intervals, and then to form some sort of weighted average. The approach here is to use the cumulated infection and cumulated death data for the US and nonlinear parameter identification to approximate the three SIRD parameters.

In addition to the three parameters, two implicit parameters will include "death delay" and "effective population size." The death delay is related to the time gap between infection and time of death. The effective population is initially small and then increases as the population moves and spreads the virus. The following code attempts to quantify the three SIRD parameters and these two implicit parameters.

9.8 Matlab Code sird_paridcuscovid2.m

This code is a work-in-progress, dated 9-15-2020 by Robert E. White. The two outer loops have been set for fixed death delays and fixed data noise. The reader will find it interesting to experiment with these. There are six subsections on the general code, effective population, death delay, error and data sensitivity, calculation from 11-15-2020 with variable data, and variable effective contact.

9.8.1 Basic Code Date 9-15-2020

Cumulated infected and cumulated death data are given in lines 34-43 where the first day is 3-22-2020. The data is contained in the UScovid19.m file and may have been updated. Figure 9.5.1 has a daily graph of these, and note the cumulated graphs should be the integral or areas in this figure. The units are in lines 44-46. Units can be one person, 100 percent of the population, $100,000$ or adjusted to $12,000$ as in line 44. The full $100,000$ persons gives 327.2 million in the US. The adjusted $12,000$ gives the effective population of $3272 * 12000 = 39.264$ million. This was found by minimizing the norm in lines 200-203.

The recovery data in lines 58-63 is uncertain. One should experiment with the recovery ratio in line 60. Roughly, $ratio = 0.90$ means 90 percent of the infected recover and are no longer susceptible. The death data delay is set at $P = 10$ in lines 65-68. $P = 10$ was found by experimentation to minimize the norm in line 176. The noise data is set in lines 75-80.

