The Pennsylvania State University

The Graduate School

TRUNCATED URV DECOMPOSITIONS (TURVDS) FOR SUBSPACE TRACKING IN "SLOW-MOVING" VIDEO SEQUENCES

A Thesis in

Computer Science and Engineering

by

Manjesh Bhargav Malavalli

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

August 2008

The thesis of Manjesh Bhargav Malavalli was reviewed and approved* by the following:

Jesse Barlow

Professor, Department of Computer Science and Engineering

Thesis Adviser

Padma Raghavan

Professor, Department of Computer Science and Engineering

Raj Acharya

Professor and Head, Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

A small subset of successive frames of a "slow-moving" video sequence constitutes a subspace whose dimensionality is much less than the frame size. Any frame drawn from the video sequence can be expressed as a linear combination of the frames close to it in time. The primary objective of tracking the subspace of this subset is to reduce the dimensionality required to store similar frames by projecting them onto the subspace. Also, the subspace is updated only for frames that are sufficiently different from the ones in the subset. The traditional subspace tracking algorithm uses the *Singular Value Decomposition* (SVD) to maintain the subspace information. This work proposes a modification to the tracking algorithm where the SVD is approximated by a *Truncated URV Decomposition* (TURVD). A TURVD can be updated and downdated more efficiently than the corresponding SVD. It is also more efficient than calculating the SVD from scratch. This work also proposes algorithms for updating and downdating a TURVD, and provides a detailed analysis of the accuracy and efficiency of these algorithms when compared to calculating the SVD from scratch.

Table of Contents

List of Figures

Acknowledgments

I would like to take this opportunity to express my deep gratitude to my advisor, Dr. Jesse Barlow, whose support and guidance has been of immense value to me throughout my Masters Program here at Penn State. I would also like to thank my family (Pappa, Amma and Akhilesh, Anna and Ajji Amma) for their unwavering support during both good times and bad. Finally, I would like to thank Kumar Uncle, Seetharam Uncle, Vijaya Aunty and Usha Aunty for looking after me as their own son. I will be forever indebted to you, and try my very best to live up to your expectations.

Dedication

Yogini Amma, whose fond memories and sound advice will forever be with me.

Chapter 1

Introduction

Applications that stream video over a network have become commonplace due to the increase in available bandwidth over recent years. The central challenge involved in developing such applications is the definition of an encoding/decoding scheme that deals with the available bandwidth of a given communication channel. Video compression standards such as MPEG, CCIT H.261 exploit the temporal redundancy of several closely placed video frames in order to reduce the size of the video during transmission, thereby utilizing considerable less bandwidth than the case where the entire video is transmitted as it is. They achieve this by maintaining a set of *basis images* that span a subspace of the frame space. A reduction in the number of dimensions required for representing a given video frame is achieved by projecting it onto this lower dimensional subspace.

1.1 Subspace Tracking

In the case of *"slow-moving"* video sequences, we can further reduce the transmission bandwidth by projecting each non-basis frame f onto a basis set that is constantly modified to accurately represent the subspace of frames in f's neighborhood. Such a variable basis set approach can

effect a greater reduction in bandwidth required for video transmission than the fixed basis image set because theoretically, for *"slow-moving"* video sequences, the basis set should also vary slowly over time. This is the basic premise of *subspace tracking*. Wu et. al. [1] propose a technique to track the subspaces of slow-moving video sequences of facial expressions, using the *singular value decomposition* (SVD), which is the basis for the subspace tracking algorithm used in this work. The basic concept of subspace tracking and its algorithm are explained in detail in the following section.

1.2 Algorithm for Tracking Subspaces

Let $X = (x_1, x_2, ..., x_n), x_i \in \Re^k, X \in \Re^{k \times n}$ be the matrix whose columns ($each\ x_i$) are the mean-separated vectorized forms of a subset of n frames of a video sequence that are pretty close to one another. Then, any frame represented by the vector t which is close to the frames in X can be projected onto the column space of X as follows,

$$Xw \quad = \quad t$$

where $w \in \Re^n$. Here, Xw is the subspace representation of t in the column space of X, that is $R(X)$. The dimensionality of w can be reduced substantially by maintaining the orthonormal basis vectors of the fundamental column subspace of X, and discarding those basis vectors that represent the directions of small variances. In order to achieve this, the SVD of X is defined as follows,

$$X \quad = \quad U\Sigma V^T$$

where U and V are the orthogonal eigenvector matrices of XX^T and X^TX respectively. The columns of U are the orthonormal basis vectors of the column space of X. Hence, the dimension

of w can be reduced by the following expression,

$$w = {U_r}^T t \tag{1.1}$$

where U_r is the matrix obtained by retaining only those columns of U corresponding to the r largest singular values ($r << n$).

The following is the algorithm for tracking the subspace of a slow-moving video sequence.

Algorithm 1 Subspace Tracking for Slow Moving Video

```
Input F, m, r, s, d
Initialize the basis image matrix X and its SVD, U, Σ, V
Initialize μ, the mean of the columns of X
for i = s to NumFrames(F), increment by s do
   Project the s − 1 previous frames onto R(U_r), U_r being the first r columns of U.
   if ‖F_i − U_r U_r^T F_i‖_2 ≥ d then
      X ← (X F_i)
      Update μ
      X_m ← mean-subtracted columns of X
      (U, Σ, V) ← SVD(X_m)
      if NumCols(U) > m then
         X ← X(:, 2 : end)
         Update μ
         X_m ← mean-subtracted columns of X
         (U, Σ, V) ← SVD(X_m)
      end if
   end if
end for
```

Here, F is a matrix whose columns represent the vectorized frames of a particular video sequence. At the start of the algorithm, the SVD of the basis image matrix X is initialized by using the first frame as a basis image. The video sequence is sampled every s frames and these sample frames are considered for updating X and its SVD. Any such sampled frame whose Euclidean distance from its projection onto $R(U_r)$ crosses a certain threshold d, is included in the basis image matrix X, and the SVD of the updated X is determined from scratch. The unsampled frames are just projected onto $R(U_r)$, and these projection vectors are used to reconstruct the frames during the decoding stage. The concept of *tracking* is incorporated by setting an bound

(m) on the maximum number of basis images that can be stored at any instant of time. This algorithm assumes that this bound is a prespecified constant, although a case can be made for determining the bound dynamically, based on the extent of temporal redundancy available across frames. The presence of this bound means that, whenever the number of basis images reaches m, addition of any new basis image to the matrix X must be preceded by the removal of the basis image in X that was least recently added (first column of X). The reconstruction of unsampled frames is achieved as follows. If p is the projection of a mean-subtracted unsampled frame $F_i - \mu$ onto $R(U_r)$ ($p = {U_r}^T(F_i - \mu)$), then the corresponding reconstructed frame $\hat{F}_i$ is given by $\hat{F}_i = U_r p + \mu$.

1.3 Proposed Modification

The implementation of the subspace tracking algorithm typically involves the computation of the SVD of the matrix X from scratch each time it is modified, which is computationally very expensive, of the order $O(mn^2)$, (assuming $X \in \Re^{m \times n}$). The solution proposed to this problem in this work is to use the *Truncated URV Decomposition* (TURVD) to approximate the SVD of X. The advantage of TURVDs is that they can be updated in $O(mn)$ time, which is faster than computing the SVD from scratch. Experimental results show that the TURVD can indeed provide a reasonably accurate approximation of the SVD, and the time taken to update it is less than the time taken to compute the SVD from scratch.

