A Study of Vandermonde-like Matrix Systems With Emphasis on Preconditioning and Krylov Matrix Connection.

By

Jyoti Saraswat

Submitted to the Department of Mathematics and the
Faculty of the Graduate School of the University of Kansas
in partial fulfillment of the requirements for the degree of

Master of Arts

Committee members

Dr. Hongguo Xu ,Math Dept.,
Chairperson

Dr. Weizhang Huang ,Math Dept.

Dr. Erik Van Vleck ,Math Dept.

Date defended: June 09,2009

The Dissertation Committee for Jyoti Saraswat certifies
that this is the approved version of the following thesis:

A Study of Vandermonde-like Matrix Systems With Emphasis on Preconditioning and Krylov Matrix Connection.

Committee:

Dr. Hongguo Xu ,Math Dept.,
Chairperson

Dr. Weizhang Huang ,Math Dept.

Dr. Erik Van Vleck ,Math Dept.

Date approved: June 15,2009

Contents

Abstract

The study focuses primarily on Vandermonde-like matrix systems. The idea is to express Vandermonde and Vandermonde-like matrix systems as the problems related to Krylov Matrices. The connection provides a different angle to view the Vandermonde-like systems. Krylov subspace methods are strongly related to polynomial spaces, hence a nice connection can be established using LU factorization as proposed by Bjorck and Pereyra [2] and QR factorization by Reichel [11]. Further an algorithm to generate a preconditioner is incorporated in GR algorithm given by Reichel [11]. This generates a preconditioner for Vandermonde-like matrices consisting of polynomials which obey a three term recurrence relation. This general preconditioner works effectively for Vandermonde matrices as well. The preconditioner is then tested on various distinct nodes. Based on results obtained, it is established that the condition number of Vandermonde -like matrices can be lowered significantly by application of the preconditioner, for some cases.

Chapter 0

Some Useful Definitions and Notations.

0.1 Notations

$\mathbb{R}$:- Real number field

$\mathbb{R}^n$:- n-dimensional real vector space

$\mathbb{R}^{n\times n}$:- Space of $n \times n$ real matrices

$\mathbb{P}$:- Space of polynomials

$\mathbb{P}_n$:- Space of polynomials with degree less than or equal to n

$\mathbb{C}$:- Complex number field

$\mathbb{C}^n$:- n-dimensional complex vector space

$\mathbb{C}^{n\times n}$:- Space of $n \times n$ complex matrices

$C[a,b]$:- Space of continuous functions on the interval [a,b]

$\|\cdot\|$:- A matrix norm is a function $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$ satisfying the three vector norm properties. The Frobenius norm for a matrix $A \in \mathbb{R}^{m\times n}$ is given by

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2} = \sqrt{(tr(A^TA))}.$$

The Euclidean norm or the two norm of a matrix $A \in \mathbb{R}^{m\times n}$ is given by

$$\|A\|_2 = \sqrt{(\rho(A^TA))}, \text{ where } \rho(A^TA) \text{ is the spectral radius of } A^TA.$$

0.2 Definitions

0.2.1 Vandermonde Matrix.

A classical $(n+1)\times(n+1)$ Vandermonde matrix is defined as follows

$$\breve{V} = \begin{bmatrix} 1 & x_1 & \dots & x_1^n \\ 1 & x_2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n+1} & \dots & x_{n+1}^n \end{bmatrix}.$$

Vandermonde matrices generally arise when matrix methods are used in problems of polynomial interpolation, in solution of differential equations, and in analysis of recursively defined sequences.

0.2.2 Vandermonde-like Matrix.

A Vandermonde like matrix is defined by

$$V\left[p_{j-1}(x_i)\right]_{1\le j\le m+1, 1\le i\le n+1} = \begin{bmatrix} p_0(x_1) & p_1(x_1) & \dots & p_m(x_1) \\ p_0(x_2) & p_1(x_2) & \dots & p_m(x_2) \\ \vdots & \vdots & \dots & \vdots \\ p_0(x_{n+1}) & p_1(x_{n+1}) & \dots & p_m(x_{n+1}) \end{bmatrix}$$

$$= \left[v_{kj}\right]_{1\le k\le n+1, 1\le j\le m+1}$$

where p_j is a polynomial of degree j.

0.2.3 Krylov Matrices.

For a given matrix $A \in \mathbb{R}^{n\times n}$, and a given vector $x \in \mathbb{R}^n$, a Krylov matrix is defined as

$$K(A,x) = K_n(A,x) = \left[x, Ax, A^2x, A^3x, \ldots, A^{n-1}x\right] \in \mathbb{R}^{n\times n}.$$

Krylov subspace methods form an important class of iterative methods while solving for large scale system of linear equations and eigenvalue problems. For details refer [16].

0.2.4 Condition Number of a Matrix.

Condition number of a matrix $A \in \mathbb{R}^{n\times n}$ is defined as

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

where $\|\cdot\|$ is a norm defined on the matrix space $\mathbb{R}^{n\times n}$. A problem with a small condition number is said to be well-conditioned, while a problem with a large condition number is said to be ill-conditioned. If matrix A is singular, i.e , $\det(A) = 0$, then $\kappa(A) = \infty$.

0.2.5 Orthogonal Polynomials Satisfying a Recurrence Relation.

Define the inner product $\langle p, q\rangle_w = \int_a^b w(x)p(x)q(x)dx, \quad \forall p(x), q(x) \in \mathbb{P}$, where $w(x) > 0$ for $a \leq x \leq b$ is a weight function. A sequence of polynomials $\{p_0(x), \ldots p_k(x), \ldots\}$ are orthogonal if $\langle p_i(x), p_j(x)\rangle_w = 0$, for $i \neq j$. It is orthonormal if it further satisfies

$\langle p_i(x), p_i(x)\rangle_w = 1, \quad \text{for } i = 0, 1, \ldots.$

For any weight function $w(x)$ and interval $[a,b]$, so that $\langle\cdot,\cdot\rangle_w$ is well defined, a sequence of orthogonal polynomials can be constructed by applying the Gram-Schmidt orthogonalization to $1, x, x^2, \ldots, x^k, \ldots$. Such a sequence of polynomials $\{p_0(x), \ldots p_k(x), \ldots\}$ always satisfies a three term recurrence relation and $\deg p_k(x) = k$ for $k = 0, 1, \ldots$. For instance when $w(x) = (1-x^2)^{-1/2}$ with limits of orthogonality being $[-1,1]$, we get the well known sequence of polynomials known as Chebyshev polynomials, $T_k(x) = \cos(k \arccos(x))$. For $w(x) = 1$, a sequence of polynomials known as Legendre polynomials, is given by Rodrigues formula

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left[(x^2-1)^n\right], \quad x \in [-1,1].$$

Let $x_1, x_2, \ldots, x_{n+1}$ be distinct real numbers and let $w = [w_1, w_2, \ldots, w_{n+1}]$ with $w_1, w_2, \ldots, w_{n+1} > 0$, a discrete-type inner product can be defined for the polynomial space $\mathbb{P}_n$ as,

$$\langle p,q\rangle_w = \sum_{i=1}^{n+1} p(x_i)q(x_i)w_i^2 \quad \forall \quad p(x), q(x) \in \mathbb{P}_n.$$

0.2.6 Preconditioner.

For a given system of linear equations $Ax = b$, one intends to find a non-singular matrix P to transform the system to $PAx = Pb$ so that the condition number of PA is (hopefully) much smaller than condition number of A, and the cost of computation of such a P is not very expensive. Matrix P is called a preconditioner of A. The ideal choice of P is A^{-1}, but this is impractical as computation of A^{-1} is much more expensive than solving $Ax = b$. In practice, it is required that P is in a certain simple form. In this study we will restrict P to be diagonal.