Lines 71, 72-88 choose the initial data intervals. Line 71 chooses $nd = 20$ data points and $shift = 110$ starting day with death delay $P = 10$. The Levenberg-Marquardt algorithm is executed in lines 91-153. The initial estimate for the three parameters is in lines 91-98. The target in lines 101-102 has three vectors for C, R and D data. The condition number of the derivative matrix is of order 10^5, and therefore, variations in the numerical solution due to round-off or uncertain data can be important.

```
1 % This code uses least squares to identify three parameters
2 % in the SIRD model:
3 %   S_t = -a SI,
4 %   I_t = a SI - (br + bd) I,
5 %   R_t = br I and
6 %   D_t = bd I where
7 %         a = "contagious" (effective contact) parameter,
8 %         br = "recovery" parameter and
9 %         bd = "death" parameter.
10 %
11 % An equivalent system is for the cumulative sum or integral
12 % of the infection (solution of the CI-equation)
13 %      C_t = popmax - S(t0) - R(t0) - D(t0)
14 %            + S(t0)(1 - exp(-aC)) - br C - bd C, C(t0) = 0.
15 % This requires initial cumulative both infection and death data.
16 %
17 % function [t y] = sirdidc1
18 %   global olda oldbr oldbd so io ro do popmax t0 y0 tend
19 %   %y0 = io;
20 %   %to = 0;
21 %   tf = tend;
22 %   opts = odeset('RelTol',1e-3,'AbsTol',1e-6);
23 %   [t y] = ode45('ypsirdidc',[t0:1:tf],y0,opts);
24 %
25 % function ypsirdidcout = ypsirdidc(t,C)
26 %   global olda oldbr oldbd so io ro do popmax t0 y0 tend
27 %   ypsirdidcout = popmax - ro - do - so*exp(-olda*C)...
28 %                  - oldbr*C - oldbd*C;
29 %
30 clear;
31 clf(figure(1)); clf(figure(2));
32 clf(figure(3)); clf(figure(4));
33 global olda oldbr oldbd so io ro do popmax t0 y0 tend
34 %
35 % US COVID-19 Cumulative Data: infection (ID) and deaths (DD).
36 % Data is from coronavirus.jhu.edu.
37 % First day is 3-22-2020. Some numbers have been updated!
38 %
39 UScovid19;
40 % Populations are in units per 100K. There are 3272 units in US.
41 popmax = 3272;
```

```
ID = ID*popmax;    % gives cumulated infected caess
DD = DD*popmax;    % gives cumulated deaths
popunit = 12000;   % gives data per popunit ???
popmax*popunit     % effective population
DD = DD/popunit; ID = ID/popunit;
%
figure(1)
plot(diff(DD))
hold on
plot(diff(ID))
title('US COVID-19 Data');
ylabel('population')
xlabel('time')
legend('dead data','infected data','Location', 'best');
grid on
%
[m n] = size(ID); td = 0:n-1;
recovratio = (ID(n) - DD(n))/ID(n);
RD(1) = 0; ratio = 0.9                               %???
for i = 1:n-1                                        %???
  RD(i+1) = RD(i) + ratio*recovratio*(ID(i+1)-ID(i)); %???
end
%
Pmin = 10; Pmax = 10;   % delay in death data          ???
countP = 1;
%
for P = Pmin:Pmax     % computations with variable delays
%
for testit = 1:1      % computations with noise in data
nd = 20; shift = 110; % number of data starting at shift
Cd = ID((1:nd)+shift)' - ID(shift+1)';
Dd = DD((1:nd)+shift + P)' ;     % use delayed death data
Rd = RD((1:nd)+shift)';
rvec = rand(nd,1);               % used in mean and std
%Cd = Cd + Cd.*(2*rvec - 1)/50;
rvec = rand(nd,1);               % used in mean and std
%Rd = Rd + Rd.*(2*rvec - 1)/100;
rvec = rand(nd,1);               % used in mean and std
%Dd = Dd + Dd.*(2*rvec - 1)/100;
td = (1:nd) + shift; Td = 1:nd;
t0 = td(1); tend = td(nd);
CP = ID(shift+1);
y0 = 0;
io = sum((ID((-2:4)+shift) - ID((-3:3)+shift)))/7;
ro = Rd(1);
```

```
do = Dd(1);
so = popmax - io - ro - do;
%data = [td' Cd Dd]
%
% Parameter ID initial estimate
%
  oldbr = Cd\(Rd - Rd(1));
  oldbd = Cd\(Dd - Dd(1));
  Sd = popmax - (diff(Cd)) - Rd(1:(nd-1)) - Dd(1:(nd-1));
  olda = Cd(1:(nd-1))\(-log(Sd/Sd(1)));
  display('Initial Parameters a,br,bd ')
  x = [olda oldbr oldbd]'
% Target and Start for Levenberg-Marquardt Algorithm
[t yp] = sirdidc1; C = yp ;        % SIRD (CI-equation)
dd = [Cd(1:nd); Rd(1:nd);...
       Dd(1:nd)];                  % target
solc = [C(Td) ; ro + oldbr*(C(Td)); ...
       do + oldbd*(C(Td))];        % initial computed
%dd = [Cd(1:nd); Dd(1:nd)];        % target
%solc = [C(Td); do + oldbd*C(Td)]; % initial computed
FF = dd - solc;
%display('Norm of initial residual')
%norm(FF)
%
% Parameter ID from Levenberg-Marquardt Algorithm
%