Chapter 2

Truncated URV Decompositions

2.1 Introduction

The *Truncated URV Decomposition* (TURVD) of a matrix $X \in \Re^{m \times n}, m > n$, is of the form,

$$X = U_1 R {V_1}^T + E \tag{2.1}$$

where $U_1 \in \Re^{m \times k}$, $R \in \Re^{k \times k}$ is non-singular and upper triangular, $V_1 \in \Re^{n \times k}$(k is the rank of X), and $E \in \Re^{m \times n}$ is the *error matrix*. U_1 and V_1 are *left orthogonal*, that is, ${U_1}^T U_1 = {V_1}^T V_1 = I_k$. TURVD is different from the regular URV decomposition [2] in the sense that the error matrix is not stored, but its Frobenius norm approximate is retained and regularly updated. E and R satisfy the following conditions :

- $EV_1 = 0$,
- $\|R^{-1}\|_2 \leq \epsilon^{-1}$ and $\|E\|_2 \leq \epsilon$, for some tolerance ϵ.

The second constraint can be enforced only through a computationally expensive SVD of X. Hence, the efficiency of the updating and downdating algorithms can be increased by changing

the constraint from a bounds problem to a minimization problem. That is, the TURVD of X should have $\|R^{-1}\|_2$ minimized subject to the constraint $\|E\|_F \leq \epsilon_F$, for some *Frobenius norm tolerance* ϵ_F. This is similar to the constraint on TULVDs [3].

2.2 TURVD Operations

Algorithms for the following TURVD operations are provided in the coming sections :

2.2.1 Update a Column

Given the TURVD of X as specified by (2.1), and a column $x_0 \in \Re^m$, the TURVD of $(X\ x_0)$ is given by :

$$(X, x_0) \quad = \quad \bar{U}_1 \bar{R} \bar{V}_1^T + \bar{E} \tag{2.2}$$

where $\bar{U}_1 \in \Re^{m \times \bar{k}}$, $\bar{R} \in \Re^{\bar{k} \times \bar{k}}$, and $\bar{V}_1 \in \Re^{(n+1) \times \bar{k}}$, $\bar{E} \in \Re^{m \times (n+1)}$, and $\bar{k} \in \{k,\ k+1\}$.

2.2.2 Downdate a Column

Given the TURVD of X as specified by (2.1), and if X can be written as $X = (x_0, X_d)$, then the TURVD of X_d is given by :

$$X_d \quad = \quad \hat{U}_1 \hat{R} \hat{V}_1^T + \hat{E} \tag{2.3}$$

where $\hat{U}_1 \in \Re^{m \times \bar{k}}$, $\hat{R} \in \Re^{\bar{k} \times \bar{k}}$, and $\hat{V}_1 \in \Re^{(n-1) \times \bar{k}}$, $\hat{E} \in \Re^{m \times (n-1)}$, and $\bar{k} \in \{k,\ k-1\}$.

2.2.3 Rank One Update

Given the TURVD of X as specified by (2.1), and if $f \in \Re^m$, and $g \in \Re^n$, then the TURVD of $X + fg^T$ is given by :

$$X + fg^T \quad = \quad \tilde{U}_1 \tilde{R} \tilde{V}_1^T + \tilde{E} \tag{2.4}$$

where $\tilde{U}_1 \in \Re^{m \times \bar{k}}$, $\tilde{R} \in \Re^{\bar{k} \times \bar{k}}$, and $\tilde{V}_1 \in \Re^{n \times \bar{k}}$, $\tilde{E} \in \Re^{m \times n}$, and $\bar{k} \in \{k+1,\ k,\ k-1\}$.

2.3 Matrix Computation Tools

Before the algorithms are described, it is prudent to be aware of the following matrix computation tools which are used in their implementation.

2.3.1 $[\sigma_s, y_s, z_s] =$ ***smallest_sing***(M)

Given a matrix M, the above routine returns its smallest singular value σ_s along with unit vectors y_s and z_s such that $Mz_s = \sigma_s y_s$.

2.3.2 $[\hat{q}, s, \alpha] =$ ***cgs_orth***(Q, x)

Given a matrix $Q \in \Re^{m \times n}, m > n$, with orthonormal columns and a vector $x \in \Re^m$, returns $\hat{q}$, s, and α such that $Q^T \hat{q} = 0, \|\hat{q}\|_2{}^2 = 1$ and $x = Qs + \alpha \hat{q}$. The algorithm for the orthogonalization routine is discussed in [4].

2.3.3 $[\bar{U}_1, \bar{R}, \bar{V}_1] =$ ***TURV_deflate***(U_1, R, V_1, w)

Given the TURVD of X and any vector $w \in \Re^k$, the above function returns a modified TURVD where $\bar{U}_1 = U_1 Q_1, \bar{R} = {Q_2}^T R Q_1$ remains upper triangular, and $\bar{V}_1 = V_1 Q_2$, Q_1 and Q_2 being orthogonal matrices such that ${Q_1}^T w = e_k$. The algorithm for deflation is described in [5].

2.3.4 $[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] =$ ***TURV_refine***$(U_1, R, V_1, \|E\|_F)$

Given the TURVD of X as specified by (2.1), the above function returns a modified TURVD

$$X = \bar{U}_1 \bar{R} \bar{V}_1^T + \bar{E} \tag{2.5}$$

where $\bar{U}_1 \in \Re^{m \times \bar{k}}$, $\bar{R} \in \Re^{\bar{k} \times \bar{k}}$, and $\bar{V}_1 \in \Re^{n \times \bar{k}}$, $\bar{E} \in \Re^{m \times n}$, and $\bar{k} \in \{k,\ k-1\}$. If $\bar{k} = k$, then $\|\bar{E}\|_F \leq \|E\|_F$. See Barlow and Erbay [3] for details on the algorithm.

2.4 Algorithm for TURVD Column Update

Given the TURVD of a matrix X of the form given by (2.1), the following routine determines the TURVD of the form given by (2.2).

procedure $[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] =$ ***TURVD_update_col***$(U_1, R, V_1, \|E\|_F, r, x_0)$

Step 1:

Let $[\hat{u}, s, \alpha] = cgs\_orth(U_1, x_0)$

Then,

$$(X, x_0) = (U_1, \hat{u}) \begin{pmatrix} R & s \\ 0 & \alpha \end{pmatrix} \begin{pmatrix} {V_1}^T & 0 \\ 0 & 1 \end{pmatrix} + (E\ 0)$$

Step 2:

Let $[\sigma_s, y_s, z_s] = smallest\_sing(R')$, where $R' = \begin{pmatrix} R & s \\ 0 & \alpha \end{pmatrix}$

if $(\sqrt{{\sigma_s}^2 + {\|E\|_F}^2} \leq \epsilon_F$ *OR* $numcols(U_1) = r)$ *goto* ***Step 3*** *else goto* ***Step 4***.