0.2.7 Hessenberg Matrix.

A matrix $H \in \mathbb{R}^{n\times n}$ is said to be upper Hessenberg if $h_{ij} = 0$ whenever $i > j+1$. This means that the matrix has a nearly triangular form

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix}.$$

H is said to be lower Hessenberg if $h_{ij} = 0$ whenever $i < j-1$.

0.2.8 Orthogonal Matrix.

A matrix $Q \in \mathbb{R}^{n\times n}$ is said to be orthogonal if $QQ^T = I$. This implies that Q has an inverse and, $Q^{-1} = Q^T$. By the property that a matrix commutes with its inverse, we have $Q^TQ = I$. Further $\det(Q) = \pm 1$, and also $\forall x \in \mathbb{R}^n$, $\|Qx\|_2 = \|x\|_2$.

0.2.9 Givens Rotation.

Givens rotations are effective tools for introducing zeros on a grand scale selectively. A Givens rotation is given in matrix form as follows.

$$G(i,k,\theta) = \begin{array}{c} \\ \\ \\ i \\ \\ k \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & i & & k & & \end{array} \\ \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix} \end{array}, \quad c = \cos(\theta), \quad s = \sin(\theta), \quad c^2 + s^2 = 1,$$

for some rotation angle θ. A Givens rotation $G(i,k,\theta)$, when applied to a matrix A from left, affects the i and k rows only of A.

0.2.10 Newton's Divided Difference

Let $f(x)$ be a function, defined on $(n+1)$ distinct points on its domain. Let $p(x) \in \mathbb{P}_n$ be the interpolating polynomial of at most degree n, approximating $f(x)$ at these points. The general form of the interpolating polynomial based on Newton's divided difference for $(n+1)$ data points, $(x_1, f(x_1), \dots, (x_{n+1}, f(x_{n+1}))$ is given by

$$p(x) = b_0 + b_1(x-x_1) + b_1(x-x_1)(x-x_2) + \cdots + b_n(x-x_1)\dots(x-x_n),$$

where

$$b_0 = f[x_1] = f(x_1),$$

$$b_1 = f[x_1, x_2] = \frac{f(x_1) - f(x_2)}{x_1 - x_2},$$

and proceeding in this way, the k^{th} term is given by

$$b_k = f\left[x_1, \ldots, x_{k+1}\right] = \frac{f\left[x_1, x_2, \ldots, x_k\right] - f\left[x_2, x_3, \ldots, x_{k+1}\right]}{x_1 - x_{k+1}}.$$

The terms b_i are known as Newton's divided difference of i^{th} order.

Chapter 1

Introduction

1.1 Overview

There is more to Vandermonde matrix than meets the eye. A Vandermonde matrix is defined as follows

$$\breve{V} = \begin{bmatrix} 1 & x_1 & \dots & x_1^n \\ 1 & x_2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n+1} & \dots & x_{n+1}^n \end{bmatrix}.$$

The determinant of $\breve{V}$ is given by the compact formula $\det(\breve{V}) = \prod_{1 \le i < j \le n+1} (x_j - x_i)$. Before going into detail, let us talk a little about polynomial interpolation. Suppose we have $n+1$ distinct points $(x_1, y_1), (x_2, y_2), \dots, (x_{n+1}, y_{n+1})$. The process of fitting an n degree polynomial to these points is usually referred to as polynomial interpolation. If the polynomial is $p(x) = a_0 + a_1 x + \dots + a_n x^n$, then the coefficients a_i can be determined by solving the equations $p(x_i) = y_i$ for $i = 1, \dots, n+1$.

The matrix form of such a system is given by

$$\begin{bmatrix} 1 & x_1 & \dots & x_1^n \\ 1 & x_2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n+1} & \dots & x_{n+1}^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n+1} \end{bmatrix}.$$

Observe that the coefficient matrix is a Vandermonde matrix. The determinant is nonzero if all x_i are distinct.

The Vandermonde matrix is generated by the monomials $1, x, \dots, x^n$ at the given nodes. For this study we chose real nodes. Since we will be discussing QR factorization by tridiagonal reduction, we avoid complex nodes. If we replace the monomials in the Vandermonde matrix by polynomials $p_0(x), p_1(x), \dots, p_n(x)$, the resulting matrix is a Vandermonde-like matrix.

Vandermonde matrices are an important tool due to their connection to FFT. They can also be used to solve minmax problem [10]. Vandermonde matrices also find use in the solution of multivariate interpolation problem [4]. In the study, the main focus would be on Vandermonde-like matrices, especially when $p_j(x)$ satisfy a recurrence relation. The reason of such interest in Vandermonde-like matrices is based on the fact that for certain polynomials they tend to be better conditioned than classical Vandermonde matrices.

1.2 Goal

One of the goals of the study is to reveal some basic relations of Vandermonde and Vandermonde-like matrices and Krylov Matrices. We make an effort to look at the interpolation problem in context of Krylov matrix. We will discuss the relation with LU factorization given by Bjorck and Pereyra [2] and the QR factorization by Reichel in [11]. The focus is also on finding a preconditioner, in this case a diagonal matrix. The condition number in Frobenius norm, is computed by taking all weights unity and then the optimal weights from the preconditioner. A comparison of condition numbers is made to detect any improvement upon the condition number. The study also discusses the inverse of Vandermonde matrices.

1.3 Outline

The study is broken into four areas. Chapter 2 reviews few theorems which are used in the study. Chapter 3 focuses on the fast LU factorization of Vandermonde matrices. Using the tools, which were used in this process, a connection is established between Vandermonde matrices and Krylov matrices. Further the same tools are used in chapter 5 to find the inverse.

Chapter 4 focuses on fast QR factorization. For doing so Vandermonde-like matrices are considered, which are constructed using orthogonal polynomials with three term recurrence relation. The GR algorithm is used to compute the upper triangular matrix R for a given Vandermonde-like matrix. The study is not restricted to only Vandermonde-like matrices. It is also shown that the same can be extended to classical Vandermonde and Vandermonde-like matrix with polynomials not following three term recurrence.

Chapter 5 discusses work of select authors on finding inverse of Vandermonde matrix. This chapter also shows how the method given by Traub [12] can be connected to Krylov matrix in finding the inverse of Vandermonde matrix.

Chapter 6 focuses in deriving the preconditioner for the Vandermonde-like matrices, which is restricted to be diagonal. The diagonal preconditioner is incorporated with the GR algorithm to see if an improvement on condition number of the Vandermonde-like matrices is obtained. A comparison is made between condition numbers obtained by using programming in MATLAB. Finally some conclusions are drawn in the last chapter.

Chapter 2

Theory of HR Factorizations.

2.1 Overview: HR Factorizations.

This section will go over some basic theory. A few theorems from [6] are cited in this section.