lam = 1.0; alpha = 4.0; p = x;
for lm = 1:2000
  da = 0.0000001;
  olda = p(1) + da; oldbr = p(2); oldbd = p(3);
  [t yp] = sirdidc1;
  solp = [yp(Td); ro + oldbr*(yp(Td)); ...
  do + oldbd*(yp(Td))];
  %solp = [yp(Td); do + oldbd*yp(Td)];
  FFa = (solp - solc)/da;
  dbr = 0.00001;
  oldbr = p(2) + dbr; olda = p(1); oldbd = p(3);
  [t yp] = sirdidc1;
  solp = [yp(Td) ; ro + oldbr*(yp(Td)); ...
  do + oldbd*(yp(Td))];
  %solp = [yp(Td); do + oldbd*yp(Td)];
  FFbr = (solp - solc)/dbr;
  dbd = 0.00001;
  oldbd = p(3) + dbd; olda = p(1); oldbr = p(2);
  [t yp] = sirdidc1;
  solp = [yp(Td) ; ro + oldbr*(yp(Td)); ...
  do + oldbd*(yp(Td))];
  %solp = [yp(Td); do + oldbd*yp(Td)];
  FFbd = (solp - solc)/dbd;
  FFP = -[FFa FFbr FFbd ];
  newp = p - alpha*(FFP'*FFP + lam*eye(3))\(FFP'*FF);
  error = norm(newp - p)/norm(p);
  lm;
  p = newp;
  olda = p(1); oldbr = p(2); oldbd = p(3);
  [t y] = sirdidc1;
  solc = [y(Td) ; ro + oldbr*(y(Td)); ...
  do + oldbd*(y(Td))];
  %solc = [y(Td); do + oldbd*y(Td)];
  FF = dd - solc;
  %norm(FF)
  %pause
  err(lm,1:3) = p';
  if error < 0.000001
    break
  end
end
[olda oldbr oldbd]';
norm(FF);
error;
lm ;
%
% Use New Parameters to Solve SIRD Model of US COVID-19
%
tend = 230; olda = olda*1.0
lm
Dd = DD((1:nd)+shift + P)' ;
Rd = RD((1:nd)+shift)';
io = sum((ID((-2:4)+shift) - ID((-3:3)+shift)))/7;
ro = Rd(1);
do = Dd(1);
so = popmax - io - ro - do;
[t y] = sirdidc1; C = y + 0;
S = so*exp(-olda*(C(1:(tend-1-shift))));
I = diff((C(1:(tend-shift)) - 0));
R = ro + oldbr*(C(1:(tend-1-shift)));
D = do + oldbd*(C(1:(tend-1-shift)));
maxD(testit) = max(D(1:60));
end        % end loop for mean and std computations
res(countP) = norm(Dd(Td) - (do + oldbd*C(Td)));
```

```
countP = countP + 1;
end        % end loop for delay
%
% Output of computations
%
display( 'Data from shift to shift+(number of data points)')
[shift shift+nd]
display('Mean maxD ')
mean(maxD)
display('Standard deviation maxD ')
std(maxD)
display('R0 at ndays past shift' )
ndays = 1
S(ndays)*olda/(oldbr + oldbd)
format long
display('Parameters a,br,bd ')
[olda oldbr oldbd ]'
display('Singular values of FFP')
svd(FFP)
cond(FFP)
format short
display('Norm of target - computed in LM')
norm(FF)
display('Norm of daily I data - daily I computed')
Idata = diff(ID((shift+1):(shift+1+2*nd))'*popunit);
Icomp = I((1:(2*nd)))*popunit;
norm(Idata - Icomp)
[resmin countP] = min(res);
PP = Pmin + countP - 1
display('Norm death data - computed death')
resmin
res'
%
figure(2)
plot(td,(Cd(1:nd) + CP)*popunit,'o', td,Dd(1:nd)*popunit,'d')
hold on
time = (shift+1):tend-1;
plot(time,S*popunit,'b' ,time,I*popunit,'m' ,...
time,R*popunit,'r' ,time,D*popunit,'k')
%plot(time,S+I+R+D)
plot(time, (C(1:(tend-1-shift)) + CP)*popunit )
title(['a = ' ,num2str(olda),' br = ' ,num2str(oldbr),...
       ' bd = ' ,num2str(oldbd)]);
ylabel('population')
xlabel('time')
```

```
222 legend('cum. infected data', 'cum. death data',...
223          'susceptible', 'infected','recovered',...
224          'dead ', 'computed C', 'Location', 'northeast');
225 grid on
226 %
227 figure(3)
228 plot( time,I*popunit,'m', time,D*popunit,'k')
229 hold on
230 %resD = DD((1:nd)+shift+PP)'- D(1:nd);
231 nn = n - shift - P ;
232 plot((1:nn) + shift , DD((1:nn) + shift + P )*popunit)
233 title(['olda = ' ,num2str(olda),' oldbr = ' ,num2str(oldbr),...
234        ' oldbd = ' ,num2str(oldbd)]);
235 ylabel('population')
236 xlabel('time')
237 legend( 'computed infected', 'computed cum. dead',...
238         'cum. dead data', 'Location', 'best');
239 grid on
240 %
241 figure(4)
242 plot(time,I*popunit)
243 hold on
244 plot(1:n-1,diff(ID)*popunit)
245 title(['olda = ' ,num2str(olda),' oldbr = ' ,num2str(oldbr),...
246        ' oldbd = ' ,num2str(oldbd)]);
247 ylabel('population')
248 xlabel('time')
249 legend('computed infected', 'infected data',...
250 'Location', 'northeast');
251 grid on
252 %
253 % Use sirdid.m which solves the SIRD without the CI-equation.
254 %
255 [tt yy] = sirdid; % uses shift, t0, tend and above parameters
256 figure(5);
257 subplot(2,1,1)
258 plot(tt,yy(:,1)*popunit, tt,yy(:,3)*popunit)
259 title('susceptible and recovered')
260 grid on
261 subplot(2,1,2)
262 plot(tt,yy(:,2)*popunit, tt,yy(:,4)*popunit)
263 title('dead and infected')
264 grid on
```