Step 3:

$[\tilde{U}_1, \tilde{R}, \tilde{V}_1] = TURV\_deflate((U_1, \hat{u}), R', {V_1}', z_s)$, *where* ${V_1}' = \begin{pmatrix} V_1 & 0 \\ 0 & 1 \end{pmatrix}$

Hence, $\tilde{U}_1 = (\bar{U}_1, \bar{u})$, $\tilde{V}_1 = (\bar{V}_1, \bar{v})$, $\tilde{R} = \begin{pmatrix} \bar{R} & 0 \\ 0 & \sigma_s \end{pmatrix}$, and $\|\bar{E}\|_F = \sqrt{{\sigma_s}^2 + {\|E\|_F}^2}$

Step 4:

$[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] = TURV\_refine((U_1, \hat{u}), R', {V_1}', \|E\|_F)$

2.5 Algorithm for TURVD Column Downdate

Given the TURVD of a matrix X of the form given by (2.1), the following routine determines the TURVD of the form given by (2.3).

procedure $[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] =$ ***TURVD_downdate_col***$(X, U_1, R, V_1, \|E\|_F)$

Step 1:

Let $[v_0, f, \beta] = cgs\_orth(V_1, e_1)$ and $[u_0, g, \alpha] = cgs\_orth(U_1, \beta X v_0)$

Then,

$$\begin{aligned} X &= (x_0, X_d) \\ &= \tilde{U}_1 \tilde{R} \tilde{V}_1^T + \tilde{E} \end{aligned}$$

where $\tilde{U}_1 = (U_1, u_0)$, $\tilde{V}_1 = (V_1, v_0)$, $\tilde{R} = \left[\begin{pmatrix} R & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} g \\ \alpha \end{pmatrix} (f^T \ \beta) \right]$, $\tilde{E}e_1 = 0$

and $\|\tilde{E}\|_F = \sqrt{{\|E\|_F}^2 - {\|g\|_2}^2 - \alpha^2}$

Step 2:

$[\tilde{U}_1, \tilde{R}, \tilde{V}_1] = TURV\_deflate(\tilde{U}_1, \tilde{R}, \tilde{V}_1, w)$, *where* $w = \begin{pmatrix} f \\ \beta \end{pmatrix}$

Hence, $\tilde{U}_1 = (\hat{U}_1, \tilde{u})$, $\tilde{R} = \begin{pmatrix} \hat{R} & \tilde{g} \\ 0 & \tilde{\alpha} \end{pmatrix}$, and $\tilde{V}_1 = \begin{pmatrix} 0 & 1 \\ \hat{V}_1 & 0 \end{pmatrix}$

Therefore,

$$X_d = \hat{U}_1 \hat{R} \hat{V}_1^T + \hat{E}$$

where $\hat{E} = \tilde{E}(:, 2:n)$ and $\|\hat{E}\|_F = \|\tilde{E}\|_F$

Step 3:

Let $[\sigma_s, y_s, z_s] =$ *smallest_sing*$(\hat{R})$

if $(\sqrt{{\sigma_s}^2 + {\|\hat{E}\|_F}^2} \leq \epsilon_F)$ *goto* ***Step 4*** *else goto* ***Step 5***.

Step 4:

$[\hat{U}_1, \hat{R}, \hat{V}_1] =$ *TURV_deflate*$(\hat{U}_1, \hat{R}, \hat{V}_1, z_s)$

Hence, $\hat{U}_1 = (\bar{U}_1, \bar{u})$, $\hat{V}_1 = (\bar{V}_1, \bar{v})$, $\hat{R} = \begin{pmatrix} \bar{R} & 0 \\ 0 & \sigma_s \end{pmatrix}$, and $\|\bar{E}\|_F = \sqrt{{\sigma_s}^2 + {\|\hat{E}\|_F}^2}$

Step 5:

$[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] =$ *TURV_refine*$(\hat{U}_1, \hat{R}, \hat{V}_1, \|\hat{E}\|_F)$

2.6 Algorithm for TURVD Rank One Update

Given the TURVD of a matrix X of the form given by (2.1), the following routine determines the TURVD of the form given by (2.4).

procedure $[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] =$ ***TURVD_rank_one_update***$(U_1, R, V_1, \|E\|_F, r, f, g)$

Step 1:

Let $[\hat{u}, s, \alpha] = cgs\_orth(U_1, f)$, and $[\hat{v}, t, \beta] = cgs\_orth(V_1, g)$

Then,

$$\begin{aligned} \bar{X} &= X + fg^T \\ &= (U_1, \hat{u}) \left[\begin{pmatrix} R & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} s \\ \alpha \end{pmatrix} (t^T \ \beta) \right] \begin{pmatrix} {V_1}^T \\ \hat{v}^T \end{pmatrix} + E \end{aligned}$$

Step 2:

Find orthogonal matrices Q_1 and Q_2 such that,

- ${Q_1}^T \begin{pmatrix} t \\ \beta \end{pmatrix} = \delta e_{k+1}$, *where* $\delta = \left\| \begin{pmatrix} t \\ \beta \end{pmatrix} \right\|$
- $\tilde{R} = {Q_2}^T \left[\begin{pmatrix} R & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} s \\ \alpha \end{pmatrix} (t^T, \beta) \right] Q_1$ remains upper triangular.
- $\tilde{U_1} = (U_1, \hat{u})Q_1$ *and* $\tilde{V_1} = (V_1, \hat{v})Q_2$

Step 3:

Let $[\sigma_s, y_s, z_s] = smallest\_sing(\tilde{R})$

if $(\sqrt{{\sigma_s}^2 + {\|E\|_F}^2} \leq \epsilon_F$ *OR* $numcols(U_1) = r)$ *goto* ***Step 4*** *else goto* ***Step 6***.

Step 4:

$[\tilde{U}_1, \tilde{R}, \tilde{V}_1] = TURV\_deflate(\tilde{U}_1, \tilde{R}, \tilde{V}_1, z_s)$

Hence, $\tilde{U}_1 = ({U_1}', \tilde{u})$, $\tilde{V}_1 = ({V_1}', \tilde{v})$, $\tilde{R} = \begin{pmatrix} R' & 0 \\ 0 & \sigma_s \end{pmatrix}$, and $\|E'\|_F = \sqrt{{\sigma_s}^2 + {\|E\|_F}^2}$

Let $[\sigma_l, y_l, z_l] = smallest\_sing(R')$

if $(\sqrt{{\sigma_l}^2 + {\|E'\|_F}^2} \leq \epsilon_F)$ *goto* ***Step 5****.*

else $\bar{U}_1 = {U_1}'$, $\bar{V}_1 = {V_1}'$, $\bar{R} = R'$, *and* $\|\bar{E}\|_F = \|E'\|_F$

Step 5:

$[{U_1}', R', {V_1}'] = TURV\_deflate({U_1}', R', {V_1}', z_l)$

Hence, ${U_1}' = (\bar{U}_1, u')$, ${V_1}' = (\bar{V}_1, v')$, $R' = \begin{pmatrix} \bar{R} & 0 \\ 0 & \sigma_l \end{pmatrix}$, and $\|\bar{E}\|_F = \sqrt{{\sigma_l}^2 + {\|E'\|_F}^2}$

Step 6:

$[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] = TURV\_refine(\tilde{U}_1, \tilde{R}, \tilde{V}_1, \|E\|_F)$

2.7 Subspace Tracking with TURVDs

Now that the concept of TURVDs is introduced and their updating techniques explained in detail, we can understand how they can be used for subspace tracking in [Algorithm 1] as an alternative to determining the SVD from scratch. The TURVD of the basis image matrix X at any instant of time is represented by the eq.(2.1). Let us assume that this matrix is mean-separated, meaning the mean (μ) of the columns of X is subtracted from each of its column, thereby obviating the necessity of X_m. Then, there are two cases when this decomposition has to be modified.