Theorem 1. *[6][QR Decomposition] Given $A \in \mathbb{R}^{n\times n}$, there exists a real orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ i.e. $Q^T Q = I_n$, and an upper triangular matrix R such that $A = QR$. Moreover if A is non-singular and there exists another QR decomposition $A = \tilde{Q}\tilde{R}$ then $\tilde{Q} = QD$, $\tilde{R} = DR$, where $D = diag\left[d_1, d_2, \ldots, d_n\right], \quad d_i = \pm 1, \quad i = 1, 2, \ldots, n$, i.e, D is a signature matrix.*

Theorem 2. *[6][LU factorization] Suppose $A \in \mathbb{R}^{n\times n}$, and $\det A_k \neq 0, \quad \forall\, A_k$, the leading principal $k \times k$ submatrix of A, $k = 1, 2, \ldots, n-1$. Then there exists a lower triangular L and a unit upper triangular matrix U such that*

$$A = LU.$$

Moreover the factorization is unique.

Note that usually for an LU factorization, L is unit lower triangular and U is upper triangular. The factorization described in the above theorem is slightly different, but they are essentially the same by connecting both to the LDU factorization with L, U being unit lower and unit upper triangular respectively and D diagonal .

Theorem 3. *Suppose $A \in \mathbb{R}^{n\times n}$, $0 \neq b \in \mathbb{R}^n$, and*

$$K = K(A,b) = \left[b, Ab, A^2b, \dots, A^{n-1}b\right] \in \mathbb{R}^{n\times n}.$$

Suppose $X \in \mathbb{R}^{n\times n}$ is non-singular. If

$$X^{-1}AX = H, \quad X^{-1}b = \sigma e_1, \tag{2.1}$$

where H is upper Hessenberg and $\sigma \in \mathbb{R}$, and e_1 is the first column of I, the identity matrix, then

$$K = XR \tag{2.2}$$

where $R = \sigma\left[e_1, He_1, \dots, H^{n-1}e_1\right]$ is upper triangular.
Conversely, if R is non-singular(so is K), then (2.2) implies (2.1).
Moreover, if X is either lower triangular or real orthogonal, and R is non-singular, then (2.1) and (2.2) are essentially unique, meaning if there is another $\tilde{X}$ satisfying both (2.1) and (2.2). Then $\tilde{X} = XD$, where D is a non-singular diagonal matrix.

Proof. (2.1) $\Longrightarrow$ (2.2)
The equation $X^{-1}b = \sigma e_1$ gives $b = \sigma X e_1$. Notice if $X^{-1}AX = H$, then $X^{-1}AXX^{-1}AX = H^2$ or $X^{-1}A^2X = H^2$. Which further implies that for any $k \in \mathbb{Z}$, $\quad X^{-1}A^{k-1}X = H^{k-1}$.
Now

$$K = \left[b, Ab, A^2b, \dots, A^{n-1}b\right]$$

$$= \left[b, XHX^{-1}b, \ldots, XH^{n-1}X^{-1}b\right]$$

$$= \left[\sigma Xe_1, XH\sigma e_1, \ldots, XH^{n-1}\sigma e_1\right]$$

$$= \sigma X\left[e_1, He_1, \ldots, H^{n-1}e_1\right].$$

So we have

$$K = XR,$$

where $R = \sigma\left[e_1, He_1, \ldots, H^{n-1}e_1\right]$ is upper triangular.

(2.2) $\Longrightarrow$ (2.1)

Since $X^{-1}K = R$ we have

$$X^{-1}K = X^{-1}\left[b, Ab, A^2b, \ldots, A^{n-1}b\right] = R,$$

$$\Longrightarrow \left[X^{-1}b, X^{-1}Ab, X^{-1}A^2b, \ldots, X^{-1}A^{n-1}b\right] = R.$$

Comparing both sides column by column, we establish $X^{-1}b = R_1e_1 = r_{11}e_1$.

Further $(X^{-1}AX)(X^{-1}b) = Re_2 \Rightarrow (X^{-1}AX)r_{11}e_1 = \begin{bmatrix} r_{12} \\ r_{22} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$

If R is non-singular then $r_{ii} \neq 0$, hence

$$(X^{-1}AX)e_1 = \begin{bmatrix} \frac{r_{12}}{r_{11}} \\ \frac{r_{22}}{r_{11}} \\ 0 \\ \vdots \\ 0 \end{bmatrix} =: h_{11}e_1 + h_{21}e_2, \tag{2.3}$$

where $h_{21} = \frac{r_{22}}{r_{11}} \neq 0$.

Next, from

$$(X^{-1}AX)^2X^{-1}b = Re_3,$$

we have

$$(X^{-1}AX)^2r_{11}e_1 = Re_3.$$

Multiplying both sides of (2.3), by $(X^{-1}AX)$, we get

$$(X^{-1}AX)^2e_1 = (X^{-1}AX)h_{11}e_1 + (X^{-1}AX)h_{21}e_2 = \frac{1}{r_{11}}Re_3.$$

Or equivalently

$$(X^{-1}AX)h_{21}e_2 = \frac{1}{r_{11}}Re_3 - (X^{-1}AX)h_{11}e_1.$$

Using (2.3) and the fact, that Re_3 is a linear combination of e_1, e_2, e_3, we have

$$(X^{-1}AX)e_2 = h_{12}e_1 + h_{22}e_2 + h_{23}e_3,$$

where $h_{23} = \frac{r_{33}}{r_{11}h_{21}} \neq 0$.

Proceeding in the same manner and using induction, arranging the results column by column and noticing that $(X^{-1}AX)e_1, (X^{-1}AX)e_2 \dots (X^{-1}AX)e_n$ form the columns of

$X^{-1}AX$, we finally have

$$X^{-1}AX = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2n} \\ & \ddots & \ddots & \vdots \\ 0 & 0 & h_{n-1,n-1} & h_{nn} \end{bmatrix} = H.$$

The essential uniqueness follows from the fact that of (2.2) shown in Theorem 1 and 2 or [6]. □

Chapter 3

LU Factorization.

3.1 Bjorck and Pereyra Algorithm for Solutions of Vandermonde Systems.

This section discusses the fast algorithm to compute LU factorization of Vandermonde matrix as given by Bjorck and Pereyra in [2]. Consider the Vandermonde system of equations

$$\breve{V}a = f, \tag{3.1}$$

where

$$\breve{V} = \begin{bmatrix} 1 & x_1 & \dots & x_1^n \\ 1 & x_2 & \dots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n+1} & \dots & x_{n+1}^n \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}, \quad f = \begin{bmatrix} f_1 \\ f_1 \\ \vdots \\ f_{n+1} \end{bmatrix},$$

$x_1, x_2, \dots, x_{n+1}$ are distinct points, and $a = \begin{bmatrix} a_0, a_1, \dots, a_n \end{bmatrix}^T \in \mathbb{R}^{(n+1)}$ is an unknown vector. Such a system of equations arises in a variety of applications. One of them is classical interpolation problem.

Given a function $f(x)$ and a set of distinct points $\{x_1, x_2, \dots, x_{n+1}\}$, determine a polynomial $P(z) = a_0 + a_1 z + \ldots + a_n z^n$ with degree at most n, such that $P(x_i) = f(x_i) =: f_i$, for $i = 1....n+1$. The coefficients of $P(x)$ can be determined by solving Vandermonde matrix system given by (3.1)

Another way to compute the coefficient vector a of $P(z)$ is to use Newton's divided difference method and the Horner's like formula as given in [8].