The numerical output in lines 180-209 follows from the updated solution in lines 159-173. Figures 9.8.1, 9.8.2 and 9.8.3 follow from lines 210-225, 227-

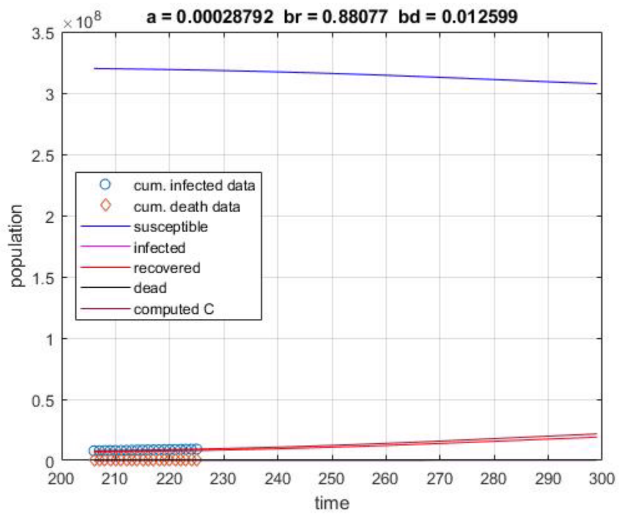


Figure 9.8.1: Aggregated US COVID-19

239 and 241-251. The figure generated by lines 253-264 check the CI-equation calculations. In Figure 9.8.1 the top curve is the susceptible population, the middle curve is the cumulated infected and the third curve is the recovered population. Figure 9.8.2 has the death data and computed cumulated dead at the top, and the bottom curve is the computed daily infected. Figure 9.8.3 compares the daily infected data with the daily computed infected.

9.8.2 Effective Population

Figure 9.8.4 experiments with different effective population sizes by adjusting *popunit* in lines 44 and 45. The three computed infected populations to the left side correspond to *popunit* $= 1,000$, $5,000$ and $10,000$, and it uses the data from day 4 to 24. The optimal choice of *popunit* can be found by minimizing the norm in lines 201-203. As the US population moves about the country, the effective population increases. The three computed infected populations to the right side correspond to *popunit* $= 10,000$, $50,000$ and $100,000$, and it uses the data from day 110 to 130. The near optimal choice of *popunit* is $16,000$ as is illustrated in Figure 9.8.3.

9.8.3 Death Delay

The next two figures indicate the death delays associated with the two time intervals near the relative maximums. Figure 9.8.5 is for the time interval from day 4 to 24 and uses the effective population equal to 6.5 million. The

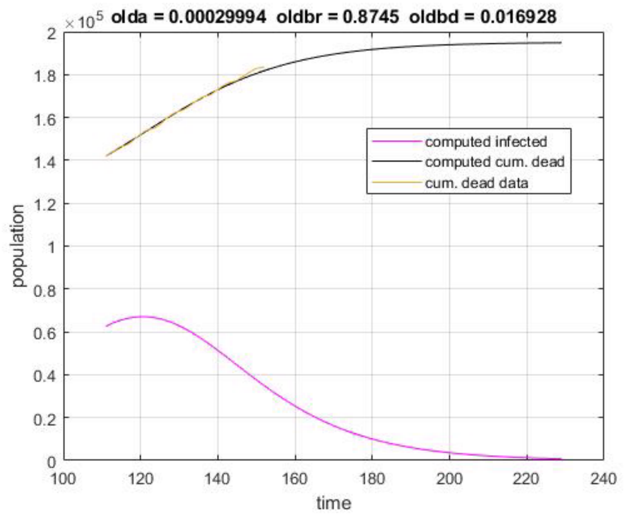


Figure 9.8.2: Aggregated US COVID-19

norms are computed in lines 176 and 207, and $P = 7$ gives the smallest norm. Figure 9.8.6 is for the time interval from day 110 to 130 and uses the effective population equal to 52.4 million. The norms are computed in lines 176 and 207, and $P = 10$ gives the smallest norm. Both these calculations use 20 data points to determine the three SIRD parameters, and any predictions for times larger than 20 additional days is not realistic. The primary reason for this is that the parameters do vary with time because the implementation of the protocols for the control of the virus do vary.