2.7.1 Addition of a New Basis Image

Let x be the vectorized form of the new basis image to be added to the matrix X. This is achieved by appending $x-\mu$ as the rightmost column of X, determining the updated mean (μ'), and finally adjusting the columns of the augmented X matrix for the updated mean. In mathematical terms, the augmented basis image matrix ($\bar{X}$) is obtained by the following expression:

$$\bar{X} \quad = \quad (X, x-\mu) + (\mu - \mu')e^T$$

where, $\mu' = \frac{1}{n+1}(n\mu + x)$, $e = (1, 1, 1, ..., 1)^T$ and $e \in \Re^{(n+1)}$. From the above expression, it is clear that the augmented matrix is obtained by a column addition followed by a rank-one update. Hence, the TURVD of $\bar{X}$ can be obtained from that of X by calling the following routines in the specified order:

- $[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] = TURVD\_update\_col(U_1, R, V_1, \|E\|_F, r, x-\mu)$
- $[\tilde{U}_1, \tilde{R}, \tilde{V}_1, \|\tilde{E}\|_F] = TURVD\_rank\_one\_update(\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F, r, (\mu - \mu'), e)$

Therefore, $\tilde{U}_1$, $\tilde{R}$, $\tilde{V}_1$, and $\|\tilde{E}\|_F$ represent the TURVD of $\bar{X}$.

2.7.2 Removal of a Basis Image

Whenever the number of basis images in X exceeds the threshold m, the least recently added basis image (the first column of X) is removed. Then the downdated matrix $\hat{X}$ is obtained by the following expression:

$$\hat{X} \quad = \quad X(:, 2:n) + (\mu - \mu')e^T$$

where, $\mu' = \mu - \frac{X(:,1)}{(n-1)}$, $e = (1, 1, 1, ..., 1)^T$ and $e \in \Re^{(n-1)}$. From the above expression, it is clear that the downdated matrix is obtained by a column removal followed by a rank-one update. Hence, the TURVD of $\hat{X}$ can be obtained from that of X by calling the following routines in the specified order:

- $[\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F] = TURVD\_downdate\_col(X, U_1, R, V_1, \|E\|_F)$
- $[\tilde{U}_1, \tilde{R}, \tilde{V}_1, \|\tilde{E}\|_F] = TURVD\_rank\_one\_update(\bar{U}_1, \bar{R}, \bar{V}_1, \|\bar{E}\|_F, r, (\mu - \mu'), e)$

Therefore, $\tilde{U}_1$, $\tilde{R}$, $\tilde{V}_1$, and $\|\tilde{E}\|_F$ represent the TURVD of $\hat{X}$.

Chapter 3

Experimental Results

This chapter is divided into two sections. Section 1 provides analysis of the results of experiments which determine the accuracy of the TURVD modifying techniques with respect to the results obtained by *ab initio* SVD calculation. It also provides the details of the metrics on the basis of which the accuracy of the TURVD results with respect to the SVD results, is determined. Section 2 compares the performance of both the subspace tracking decompositions when applied to real-world video data, in terms of the similarity of the basis images obtained and execution time. All the experiments were run on a stand-alone *Pentium IV, 3.2 GHz* machine, with the *Windows XP Professional Edition* operating system. *Matlab 7.0* was used to implement the algorithms and derive results. The algorithm used to calculate the SVD in these experiments is explained in detail in [6].

3.1 Accuracy of TURVD Techniques

3.1.1 Updating and Downdating a Column

In order to measure the accuracy of the TURVD updating/downdating with respect to *ab initio* SVD calculation, the *sliding window analysis technique* [3] was implemented. Sliding window analysis involves a matrix $A \in \Re^{m \times n}$, and *window size* $w < n$. In each iteration $1 \leq i \leq n - w$, matrix $A(i) = A(:, i: i + w)$ is obtained from $A(i - 1)$ by deleting its first column and adding the column of A that is to its immediate right. Hence, given the SVD of $A(i) = (Y_1(i), Y_2(i))\ \Sigma(i)\ (W_1(i), W_2(i))^T$, the following measurements reflect the accuracy of its corresponding TURVD $(U_1(i), R(i), V_1(i), E(i))$,

- The *rank estimate* of $A(i)$, which is equivalent to the dimension of $R(i)$.
- The *approximation error*, which is $\|A(i) - U_1(i)R(i)V_1(i)^T\|_F$.
- The *subspace error*, which is $\|U_1(i)^T Y_2(i)\|_2$.
- The *error in orthogonality*, which can be measured for both $U_1(i)$ and $V_1(i)$, which are $\|I - U_1(i)^T U_1(i)\|_F$ and $\|I - V_1(i)^T V_1(i)\|_F$.

3.1.1.1 Example 1

A is a 10×50 matrix with $w = 6$, whose elements are selected over a uniform distribution in the range $[0, 1]$. 30 of its randomly chosen columns are multiplied by 10^{-15}, with the tolerance parameter $\epsilon_F = 10^{-8}$. Fig. 3.1 depicts the results generated by the sliding window analysis with the given data. The TURVD is successful in accurately tracking the rank of $A(i)$ in all the iterations. The negative exponents of the different errors prove that the TURVD of each $A(i)$ is very accurate with respect to its corresponding SVD.

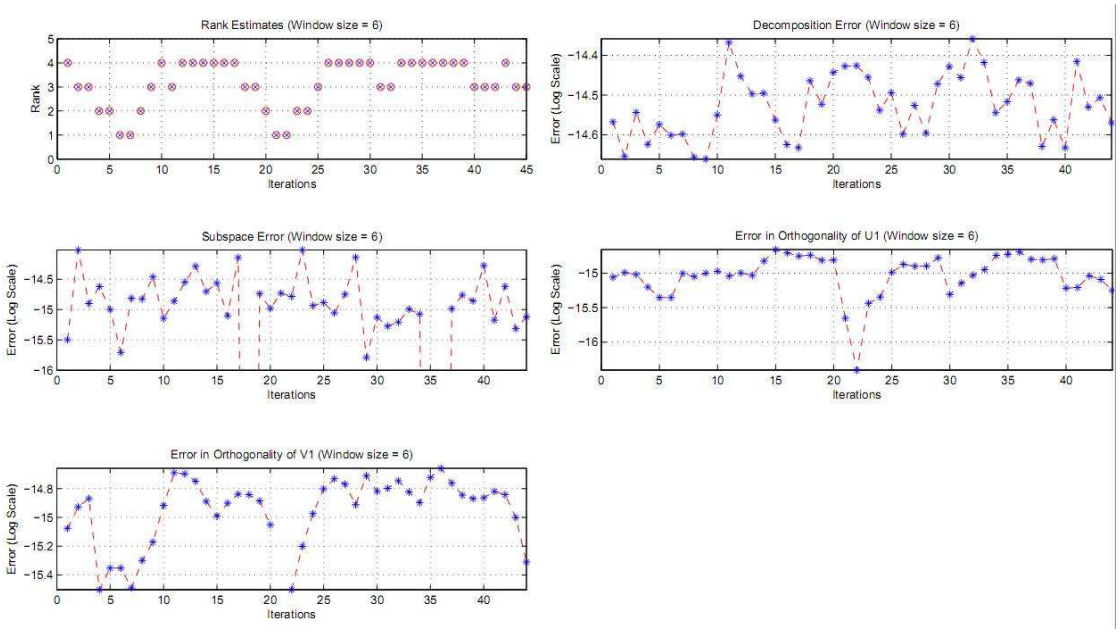

Figure 3.1. Accuracy of TURVD Update/Downdate (Example 1)

3.1.1.2 Example 2

A is a 10×50 matrix with $w = 6$, whose elements are selected over a uniform distribution in the range $[0, 1]$. 30 of its columns are randomly chosen to be the sum of their corresponding preceding and succeeding columns, with the tolerance parameter $\epsilon_F = 10^{-4}$. Fig. 3.2 depicts the results generated by the sliding window analysis with the given data.