Introduce the polynomials

$$q_0(z) = 1, \qquad q_k(z) = \prod_{i=1}^{k}(z - x_i), \qquad k = 1, \ldots n.$$

$P(z)$ can be expressed as

$$P(z) = [q_0(z), q_1(z), \ldots, q_n(z)]\, c, \quad c = [c_0, c_1, \ldots, c_n]^T,$$

where $c_0 = f(x_1)$, and

$$c_k = f[x_1, x_1, \ldots, x_{k+1}] = \frac{f[x_1, x_2, \ldots, x_k] - f[x_2, x_3, \ldots, x_{k+1}]}{x_1 - x_{k+1}}, \quad k = 1, 2, \ldots, n,$$

with $f\left[x_j\right] = f(x_j), \quad j = 1, 2, \ldots, n+1$. The coefficient vector c can be calculated by the following recurrence, based on Newton's divided difference scheme which is given in Algorithm (1),

with $c = c^{(n)}$.

The coefficient vector a can be determined based on the following relations. $P(z) = P_0(z)$, where $P_0(z)$ is derived by the following recurrence

$$P_n(z) = c_n, \quad P_k(z) = (z - x_{k+1})P_{k+1} + c_k, \qquad k = n-1, \ldots, 1, 0 \tag{3.2}$$

Algorithm 1

```
Set $c^{(0)} = [f(x_1), f(x_2), \dots, f(x_{n+1})]^T =: \left[c_0^{(0)}, c_1^{(0)}, \dots, c_n^{(0)}\right]$
For $k = 0, 1, \dots, n-1$
Compute $c^{(k+1)} = \left[c_0^{(k+1)}, c_1^{(k+1)}, \dots, c_n^{(k+1)}\right]$ with the formula
    $c_j^{(k+1)} = \begin{cases} \frac{c_j^{(k)} - c_{j-1}^{(k)}}{x_{j+1} - x_{j-k}} & j > k \\ c_j^{(k)} & j \le k \end{cases}$
End
```

Denote

$$P_k(z) = a_k^{(k)} + a_{k+1}^{(k)} z + \cdots + a_n^{(k)} z^{n-k},$$

and $a^{(k)} = \left[c_0, \dots, c_{k-1}, a_k^{(k)}, \dots, a_n^{(k)}\right]^T, \quad k = 0, 1, \dots, n$, where $a_n^{(n)} = c_n$. Then $a^{(n)} = c, \quad a^{(0)} = a$.

From (3.2), we have the iteration given below in Algorithm 2.

The detailed description of this method can be found in [2]. An error analysis is given

Algorithm 2

```
Set $a_n = c := \left[a_0^{(n)}, a_1^{(n)}, \dots, a_n^{(n)}\right]$
For $k = n-1, \dots, 1, 0$
Compute $a^{(k)} = \left[a_0^{(k)}, a_1^{(k)}, \dots, a_n^{(k)}\right]^T$ with the formula
    $a_j^{(k)} = \begin{cases} a_j^{(k+1)} & j \le k-1 \& j = n \\ a_j^{(k+1)} - x_k a_{j+1}^{(k+1)} & k \le j \le n-1 \end{cases}$
End
```

in [8].

The recurrence for c_k and a_k can be formulated in matrix vector forms. The above algorithm also gives a procedure to obtain LU factorization of $\breve{V}$.

Define the lower bidiagonal matrix $\Phi_k(\alpha) \in \mathbb{R}^{(n+1)\times(n+1)}$ by

$$\Phi_k(\alpha) = \left[\begin{array}{c|cccc} I_k & & 0 & & \\ \hline & 1 & 0 & \dots & 0 \\ & -\alpha & 1 & \dots & 0 \\ 0 & \vdots & \ddots & \ddots & \vdots \\ & 0 & \dots & -\alpha & 1 \end{array}\right]$$

Further define $M_k = \Phi_k(1)$, $\quad D_k = \operatorname{diag}(1,\dots,1,x_{k+2}-x_1,\dots,x_{n+1}-x_{n-k})$, $\quad N_k = \Phi_k(x_{k+1})^T$. The vectors $a = a^{(0)}$, $\quad c = c^{(n)}$ can be computed recursively by

$$c^{(k+1)} = D_k^{-1} M_k c^{(k)}, \qquad k = 0,1,\dots,n-1,$$

and

$$a^{(k)} = N_k a_k^{(k+1)}, \quad k = n-1,\dots,1,0.$$

Since $c^{(0)} = f$, one has $c = c^n := L^{-1} f$ where L^{-1} is lower triangular matrix defined by

$$L^{-1} = D_{n-1}^{-1} M_{n-1} \dots D_0^{-1} M_0 = D_{n-1}^{-1} \Phi_{n-1}(1) \dots D_0^{-1} \Phi_0(1).$$

Similarly, since $a^{(n)} = c$, one has $a = U^{-1} c$, where U^{-1} is unit upper triangular matrix defined by

$$U^{-1} = N_0 N_1 \dots N_{n-1} = \Phi_0(x_1)^T \dots \Phi_{n-1}(x_n)^T$$

Thus $a = U^{-1} L^{-1} f$. Since U^{-1} and L^{-1} are independent of f, comparing $a = \breve{V}^{-1} f$ with

$a = U^{-1}L^{-1}f$ we have $\check{V}^{-1} = U^{-1}L^{-1}$ or equivalently $\check{V} = LU$.

Therefore, the above Newton's divided difference approach actually computes, implicitly an LU factorization of $\check{V}$. Note that commonly we call $\check{V} = LU$ an LU factorization if L is unit lower triangular and U is upper triangular. Here is L is lower triangular and U is unit upper triangular.

3.2 Illustration of Fast LU Factorization.

The LU factorization procedure is similar to the Gaussian elimination method. It can also be described with matrix operations. Let us illustrate the above process with a 3×3 Vandermonde matrix.

Let

$$\check{V} = \begin{bmatrix} 1 & x_1 & {x_1}^2 \\ 1 & x_2 & {x_2}^2 \\ 1 & x_3 & {x_3}^2 \end{bmatrix}.$$

Using

$${D_0}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{x_2-x_1} & 0 \\ 0 & 0 & \frac{1}{x_3-x_2} \end{bmatrix}, \quad M_0 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}, \quad N_0 = \begin{bmatrix} 1 & -x_1 & 0 \\ 0 & 1 & -x_1 \\ 0 & 0 & 1 \end{bmatrix},$$

we have

$$M_0\check{V} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & x_1 & {x_1}^2 \\ 1 & x_2 & {x_2}^2 \\ 1 & x_3 & {x_3}^2 \end{bmatrix},$$

$$= \begin{bmatrix} 1 & x_1 & {x_1}^2 \\ 0 & x_2 - x_1 & (x_2 - x_1)(x_2 + x_1) \\ 0 & x_3 - x_2 & (x_3 - x_2)(x_3 + x_2) \end{bmatrix},$$

$$D_0^{-1}(M_0\check{V}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{x_2 - x_1} & 0 \\ 0 & 0 & \frac{1}{x_3 - x_2} \end{bmatrix} \begin{bmatrix} 1 & x_1 & {x_1}^2 \\ 0 & x_2 - x_1 & (x_2 - x_1)(x_2 + x_1) \\ 0 & x_3 - x_2 & (x_3 - x_2)(x_3 + x_2) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & x_1 & {x_1}^2 \\ 0 & 1 & x_2 + x_1 \\ 0 & 1 & x_3 + x_2 \end{bmatrix},$$

$$(D_0^{-1}M_0\check{V})N_0 = \begin{bmatrix} 1 & x_1 & {x_1}^2 \\ 0 & 1 & x_2 + x_1 \\ 0 & 1 & x_3 + x_2 \end{bmatrix} \begin{bmatrix} 1 & -x_1 & 0 \\ 0 & 1 & -x_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & x_2 \\ 0 & 1 & x_3 + x_2 - x_1 \end{bmatrix}.$$