9.8.4 Error Sensitivity

Another concern is the choice of errors used to determine the five parameters. The following errors were used:

$$\begin{aligned}
E &= [C - C_d,\ R - R_d,\ D - D_d] \text{ to find } a, b_r \text{ and } b_d, \\
E_I &= [I - I_d] = [\frac{dC}{dt} - diff(C_d)] \text{ to find } effective\ population \text{ and} \\
E_D &= [D - D_d] \text{ to find } death\ delay.
\end{aligned}$$

In the calculations the three SIRD parameters were always computed for each possible choice of the implicit parameters. Also the norm used was always the 2-norm. The two errors $C - C_d$ and $\frac{dC}{dt} - diff(C_d)$ suggest convergence in the H_1 Sobolev norm.

Any small variation in the effective contact parameter, a, can give large variations in the solution. Variation of a by one percent (see line 161) gives

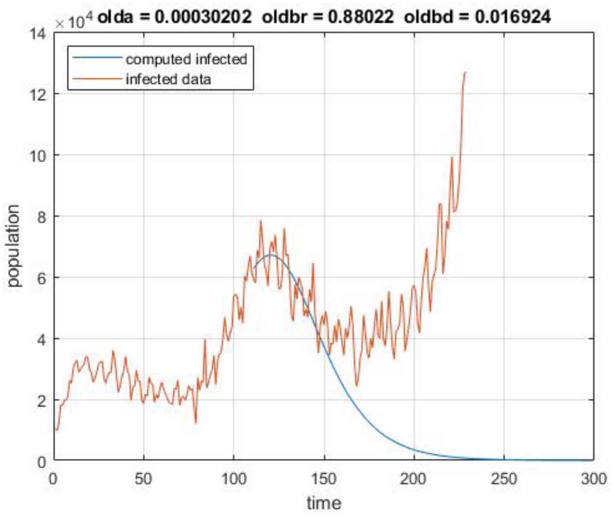


Figure 9.8.3: Aggregated US COVID-19

these changes:

$$\begin{gathered}D(171) \text{ goes from } 194,100 \text{ to } 203,800 \text{ and}\\ \text{the relative maximum of the daily infected goes}\\ \text{from } (122,\quad 67,100) \text{ to } (129,\quad 76,200).\end{gathered}$$

The condition number of the matrix used in the Levenberg-Marquardt calculations is of order 39,000. If one varies the data by two percent by using lines 75-80 and does 100 executions of the loop starting at line 70, then the mean of $D(171)$ is $194,800$ and the standard deviation is $6,600$.

9.8.5 Calculations With Variable Data

The data in Figure 9.8.7 varies from March 22, 2020 (day 1) to November 15, 2020 (day (237). The calculations use data from 20 day intervals in April, July and October. The effective populations increased from 3272*2000, to 3272*10000 and then to full population 3272*100000. This reflects the spread of the virus which is a result of human movement about the US. Because the model parameters vary with time, the computed infected population eventually do not approximate the daily infected data. Roughly, 20 past data points maybe able to predict 20 future data points. These calculations are sensitive to small variations in the effective contact parameter in line 161.