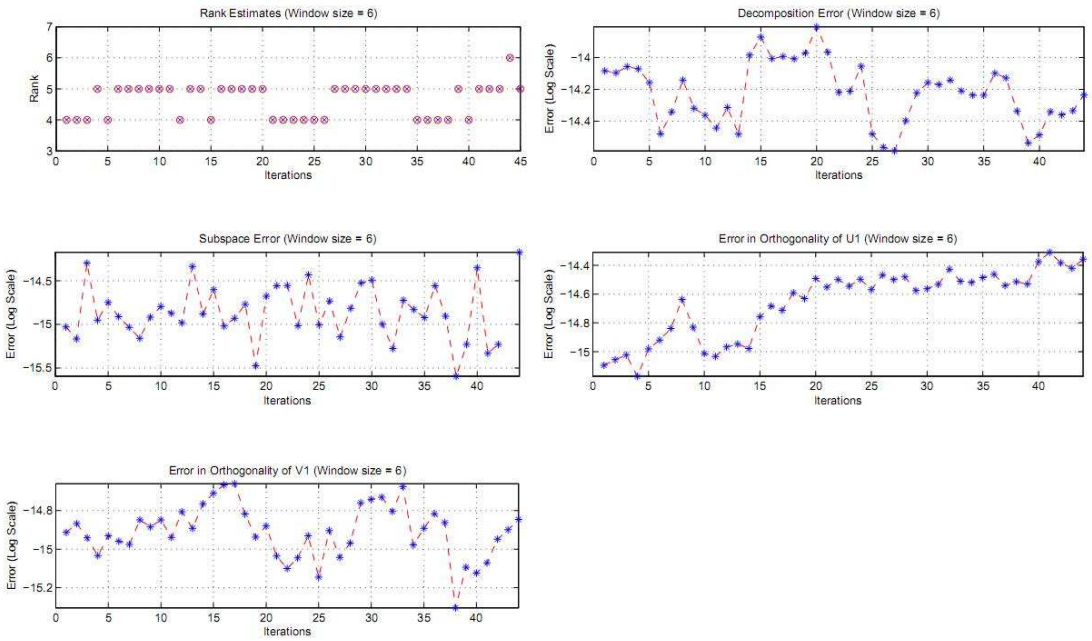

Figure 3.2. Accuracy of TURVD Update/Downdate (Example 2)

3.1.2 Rank One Update

In order to measure the accuracy of the TURVD rank-one update technique with respect to *ab initio* SVD calculation, the following analysis technique is used. Let $A \in \Re^{m \times n}$ which is to be updated. Let $F \in \Re^{m \times p}$ and $G \in \Re^{n \times p}$ be the matrices from which vectors f and g (see eqn.(2.4)) are chosen to perform the rank-one update on A. p is the number of iterations in the analysis algorithm. At any iteration i, $1 \leq i \leq p$, a rank-one update is performed on A by alternatively using one of the following two methods,

- Select f to be the i^{th} column of F, and g to be the i^{th} column of G, and perform rank-one update on A with these vectors.
- Select j in a uniformly random manner ($1 \leq j \leq p$) such that $f = \gamma A_j - A_j$ and $g = e_j$, where A_j is the j^{th} column of A and γ is very small. Perform rank-one update on A using these vectors.

The second method is necessary to decrease the rank of the A by one in order to test whether the rank-one update algorithm correctly estimates the rank. Hence, given the SVD of $A = (Y_1(i), Y_2(i))\ \Sigma(i)\ (W_1(i), W_2(i))^T$, the following measurements reflect the accuracy of its corresponding TURVD $(U_1(i), R(i), V_1(i), E(i))$,

- The *rank estimate* of A at iteration i, which is equivalent to the dimension of $R(i)$.
- The *approximation error*, which is $\|A - U_1(i)R(i)V_1(i)^T\|_F$.
- The *error in orthogonality*, which can be measured for both $U_1(i)$ and $V_1(i)$, which are $\|I - U_1(i)^T U_1(i)\|_F$ and $\|I - V_1(i)^T V_1(i)\|_F$.

3.1.2.1 Example

A is a 15×5 matrix , F is a 15×20 matrix and G is a 5×20 matrix, all of whose elements are selected over a uniform distribution in the range $[0, 1]$. The value of γ is set to 10^{-15}, and the tolerance parameter $\epsilon_F = 10^{-6}$. Fig. 3.3 depicts the results generated by applying the above analysis technique with the given data. The TURVD is successful in accurately tracking the rank of A in all the iterations. The negative exponents of the different errors prove that the TURVD of A in every iteration is very accurate with respect to its corresponding SVD.

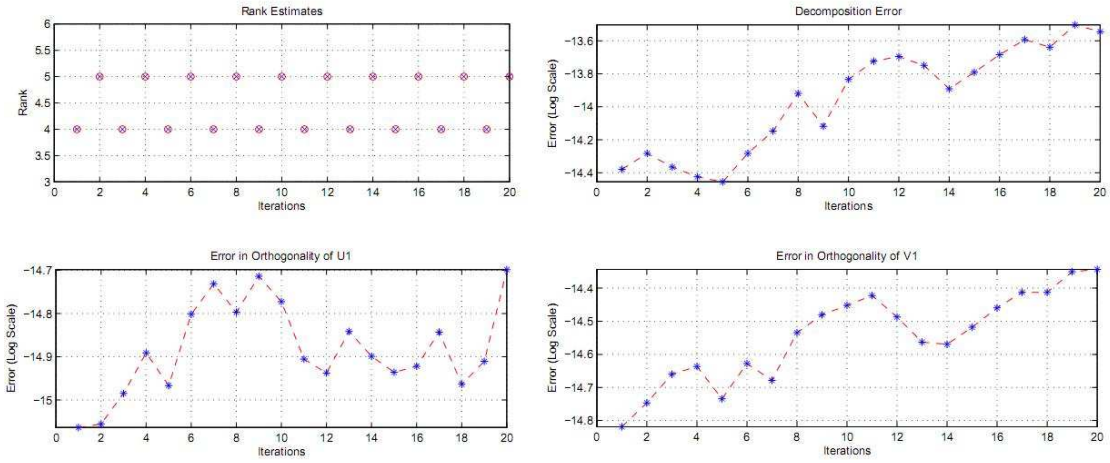

Figure 3.3. Accuracy of TURVD Rank One Update

3.2 Subspace Tracking Results

3.2.1 Example 1

This exmple uses a video sequence of a man moving a mug in circular fashion with his hands. By observing the video sequence that is reconstructed using the algorithm in section 2 with TURVD updates, it is clear that the movement of the hand is less smooth than the original video due to the loss of information due to projection onto the principal subspace. This video serves as a good approximation to the one recostructed using the tracking technique that uses the SVD algorithm in [6]. The basis images of both sequences at the end of the algorithm are given below. It is evident that the basis images generated by the TURVD updates serve as reasonable approximations of those generated by the SVD algorithm. The next example demonstrates the efficiency of the TURVD update techniques relative to the SVD algorithm.