In the next step

$$D_1^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{x_3 - x_1} \end{bmatrix}, \quad M_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}, \quad N_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -x_2 \\ 0 & 0 & 1 \end{bmatrix}.$$

The computations are as follows

$$M_1({D_0}^{-1}M_0\check{V}N_0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & x_2 \\ 0 & 1 & x_3 + x_2 - x_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & x_2 \\ 0 & 0 & x_3 - x_1 \end{bmatrix},$$

$$D_1^{-1}(M_1D_0{}^{-1}M_0\check{V}N_0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{x_3-x_1} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & x_2 \\ 0 & 0 & x_3-x_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & x_2 \\ 0 & 0 & 1 \end{bmatrix},$$

$$(D_1^{-1}M_1D_0{}^{-1}M_0\check{V}N_0N_1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & x_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -x_2 \\ 0 & 0 & 1 \end{bmatrix} = I_3.$$

In this example $n = 2$. So $n-1 = 1$. Hence for $L = M_0^{-1}D_0M_1^{-1}D_1, \quad U = N_1^{-1}N_0^{-1}$,

$$\check{V} = LU.$$

The procedure is the same for a general $(n+1)\times(n+1)$ Vandermonde matrix $\check{V}$. Proceeding in the same manner as above, post-multiplying by $N_0N_1\dots N_{n-1}$ and pre-multiplying by $D_{n-1}^{-1}M_{n-1}\dots D_0^{-1}M_0$ we get

$$D_{n-1}^{-1}M_{n-1}\dots D_0^{-1}M_0\check{V}N_0N_1\dots N_{n-1} = I.$$

Set $L^{-1} = D_{n-1}^{-1}M_{n-1}\dots D_0^{-1}M_0$ and $U^{-1} = N_0N_1\dots N_{n-1}$, or equivalently

$$L = M_0^{-1}D_0\dots M_{n-1}^{-1}D_{n-1} = \Phi_0(-1)D_0\dots\Phi_{n-1}(-1)D_{n-1}, \tag{3.3}$$

$$U = N_{n-1}^{-1}\dots N_1^{-1}N_0^{-1} = \Phi_{n-1}(-x_n)^T\dots\Phi_1(-x_2)^T\Phi_0(-x_1)^T. \tag{3.4}$$

Then

$$L^{-1}\breve{V}U^{-1} = I, \quad \Rightarrow \quad \breve{V} = LU,$$

where L is lower triangular and U is unit upper triangular.

If we compute such an LU factorization with L and U in product form, then only the diagonal elements of D_k need to be computed, which needs $\frac{n(n+1)}{2}$ flops. If L and U need to be explicitly formed then the cost will be $O(n^3)$ flops.

3.3 A Connection with Krylov Matrices

In this section we interpret the LU factorization of Vandermonde matrix by using basic properties of Krylov matrices. Let

$$\Lambda = \begin{bmatrix} x_1 & 0 & \dots & 0 \\ 0 & x_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & x_{n+1} \end{bmatrix}, \quad e = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

where $\{x_i\}_{i=1}^{n+1}$ is a set of distinct nodes. Then for L defined in (3.3), it is easily verified, that

$$L^{-1}\Lambda L = \begin{bmatrix} x_1 & 0 & \dots & \dots & 0 \\ 1 & x_2 & \dots & \dots & 0 \\ 0 & 1 & x_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 1 & x_{n+1} \end{bmatrix} =: H, \quad L^{-1}e = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = e_1.$$

Notice that H is a special type of upper Hessenberg matrix, although it is more appropriate to call it lower bidiagonal.

The Vandermonde matrix V̌ can be written as a Krylov matrix

$$\check{V} = \left[e, \Lambda e, \Lambda^2 e, \ldots, \Lambda^n e\right].$$

Using $\Lambda = LHL^{-1}$ and $e = Le_1$,

$$\check{V} = \left[e, LHL^{-1}e, LH^2L^{-1}e, \ldots, LH^nL^{-1}e\right]$$

$$= L\left[e_1, He_1, \ldots, H^n e_1\right]$$

$$=: LU.$$

Clearly L is lower triangular and it is easily verified that U is unit upper triangular. By theorem 2, the LU factorization is just the same as that derived in the previous section. The derivation examines the result of Theorem 3 with Vandermonde matrices. That is, the LU factorization of V̌ is equivalent to the upper Hessenberg reduction.

Chapter 4

QR Factorization of Vandermonde Matrices and Least Squares Polynomial Approximation.

4.1 Overview.

This section summarises the technique used by Reichel [11] to compute fast QR factorization of Vandermonde like matrices in order to determine polynomials with least squares approximation. Let $\{x_k\}_{k=1}^{n+1}$ be a set of distinct nodes on the real axis. For $\mathbb{P}_n$, the inner product is defined as follows $\langle f,g\rangle_w := \sum f(x_k)g(x_k)w_k^2$, $\forall\ f,g \in \mathbb{P}_n$, where $\{w_k\}_{k=1}^{n+1}$ is a set of positive real weights. The least squares problem is proposed as follows. Given a function f and a polynomial basis $p_0(x),\ldots,p_m(x)$ for $\mathbb{P}_m,(m \leq n)$. Determine

$$p(x) = c_0p_0(x) + c_1p_1(x) + \cdots + c_mp_m(x) \quad \in \mathbb{P}_m,$$

such that $p(x)$ minimizes $\langle f-q, f-q\rangle_w$ over $q \in \mathbb{P}_m$. This is equivalent to the least squares problem:

Determine $c = [c_0, c_1, \ldots, c_m]^T$ to minimize

$$\|Vc - f\|_w = \|W(Vc - f)\|_2, \quad \forall c \in \mathbb{R}^{m+1}, \tag{4.1}$$

where

$$V = \left[p_{j-1}(x_i)\right] = \begin{bmatrix} p_0(x_1) & p_1(x_1) & \dots & p_m(x_1) \\ p_0(x_2) & p_1(x_2) & \ddots & p_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ p_0(x_{n+1}) & p_1(x_{n+1}) & \dots & p_m(x_{n+1}) \end{bmatrix} \in \mathbb{R}^{(n+1)\times(m+1)},$$

$$f = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_{n+1}) \end{bmatrix}, \quad W = \begin{bmatrix} w_1 & 0 & \dots & 0 \\ \vdots & w_2 & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_{n+1} \end{bmatrix}.$$

If $WV = \tilde{Q}R, \quad d = \tilde{Q}^T W f$, where $\tilde{Q} \in \mathbb{R}^{(n+1)\times(m+1)}$ is orthonormal, $R \in \mathbb{R}^{(m+1)\times(m+1)}$ is upper triangular, then $c = R^{-1}d$. So the main task is to compute the reduced QR factorization $WV = \tilde{Q}R$.