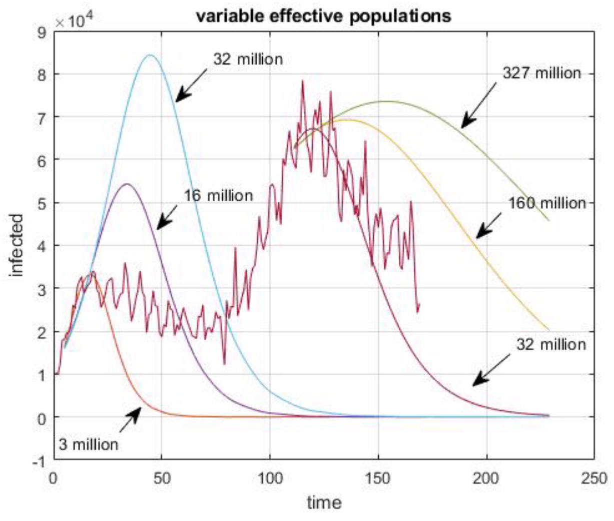


Figure 9.8.4: Effective Population Size

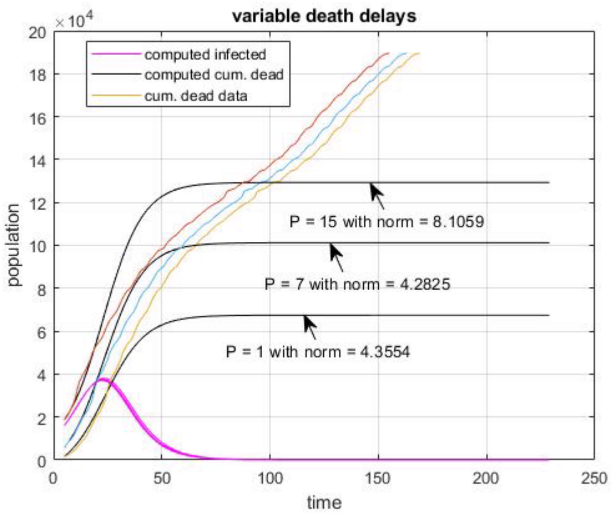


Figure 9.8.5: Deaths from Day 4 to 24 Data

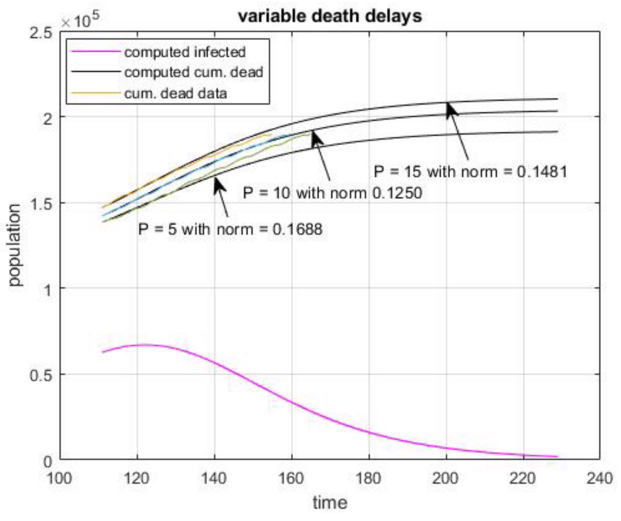


Figure 9.8.6: Deaths from Day 110 to 130 Data

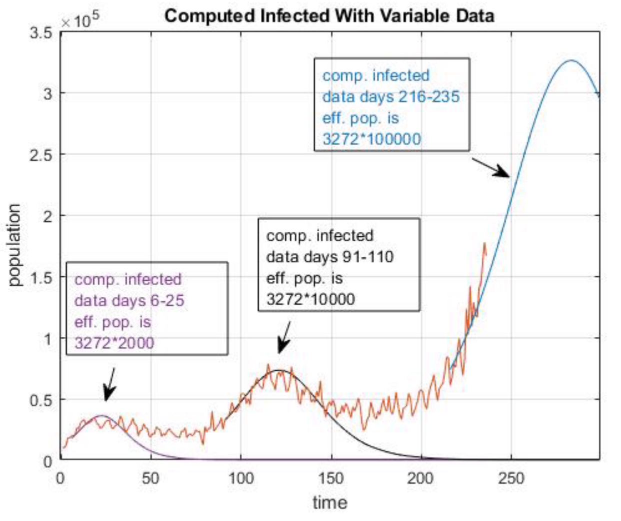


Figure 9.8.7: Computed Infected US COVID-19

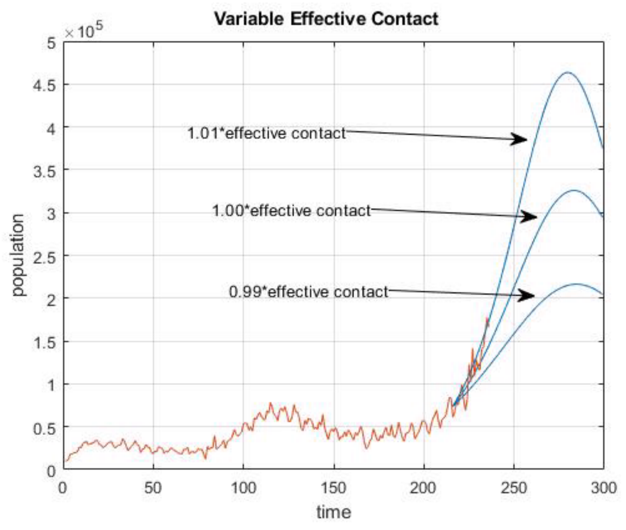


Figure 9.8.8: Control of the Infected Population

9.8.6 Use of Protocols to Adjust Effective Contact

In the SIRD model the differential equation for the infected is

$$\frac{dI}{dt} = I(aS - b_r - b_d) \text{ and } I \text{ is positive.}$$

So, if the second factor is positive (negative), then I will increase (decrease). This may also be written as

$$\frac{aS(t)}{b_r + b_d} > 1 \text{ (or } < 1) \text{ for } I \text{ to increase (or decrease).}$$

The effective contact parameter, a, controls the slope of the infective curve. The susceptible population, $S(t)$, is large and the effective contact is much smaller. This ratio is computed in line 190, and it will vary with time. At the local maximum it will be 1.0.

Use of protocols can be used to reduce this ratio. Small variations in the effective contact parameter can have larger variations in the computed infected population. This is illustrated in Figure 9.8.8 where the variation is plus or minus one percent as given in line 161. Note, the maximum infected computation vary from $216K, 326K$ to $643K$ as the effective contact varies from $0.99a, 1.00a$ to $1.01a$ where $a = 0.00028725$, respectively.

Bibliography

[1] James Balama, hypatia.gz, www.math.uri.edu/~jbaglama/#Software.