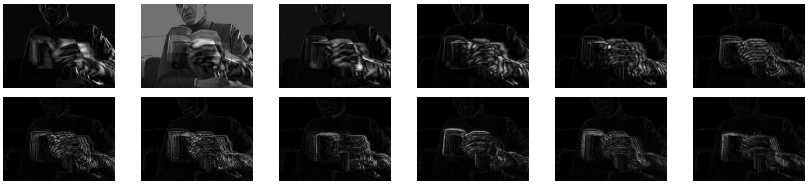

Figure 3.4. SVD Basis Images

Figure 3.5. TURVD Basis Images

3.2.2 Example 2

This example uses a video sequence of a man slowly turning his head in all four directions. This video is sourced from a video database used for research involving identity verification using speech and face information [7]. Here too, the motion of the head in the reconstructed video is less smooth the motion observed in the original video due to loss of information by projection onto the principal subspace. In this example, the maximum allowable basis images (m) is set to 30, and the percentage of basis images retained (r) is set to 90% (that is 27 out of the 30 basis images are retained throughout the execution of the algorithm). The following figure is a plot that depicts the execution times of both the SVD and TURVD update algorithms for all instances where the window matrix (X) is modified.

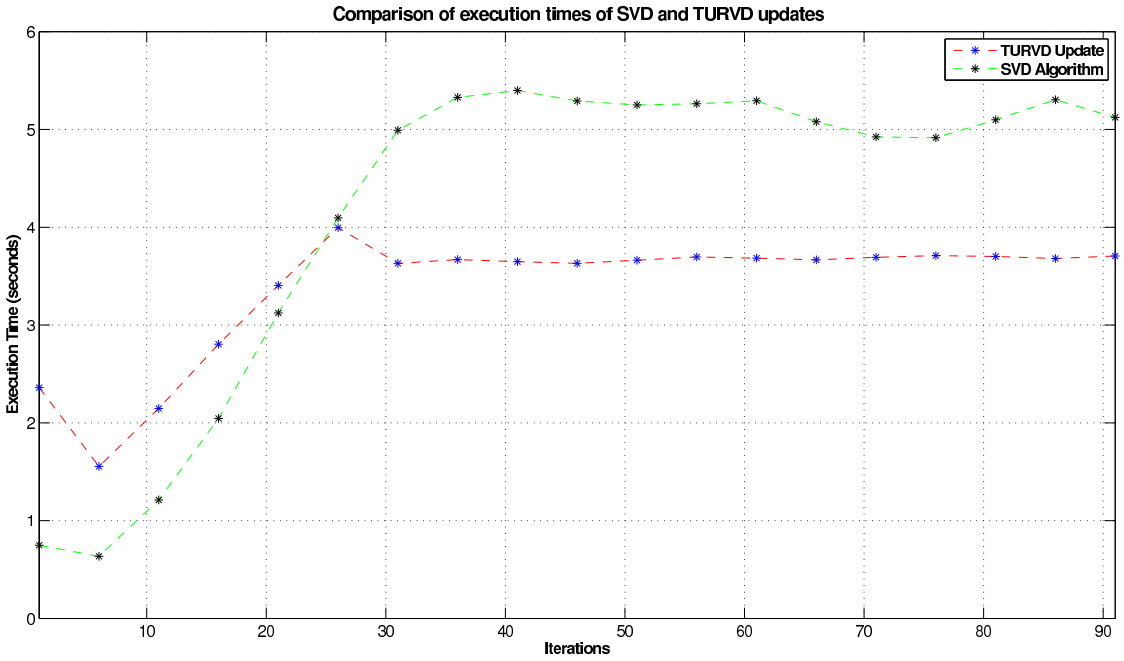

Figure 3.6. Execution Times of SVD and TURVD Algorithms

Initially, when X has fewer columns, the SVD algorithm performs as efficiently if not more than the TURVD update techniques because in such cases, the chasing and refinement routines used to update TURVDs have similar execution times as that of the SVD algorithm. But clearly, the execution time of the SVD algorithm increases more rapidly than its TURVD counterpart, as the number of basis images stored (column size of X) increases. For relatively larger number of columns, the TURVD update algorithm consistently performs better than the SVD algorithm, as seen in the right half of the plot. When the number of basis images reaches the maximum threshold m, the execution times of both the algorithms plateau. From this we can conclude that for very large matrices, maintaining their TURVDs is significantly more efficient than calcuating their SVDs from scratch every time they are modified.

Chapter 4

Conclusion

Formulating efficient encoding/decoding schemes for applications that stream video over a network is important in order to deal with varying bandwidths across different communication channels. In order to overcome the drawbacks of the *fixed basis set* approach followed by compression standards such as MPEG and CCIT H.261, Wu et. al. [1] have proposed a subspace tracking technique using the SVD to constantly update the basis image set. This technique involves the calculation of the SVD of the basis image set from scratch each time it is updated, which has proven to be computationally expensive for large sets. We have proposed an alternative rank-revealing decomposition, that is the Truncated URV Decomposition, which can be updated more efficiently than either calculating the SVD from scratch or updating the SVD itself. The above assertion has been substantiated by experimental results, which also prove that the TURVD provides reasonably accurate approximations of the left and right singular subspaces.

4.1 Future Work

The main contribution of this work has been the increase in the efficiency of the subspace tracking algorithm due to the usage of TURVDs instead of SVDs. The current subspace tracking algorithm, when required to discard a basis image from the set, does so in an ad-hoc fashion by removing that basis image that was added the earliest. One of the enhancements that can be implemented is to alter the algorithm such that it removes the basis image that has the least variance across a sliding window of previous frames. This can result in a more accurate reconstruction of a video sequence from the projection data. As to the TURVDs themselves, they can be used in other problems such as the *Total Least Squares Problem*, and the *Document Retrieval Problem*. The TURVDs are especially useful for solving rank-deficient systems because of their rank-revealing nature, and are more efficient than SVDs, both in terms of space (the error matrix of the TURVD is not stored), as well as time (demonstrated by experimental results).