Reichel proposed the following way to compute the QR factorization. First of all an orthogonal matrix Q is computed, such that

$$Q = \left[\tilde{Q}, \hat{Q}\right] \in \mathbb{R}^{(n+1)\times(n+1)}$$

in the following way. Let $\{\pi_j\}_{j=0}^{n}$ be an orthonormal basis for $\mathbb{P}_n$ with respect to the inner product defined above, with $\deg \pi_j = j$ and positive leading coefficient. The polynomials π_j satisfy a three term recurrence relation given below

$$\beta_0 \pi_0(x) = 1,$$

$$\beta_1 \pi_1(x) = (x - \alpha_1)\pi_0(x),$$

$$\beta_j \pi_j(x) = (x-\alpha_j)\pi_{j-1}(x) - \beta_{j-1}\pi_{j-2}(x), \quad j = 2,\ldots,n. \tag{4.2}$$

The coefficients α_j and $\beta_j > 0$ satisfy the following conditions.

$$\beta_0 = \langle 1,1\rangle_w^{\frac{1}{2}}, \qquad \beta_1 = (\langle x\pi_0, x\pi_0\rangle_w - \alpha_1^2)^{\frac{1}{2}},$$

$$\alpha_j = \langle x\pi_{j-1}, \pi_{j-1}\rangle_w, \quad j = 1,2,\ldots,n+1,$$

$$\beta_j = (\langle x\pi_{j-1}, x\pi_{j-1}\rangle_w - \alpha_j^2 - \beta_{j-1}^2)^{1/2}, \qquad j = 2,3,\ldots,n.$$

Instead of using the formulas directly, Reichel's method is to compute scalars α_j, β_j therefore also $\pi_0, \pi_1 \ldots, \pi_n$, and the orthogonal matrix Q, based on tridiagonal reduction. Let

$$\Lambda = \begin{bmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_{n+1} \end{bmatrix}. \tag{4.3}$$

A series of careful Givens rotations is applied to Λ to determine T, a unique, symmetric tridiagonal matrix with diagonal and subdiagonal elements consisting of coefficients α_j and β_j. The Givens rotations form Q, although Q is not an explicit output of the proposed method. The compute matrix Q and T satisfy

$$Q^T \Lambda Q = T = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_n \\ & & \beta_n & \alpha_{n+1} \end{bmatrix}, \tag{4.4}$$

$$Qe_1 = \frac{1}{\|w\|_2} w, \tag{4.5}$$

where $w = \begin{bmatrix} w_1, w_2, \ldots, w_{n+1} \end{bmatrix}^T$. We illustrate the process with $n = 2$.
Let

$$\Lambda = \begin{bmatrix} x_1 & & \\ & x_2 & \\ & & x_3 \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}.$$

We have defined the Givens rotation in chapter 0. For an arbitrary $\alpha_0 \in \mathbb{R}$ we set

$$T_2' = T_2'' = \begin{bmatrix} \alpha_0 & w_1 \\ w_1 & x_1 \end{bmatrix}.$$

We then add the next node x_2 and weight w_2 to form

$$T_3' = \begin{bmatrix} \alpha_0 & w_1 & w_2 \\ w_1 & x_1 & 0 \\ w_2 & 0 & x_2 \end{bmatrix}.$$

Now we perform a similarity transformation on T_3' with a Givens rotation acting on last two rows and columns to annihilate w_2 to get

$$T_3'' = \begin{bmatrix} \alpha_0 & \hat{\beta}_0 & 0 \\ \hat{\beta}_0 & \hat{\alpha}_1 & \hat{\beta}_1 \\ 0 & \hat{\beta}_1 & \hat{\alpha}_2 \end{bmatrix}.$$

Next, we add x_3 and w_3 to T_3'' to form

$$T_4' = \begin{bmatrix} \alpha_0 & \hat{\beta}_0 & 0 & w_3 \\ \hat{\beta}_0 & \hat{\alpha}_1 & \hat{\beta}_1 & 0 \\ 0 & \hat{\beta}_1 & \hat{\alpha}_2 & 0 \\ w_3 & 0 & 0 & x_3 \end{bmatrix}.$$

We perform another similarity transformation with a Givens rotation on the 2nd and 4th rows and columns to annihilate w_3 and to get

$$T_4'' = \left[\begin{array}{c|ccc} \alpha_0 & \beta_0 & 0 & 0 \\ \hline \beta_0 & \tilde{\alpha}_1 & \tilde{\beta}_1 & \gamma_1 \\ 0 & \tilde{\beta}_1 & \tilde{\alpha}_2 & \tilde{\beta}_2 \\ 0 & \gamma_1 & \tilde{\beta}_2 & \tilde{\alpha}_3 \end{array}\right].$$

We then perform one more similarity transformation with a Givens rotation on the last two rows and columns to annihilate γ_1 and get

$$T_4''' = \left[\begin{array}{c|ccc} \alpha_0 & \beta_0 & 0 & 0 \\ \hline \beta_0 & \alpha_1 & \beta_1 & 0 \\ 0 & \beta_1 & \alpha_2 & \beta_2 \\ 0 & 0 & \beta_2 & \alpha_3 \end{array}\right].$$

Hence the required tridiagonal matrix is given by

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & 0 \\ \beta_1 & \alpha_2 & \beta_2 \\ 0 & \beta_2 & \alpha_3 \end{bmatrix}.$$

The orthogonal matrix Q is the product of the Givens rotations by deleting the first row and column. The procedure may continue for $n > 2$, by adding x_i and w_i each time and reducing the extended matrix to tridiagonal until $i = n+1$. The matrices T and Q are obtained in the same way as described above. On the other hand, the three-term recurrence in (4.2) shows that the polynomials π_j satisfy

$$x\left[\pi_0(x), \pi_1(x), \ldots, \pi_n(x)\right] = \left[\pi_0(x), \pi_1(x), \ldots, \pi_n(x)\right] T + \pi_{n+1}(x) e_{n+1}^T, \tag{4.6}$$

where $\pi_{n+1}(x) = (x - \alpha_{n+1})\pi_n(x) - \beta_n \pi_{n-1}(x)$.

By setting $x = x_1, x_2, \ldots, x_{n+1}$, respectively in (4.6) we have

$$\Lambda\Pi = \Pi T + \mathsf{P}_{n+1} e_{n+1}^T,$$

where

$$\Pi = \begin{bmatrix} \pi_0(x_1) & \pi_1(x_1) & \ldots & \pi_n(x_1) \\ \pi_0(x_2) & \pi_1(x_2) & \ldots & \pi_n(x_2) \\ \vdots & \vdots & \ldots & \vdots \\ \pi_0(x_{n+1}) & \pi_1(x_{n+1}) & \ldots & \pi_n(x_{n+1}) \end{bmatrix}, \quad \mathsf{P}_{n+1} = \begin{bmatrix} \pi_{n+1}(x_1) \\ \vdots \\ \pi_{n+1}(x_{n+1}) \end{bmatrix}.$$