[2] M. W. Berry and M. Browne, *Understanding Search Engines: Mathematical Modeling and Text retrieval,* 2nd ed, SIAM, Philadelphia, 2005.

[3] James Chen, fourier.enghmc.edu/e161/lectures/svdcompression.html

[4] Nicolas Gillis, "Learning with Nonnegative Matrix Factorizations," *SIAM News*, vol. 25, issue 5, June, 2019.

[5] Gene H. Golub and Charles E. Van Loan, *Matrix Computations*, 2nd ed., John Hopkins University Press,1989.

[6] Serge Lang, *Linear Algebra*, Addison-Wesley, 1966.

[7] Carl D. Meyer, *Matrix Analysis and Applied Linear Algebra,* SIAM, 2000.

[8] Mattes Mollenhauser, Ingmar Schuster, Stefan Kluss and Christoff Schűtte, "Singular Value Decomposition of Operators on Reproducing Kernel Hilbert Space," arxiv.org/pdf/1807.09331vf.pdf

[9] J. M. Ortega and W. C Rheinboldt, *Iterative Solutions of Nonlinear Equations in Several Variables,* Academic Press, 1970.

[10] Lothar Reichel, www.math.kent.edu/~reichel/course/intr.num.comp.2/spring/12/lecture9/lecture9.pdf

[11] Gauthier Sallet, *Mathematical Epidemiology,* www.iecl.univ-lorraine.fr/~Gauthier.Sallet/Lecture-Notes-Pretoria-2018.pdf

[12] Gilbert Strang, *Introduction to Linear Algebra, Fourth Edition,* SIAM, 2009.

[13] Curtis R. Vogul, *Computational Methods for Inverse Problems,* SIAM, Philadelphia, 2002.

[14] Robert E. White, *Elements of Matrix Modeling and Computations with MATLAB,* Chapman Hall/CRC, 2007.
(https://white.math.ncsu.edu/toc072006.pdf)

[15] Robert E. White, "Nonlinear least squares algorithm for identification of hazards," *Congent Mathematics*, 2(1), 15 December 2015.
(https://www.cogentoa.com/article/10.1080/23311835.2015.1118219)