Appendix A

Rank Revealing Decompositions

A.1 Singular Value Decomposition

Matrix algorithms based on orthogonal transformations play a significant role in matrix computations. This is because these transformations are numerically very stable and they preserve the two-norm and the Frobenius norm. Most importantly, these decompositions can yield information about certain subspaces which makes them an invaluable tool for signal processing applications. The most famous of these orthogonal transformations is the Singular Value Decomposition (SVD). Its claim to fame is that it can very reliably detect near-rank deficiency of a matrix and yields all the necessary subspace information. The SVD of an $m \times n$ matrix X with $m \geq n$ is given by

$$X \quad = \quad U \Sigma V^T$$

where $U \in \Re^{m \times n}$ and $V \in \Re^{n \times n}$ are unitary matrices that are the *left singular* and *right singular* matrices respectively. In other words, the subspace spanned by the columns of U $(R(U))$ is called the *left singular subspace* of X, and the subspace spanned by the columns of V $(R(V))$ is called

the *right singular subspace* of X. $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_n)$ is a diagonal matrix whose diagonal elements are the n singular values of X.

Some of the most important applications of the SVD include solving ill-conditioned least squares problems, total least squares problems, subspace tracking, and isolating principal components. In these applications, our interest in the singular value decomposition is to write X in the form

$$X = U_1 \Sigma_1 {V_1}^T + U_2 \Sigma_2 {V_2}^T \tag{A.1}$$

where $\Sigma_1 = diag(\sigma_1, \sigma_2, ..., \sigma_k)$, $\Sigma_2 = diag(\sigma_{k+1}, \sigma_{k+1}, ..., \sigma_n)$ and U_1, U_2, V_1 and V_2 are the corresponding left and right singular matrices respectively.

Computing the SVD of a matrix is an $O(mn^2)$ operation, which is not a big price to pay considering the invaluable subspace and rank information it provides, but if our application requires frequent updating of the decomposition due to changes in X, then we have a problem because updating the SVD is also of $O(mn^2)$ complexity, which is unacceptable for applications with real-time constraints.The main overhead in updating SVDs is in the maintenance of the diagonality of the matrix Σ. Many real-time applications require only approximate subspace information and therefore constructing the SVD type dichotomy might be an overkill. We can utilize alternative decompositions as long as the subspace information is reasonably accurate and the rank-revealing structure is maintained. One such alternative is the *UTV Decomposition*, which is explained in the following section.

A.2 UTV Decomposition

A *UTV decomposition* of a matrix $X \in \Re^{m \times n}$ is of the form

$$X = UTV^T$$

where $U \in \Re^{m \times n}$ and $V \in \Re^{n \times n}$ are unitary matrices that are the approximations of *left singular* and *right singular* matrices of the corresponding SVD respectively, and $T \in \Re^{n \times n}$ is triangular. By having a triangular matrix in the middle of the decomposition instead of a diagonal one (as in the case of the SVD), the time complexity of updating operations can be reduced to $O(mn)$ without compromising on its rank-revealing structure.

There are two catergories of UTV decompositions, namely *ULV decompositions*, where L is lower triangular, and *URV decompositions*, where R is upper triangular. In this work, we use a truncated form of the URV decomposition defined by eq.(2.1), which is derived as follows. The URV decomposition of X is said to be *rank-revealing* if it can be partitioned in the form

$$X = (U_1\ U_2) \begin{pmatrix} R & H \\ 0 & F \end{pmatrix} \begin{pmatrix} {V_1}^T \\ {V_2}^T \end{pmatrix}$$

where $R \in \Re^{k \times k}$, $\|R^{-1}\|_2 \leq \epsilon^{-1}$, $\|F\|_2 \leq \epsilon$ for some tolerance ϵ, and $\|H\|_2$ is sufficiently small. This means that X has a gap in its singular values at k. Hence,

$$X = U_1 R {V_1}^T + U_1 H {V_2}^T + U_2 F {V_2}^T$$

From the above expansion, we can observe that, the smaller the value of $\|H\|_2$, the closer the above equation resembles the rank-revealing structure of the SVD given by eq.(A.1). Let $E = U_1 H {V_2}^T + U_2 F {V_2}^T$. Then the above equation is reduced to the desired form of eq.(2.1).

Appendix B

Matrix Computation Tools

B.1 LINPACK-style Condition Estimator

A condition estimator typically involves estimating the norm of the inverse of a triangular matrix. Since this work utilizes URV decompositions, we consider the upper triangular matrix R. Hence, if $\|.\|$ is the norm operator, then

$$\|R^{-1}\| \quad = \quad \max_{\|u\|=1} \|R^{-1}u\|$$

We can rewrite the above definition in order to avoid the explicit usage of the inverse of R as follows.

$$\|R^{-1}\| \quad = \quad \max\{\|v\| \ : \ Rv = u, \ \|u\| = 1\}$$

This means that we can approximate $\|R^{-1}\|$ by choosing a suitable u with $\|u\| = 1$, and solve the system of equations

$$Rv = u \tag{B.1}$$

By "suitable", we mean that a u such that $\|v\|$ is large. One of the methods to estimate $\|R^{-1}\|$, is given by the *LINPACK-style Condition Estimator*, which chooses the components of the vector u dynamically while solving the linear system (B.1). The following is an algorithm to estimate the 2-norm of R^{-1}.

Algorithm 2 LINPACK-style 2-norm Condition Estimator

Input R, n
Output $v, norm_R$
for $i = n$ to 1, decrement by 1 **do**
  $d = R(i, i+1:n) \times v(i+1:n)$
  if $d \geq 0$ **then**
    $v(i) = -\frac{(1+d)}{R(i,i)}$
  else
    $v(i) = \frac{(1-d)}{R(i,i)}$
  end if
end for
$v = \frac{v}{\sqrt{n}}$
$inf = \frac{1}{\|v\|_2}$
$v = inf \times v$
$norm_R = \|v\|_2$

In the above algorithm, during the forward substitution phase of solving the linear system (B.1), the i^{th} component of the vector v is given by

$$v(i) \quad = \quad \frac{u(i) - R(i, i+1:n) \times v(i+1:n)}{R(i,i)} \tag{B.2}$$

The value for $u(i)$ is dynamically chosen as follows

$$u(i) = \begin{cases} -1 & \text{if } R(i, i+1:n) \times v(i+1:n) \geq 0 \\ 1 & \text{if } R(i, i+1:n) \times v(i+1:n) < 0 \end{cases}$$

The above-mentioned choice encourages growth in the norm of v by assuring that there is no cancellation in the numerator of the right-hand side of eq.(B.2).

B.2 Gram-Schmidt Orthogonalization

The *Gram-Schmidt* process is an algorithm that produces orthonormal bases. Let S be a subspace of $\Re^n$, and let $v_1, v_2, ..., v_m$ be a basis of S. The Gram-Schmidt process uses this basis to produce its corresponding orthonormal basis $q_1, q_2, ..., q_m$, such that $S = span\{v_1, v_2, ..., v_m\} = span\{q_1, q_2, ..., q_m\}$. The crux of the Classical Gram-Schmidt (CGS) process is explained as follows.

Suppose we have determined the first $k-1$ ($k \leq m$) orthonormal vectors $q_1, q_2, ..., q_{k-1}$ such that $span\{v_1, v_2, ..., v_{k-1}\} = span\{q_1, q_2, ..., q_{k-1}\}$. Therefore, the k^{th} orthonormal vector q_k is determined as follows. Let $\hat{q}_k$ be defined as

$$\hat{q}_k \quad = \quad v_k - \{\sum_{j=1}^{k-1} q_j {q_j}^T\} v_k, \text{ and } q_k = \frac{1}{\|\hat{q}_k\|_2} \times \hat{q}_k$$

Hence, q_k is orthogonal to $q_1, q_2, ..., q_{k-1}$, and $span\{v_1, v_2, ..., v_k\} = span\{q_1, q_2, ..., q_k\}$. The algorithm for the Classical Gram-Schmidt process is given below.