Premultiplying by

$$W = \begin{bmatrix} w_1 & 0 & \ldots & 0 \\ \vdots & w_2 & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & w_{n+1} \end{bmatrix},$$

and using $W\Lambda = \Lambda W$ we get

$$\Lambda W\Pi = W\Pi T + W\mathsf{P}_{n+1}e_{n+1}^T. \tag{4.7}$$

Note that $\langle \pi_i(x), \pi_j(x)\rangle_w = \delta_{ij}, \quad i,j = 0,1,\dots,n$, and since we have $\langle \pi_{n+1}(x), \pi_j(x)\rangle_w = 0, \quad j = 0,1,\dots,n$, in matrix form these imply

$$(W\Pi)^T(W\Pi) = I_{n+1}, \quad (W\Pi)^T\mathsf{P}_{n+1} = 0.$$

By premultiplying $(W\Pi)^T$ to (4.7)

$$(W\Pi)^T\Lambda(W\Pi) = T,$$

and

$$(W\Pi)e_1 = \frac{1}{\|w\|_2}w = \frac{1}{\beta_0}w, \quad w = \begin{bmatrix} w_1 \\ \vdots \\ w_{n+1} \end{bmatrix}.$$

Since $W\Pi$ is orthogonal, and $\beta_j > 0$ by Theorem 3, we have

$$Q = W\Pi = \begin{bmatrix} w_1 & 0 & \dots & 0 \\ \vdots & w_2 & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_{n+1} \end{bmatrix} \begin{bmatrix} \pi_0(x_1) & \pi_1(x_1) & \dots & \pi_n(x_1) \\ \pi_0(x_2) & \pi_1(x_2) & \dots & \pi_n(x_2) \\ \vdots & \vdots & \dots & \vdots \\ \pi_0(x_{n+1}) & \pi_1(x_{n+1}) & \dots & \pi_n(x_{n+1}) \end{bmatrix}. \tag{4.8}$$

Next, a formula is derived for computing R, the R factor of V. Because deg $p_k(x) = k$ for $k = 0,\ldots,m$ and deg $\pi_k(x) = k$ for $k = 0,\ldots,n$, we have the expression

$$p_k(x) = \sum_{j=0}^{k} r_{j+1,k+1}\pi_j(x), \quad k = 0,1,2,\ldots,m, \tag{4.9}$$

or

$$\begin{aligned} p_0(x) &= r_{11}\pi_0(x), \\ p_1(x) &= r_{12}\pi_0(x) + r_{22}\pi_1(x), \\ &\vdots \\ p_m(x) &= r_{1,m+1}\pi_0(x) + r_{2,m+1}\pi_1(x) + \ldots + r_{m+1,m+1}\pi_m(x), \end{aligned}$$

or

$$[p_0(x), p_1(x),\ldots, p_m(x)] = [\pi_0(x), \pi_1(x),\ldots \pi_n(x)] \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1,m+1} \\ 0 & r_{22} & r_{23} & \cdots & r_{2,m+1} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & r_{mm} & r_{m,m+1} \\ 0 & 0 & 0 & 0 & r_{m+1,m+1} \end{bmatrix}.$$

Substituting $x = x_1, x_2, \ldots, x_{n+1}$, respectively,

$$V = \begin{bmatrix} p_0(x_1) & p_1(x_1) & \ldots & p_m(x_1) \\ p_0(x_2) & p_1(x_2) & \ldots & p_m(x_2) \\ \vdots & \vdots & \ldots & \vdots \\ p_0(x_{n+1}) & p_1(x_{n+1}) & \ldots & p_m(x_{n+1}) \end{bmatrix} = \begin{bmatrix} \pi_0(x_1) & \ldots & \pi_n(x_1) \\ \pi_0(x_2) & \ldots & \pi_n(x_2) \\ \vdots & \ldots & \vdots \\ \pi_0(x_{n+1}) & \ldots & \pi_n(x_{n+1}) \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix}$$

$$= \Pi \begin{bmatrix} R \\ 0 \end{bmatrix}.$$

Premultiply by W,

$$W \begin{bmatrix} p_0(x_1) & p_1(x_1) & \ldots & p_n(x_1) \\ p_0(x_2) & p_1(x_2) & \ldots & p_n(x_2) \\ \vdots & \vdots & \ldots & \vdots \\ p_0(x_{n+1}) & p_1(x_{n+1}) & \ldots & p_n(x_{n+1}) \end{bmatrix} = W\Pi \begin{bmatrix} R \\ 0 \end{bmatrix} = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = \tilde{Q}R.$$

So

$$WV = \tilde{Q}R, \tag{4.10}$$

which is the required reduced QR factorization of WV.

If we use (4.9) to compute R, we need to generate the polynomials $\pi_j(x)$ explicitly. The following approach avoids this problem.

Since $\deg p_j = j$ therefore $\deg(xp_j) = j+1, \quad j = 0, \ldots, m.$

We have the expression

$$xp_{j-1}(x) = h_{1j}p_0(x) + h_{2j}p_1(x) + \ldots + h_{j+1,j}p_j(x), \quad j = 1.....m+1,$$

where

$$h_{m+2,m+1}p_{m+1}(x) = xp_m(x) - (h_{1j}p_0(x) + h_{2j}p_1(x) + \ldots + h_{m+1,m+1}p_m(x)).$$

Define

$$H = \begin{bmatrix} h_{11} & h_{12} & \ldots & h_{1,m+1} \\ h_{21} & h_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \ldots & h_{m+1,m} & h_{m+1,m+1} \end{bmatrix}.$$

Then

$$x[p_0(x), p_1(x), \ldots, p_n(x)] = [p_0(x), p_1(x), \ldots, p_m(x)]H + h_{m+2,m+1}p_{m+1}(x)e_{m+1}^T. \tag{4.11}$$

Taking $x = x_1, x_2, \ldots, x_{n+1}$, we get

$$\Lambda V = VH + \grave{t}e_{m+1}^T,$$

where $\grave{t} = h_{m+2,m+1}\begin{bmatrix} p_{m+1}(x_1) \\ p_{m+1}(x_2) \\ \vdots \\ p_{m+1}(x_{m+1}) \end{bmatrix}.$

Premultiplying by W, we get

$$\Lambda WV = WVH + \tilde{t}e_{m+1}^T, \quad \tilde{t} = W\grave{t}. \tag{4.12}$$

Using $\Lambda = QTQ^T$, and $Q^TWV = \begin{bmatrix} R \\ 0 \end{bmatrix}$, we have

$$QTQ^TWV = WVH + \tilde{t}e_{m+1}^T$$

$$\Rightarrow TQ^TWV = Q^TWVH + Q^T\tilde{t}e_{m+1}^T,$$

i.e.

$$T\begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix} H + te_{m+1}^T, \quad t = Q^T\tilde{t}. \tag{4.13}$$

Let $t = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$, $\quad t_1 \in \mathbb{R}^{m+1}$ and $T_m = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \beta_m & \alpha_{m+1} \end{bmatrix}$. Then by comparing the

top $(m+1) \times (m+1)$ matrices in (4.13) on both sides, we have

$$T_mR = RH + t_1e_{m+1}^T. \tag{4.14}$$

Let

$$R = [r_1, r_2, \ldots, r_{m+1}].$$

By comparing the columns on both sides in the above equation, r_i can be computed by the following recurrence

$$r_1 = \frac{p_0(x)}{\pi_0(x)}e_1, \quad r_{j+1} = \frac{1}{h_{j+1,j}}(T_mr_j - h_{1j}r_1 - \ldots - h_{jj}r_j), \quad j = 1, \ldots, m.$$

Note that $p_0(x)$ and $\pi_0(x)$ are constants. Further note that t_1 has no influence on the computation of R.