One of the drawbacks of the CGS algorithm is that there is a loss of orthogonality among the $q_i's$ due to numerical errors. One of the ways to mitigate this problem is to use a *Re-orthogonalization* procedure. More details about re-orthogonalization can be found in [5].

Algorithm 3 Classical Gram-Schmidt Orthogonalization

1: Input $v_1, v_2, ..., v_m$
2: Output $q_1, q_2, ..., q_m$
3: **for** $k = 1$ to m, increment by 1 **do**
4: $\quad q_k = v_k$
5: $\quad$ **for** $i = 1$ to $k - 1$, increment by 1 **do**
6: $\quad\quad q_k = q_k - q_i {q_i}^T v_k$
7: $\quad$ **end for**
8: $\quad q_k = \frac{1}{\|q_k\|_2} \times q_k$
9: **end for**

B.3 Plane Rotations

The chasing and refinement routines require some mechanism to preserve the triangular structure of the matrix R. For this purpose, we use *Plane Rotations* in this work. Since a plane rotation is an orthogonal operation, the norms of the vectors it operates on are unchanged. A *plane rotation in the i-j plane* is an $n \times n$ matrix $P_{ij}(\theta)$, whose elements are defined below.

$$p_{xy}(\theta) = \begin{cases} c & \text{if } (x,y) = (i,i) \text{ or } (x,y) = (j,j) \\ s & \text{if } (x,y) = (i,j) \\ -s & \text{if } (x,y) = (j,i) \\ 1 & \text{if } x = y, (x,y) \neq (i,i), (x,y) \neq (j,j) \\ 0 & \text{otherwise} \end{cases}$$

where, $c = cos(\theta)$ and $s = sin(\theta)$, for some $\theta \in [0, 2\pi]$. This means that a premultiplication of the matrix ${P_{ij}(\theta)}^T$ to any vector $x \in \Re^n$ amounts to rotating the vector x by an angle θ along the *i-j plane*. That is, if $y = {P_{ij}(\theta)}^T x$, then, for any $1 \leq k \leq n$,

$$y_k = \begin{cases} cx_i + sx_j & \text{if } k = i \\ cx_j - sx_i & \text{if } k = j \\ x_k & \text{otherwise} \end{cases}$$

One of the uses of plane rotations is to introduce zeros in vectors without affecting their norms. For instance, $P_{ij}(\theta)$ can be used to introduce a zero in the j^{th} component of a vector x by choosing the following values for c and s.

$$c = \frac{x_i}{\sqrt{{x_i}^2 + {x_j}^2}}, \text{ and } s = \frac{x_j}{\sqrt{{x_i}^2 + {x_j}^2}}$$

The following algorithms can be used to develop utility routines for generating and applying plane rotations to induce zeros in vectors.

Algorithm 4 To generate a plane rotation denoted by c and s which introduces a zero in the place of b

1: Input a, b
2: Output c, s
3: $\tau = |a| + |b|$
4: **if** $\tau = 0$ **then**
5: $\quad c = 1$ and $s = 0$
6: **else**
7: $\quad \gamma = \tau \times \sqrt{(\frac{a}{\tau})^2 + (\frac{b}{\tau})^2}$
8: $\quad c = \frac{a}{\gamma}$ and $s = \frac{b}{\gamma}$
9: **end if**

Algorithm 5 To apply a plane rotation denoted by c and s to two vectors x and y

1: Input x, y, c, s
2: Output $\hat{x}, \hat{y}$
3: $\hat{x} = c \times x + s \times y$
4: $\hat{y} = c \times y - s \times x$

Note that the scaling factor τ is introduced to avoid overflows and make underflows harmless. The above algorithms are useful in implementing the chasing and refining routines, as eveident in the next two sections.

B.4 Deflation (Chasing)

While updating the TURVD of a matrix, a situation might arise where the smallest singular value of R might be smaller than the tolerance parameter ϵ. In such a situation, we need to alter the decomposition to reveal the smallest singular value. This process is called *deflation*. The basic idea of deflation is as follows. When we estimate the smallest singular value σ_s of R using Algorithm (2), we obtain a vector w such that $\sigma_s = \|Rw\|_2$. In order to reveal this singular value, we determine orthogonal matrices Q_1 and Q_2, such that ${Q_1}^T w = e_k$, where k is the order of the matrix R, and ${Q_2}^T R Q_1$ remains upper triangular. Then, we have

$$\begin{aligned} \sigma_s &= \|Rw\|_2 \\ &= \|({Q_2}^T R Q_1){Q_1}^T w\|_2 \\ &= \|\bar{R} e_k\|_2 \end{aligned}$$

Now, the last column of $\bar{R}$ is of 2-norm σ_s, and can be moved into the error matrix E by updating the error norm. The following algorithm describes the deflation procedure.

Algorithm 6 To deflate the smallest singular value of R to its last column

1: Input U_1, R, V_1, w
2: Output U_1, R, V_1
3: **for** $i = 1$ to $k-1$ increment by 1 **do**
4: Use Algorithm (4) {Input: $w(i+1), w(i)$ | Output: c, s}
5: Use Algorithm (5) {Input: $R(1:i+1, i+1), R(1:i+1, i), c, s$ | Output: $R(1:i+1, i+1), R(1:i+1, i)$}
6: Use Algorithm (5) {Input: $V_1(:, i+1), V_1(:, i), c, s$ | Output: $V_1(:, i+1), V_1(:, i)$}
7: Use Algorithm (4) {Input: $R(i, i), R(i+1, i)$ | Output: c, s}
8: Use Algorithm (5) {Input: $R(i, i+1:k), R(i+1, i+1:k), c, s$ | Output: $R(i, i+1:k), R(i+1, i+1:k)$}
9: Use Algorithm (5) {Input: $U_1(:, i), U_1(:, i+1), c, s$ | Output: $U_1(:, i), U_1(:, i+1)$}
10: **end for**

Bibliography

[1] Wu, H., D. Ponceleon, K. Wang, and J. Normile (1994) "Tracking Subspace Representation of Face Images," *IEEE Conf. on Acoustics, Speech and Signal Processing*, **5**, pp. 389–392.

[2] Stewart, G. W. (1994) "Updating URV Decompositions in Parallel," *Parallel Comp.*, **20**, pp. 151–172.

[3] Barlow, J. L. and H. Erbay (2007) "Modifiable Low-Rank Approximation to a Matrix," *In preparation.*

[4] Barlow, J. L., A. Smoktunowicz, and H. Erbay (2006) "Improved Gram-Schmidt Type Downdating Methods," *BIT Numerical Mathematics*, **45**, pp. 259–285.

[5] Stewart, G. W. (1998) *Matrix Algorithms: Basic Decompositions*, vol. 1, Society for Industrial and Applied Mathematics.

[6] Larsen, R. M. (1998) "Lanczos Bidiagonalization With Partial Reorthogonalization," *Technical Report, DAIMI PB*, **357**.

[7] Sanderson, C. and K. K. Paliwal (2004) "Identity Verification Using Speech and Face Information," *Digital Signal Processing 14*, **5**, pp. 449–480.