Further, R^{-1} satisfies

$$HR^{-1} = R^{-1}T_m - \frac{1}{r_{m+1,m+1}} R^{-1} t_1 e_{m+1}^T.$$

Similarly, let

$$R^{-1} = \left[r'_1, \ldots, r'_{n+1}\right]$$

Then

$$r'_1 = \frac{\pi_0(x)}{p_0(x)} e_1, \quad r'_{j+1} = \frac{1}{\beta_j}(Hr'_j - \alpha_j r'_j - \beta_{j-1} r'_{j-1}), \quad j = 1, \ldots, m.$$

In order to solve the least squares problem, one needs to compute $d = \tilde{Q}^T f$. This can be done during the tridiagonal reduction by premultiplying the sequence of Givens rotations to f. Once d is determined, we have $c = R^{-1}d$, where R^{-1} can be computed by above recurrence. The cost of computations of reduced QR factorization or solving least squares problem is $O(n^3)$.

4.2 QR Factorization for Three-term Recurrence Polynomials

In [11], Reichel only considers the case when the polynomials $p_0(x), \ldots, p_n(x)$ satisfy the following three-term recurrence relation.

$$b_0 p_0(x) = 1,$$

$$b_1 p_1(x) = (x - a_1) p_0(x), \tag{4.15}$$

$$b_j p_j(x) = (x - a_j) p_{j-1}(x) - b_{j-1} p_{j-2}(x),$$

$j = 2, 3....., m$, where a_j and $b_j > 0 \in \mathbb{R}$.

From (4.15) we have

$$x p_0(x) = b_1 p_1 + a_1 p_0(x),$$

$$x p_{j-1}(x) = b_{j-1} p_{j-2}(x) + a_j p_{j-1}(x) + b_j p_j(x) \quad j = 2, \ldots, m,$$

or in matrix form

$$x\left[p_0(x), p_1(x), \ldots, p_m(x)\right] = \left[p_0(x), p_1(x), \ldots, p_m(x)\right] \begin{bmatrix} a_1 & b_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_m \\ & & b_m & a_{m+1} \end{bmatrix} \tag{4.16}$$

$$+ b_{m+1} p_{m+1}(x) e_{m+1}^T,$$

where p_{m+1} and b_{m+1} satisfy

$$b_{m+1} p_{m+1}(x) = x p_m(x) - a_{m+1} p_{m-1}(x) - b_m p_{m-1}(x).$$

This is the special case of (4.11). So we have (4.14) with

$$H = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_n \\ & & b_n & a_{n+1} \end{bmatrix} =: \hat{T}, \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = b_{m+1} Q^T W \begin{bmatrix} p_{m+1}(x_1) \\ p_{m+1}(x_2) \\ \vdots \\ p_{m+1}(x_{n+1}) \end{bmatrix}. \tag{4.17}$$

Partition $R = [r_1, \dots, r_{m+1}]$. Using $T_m R = R\hat{T} + t_1 e_{m+1}^T$, the columns r_j can be computed by simpler recurrence

$$r_1 = \frac{p_0(x)}{\pi_0(x)} e_1, \quad r_{j+1} = (T_m r_j - b_{j-1} r_{j-1} - a_j r_j)/b_j, \quad j = 1, \dots, m.$$

Similarly, R^{-1} can be computed from the relation:

$$\hat{T} R^{-1} = R^{-1} T_m - R^{-1} t_1 e_{m+1}^T / r_{m+1,m+1}.$$

Partition R^{-1} in columns

$$R^{-1} = [\tilde{r}_1, \dots, \tilde{r}_{n+1}].$$

The columns $\tilde{r}_j$ can be computed by

$$\tilde{r}_1 = \frac{\pi_0(x)}{p_0(x)} e_1, \quad \tilde{r}_{j+1} = (\hat{T} \tilde{r}_j - \beta_{j-1} \tilde{r}_{j-1} - \alpha_j \tilde{r}_j)/\beta_j, \quad j = 1, \dots, m.$$

Reichel's GR algorithm for solving least squares approximation problem (4.1) is summarized in the following algorithm. For polynomials satisfying a three term recurrence, the GR algorithm is given as follows :

1. Compute Q such that $Q^T \Lambda Q = T, \quad Qe_1 = \frac{1}{\|w\|_2} w$.
2. Compute $d = Q^T W f = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}, \quad d_1 \in \mathbb{R}^{m+1}$.
3. Compute R^{-1}, based on the relation

$$\hat{T} R^{-1} = R^{-1} T_m - R^{-1} t_1 e_{m+1}^T / r_{m+1,m+1}.$$

4. Compute $c = R^{-1} d_1$.

The tridiagonal structure of $\hat{T}$, attributed to use of three term recurrence polynomials, reduces the cost to $O(n^2)$ flops. That is the advantage of using three-term recurrence. A MATLAB code of this algorithm is provided in Appendix A.

4.3 Classical Vandermonde Case.

As pointed out in [11], the fast QR factorization approach also works when the polynomials are $p_j(x) = x^j, \quad j = 0, 1, \ldots, m$. In this case, the polynomials satisfy simple recurrence $p_{j+1}(x) = xp_j(x), \quad j = 0, 1, \ldots, m-1$, or equivalently

$$x \begin{bmatrix} 1 & x & \ldots & x^m \end{bmatrix} = \begin{bmatrix} 1 & x & \ldots & x^m \end{bmatrix} \tilde{H} + x^{m+1} e_{m+1}^T,$$

where

$$\tilde{H} = \begin{bmatrix} 0 & 0 & \ldots & 0 & 0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ & \ddots & \ddots & 0 & 0 \\ & & 0 & 1 & 0 \end{bmatrix}.$$

So

$$\Lambda W \breve{V} = W \breve{V} \tilde{H} + \breve{t} e_{m+1}^T,$$

$$\breve{V} = \begin{bmatrix} 1 & x_1 & \ldots & x_1^m \\ 1 & x_2 & \ldots & x_2^m \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n+1} & \ldots & x_{n+1}^m \end{bmatrix}, \text{ where } \breve{t} = W \begin{bmatrix} x_1^{m+1} \\ x_2^{m+1} \\ \vdots \\ x_{m+1}^{n+1} \end{bmatrix}.$$

Proceeding in the same manner as in previous sections we have

$$T_m\breve{R} = \breve{R}\tilde{H} + t_1 e_{m+1}^T, \tag{4.18}$$

which is parallel to what was obtained in (4.14) in the previous section. Hence $\breve{R} = [\breve{r}_1, \dots, \breve{r}_{m+1}]$ can be computed column by column by

$$\breve{r}_1 = \frac{1}{\pi_0(x)} e_1, \quad \breve{r}_{j+1} = T_m \breve{r}_j, \quad j = 1, \dots, m.$$

Similarly,

$$\breve{R}^{-1} = [\tilde{r}_1, \dots, \tilde{r}_{m+1}]$$

satisfies

$$\tilde{H}\breve{R}^{-1} = \breve{R}^{-1}T_m - \breve{R}^{-1}t_1 e_{m+1}^T / \breve{r}_{m+1,m+1}.$$

The columns $\tilde{r}_j$ can be computed by

$$\tilde{r}_1 = \pi_0(x)e_1, \quad \tilde{r}_{j+1} = (\tilde{H}\tilde{r}_j - \beta_{j-1}\tilde{r}_{j-1} - \alpha_j \tilde{r}_j)/\beta_j, \quad j = 1, \dots, m.$$

Note (4.18) also gives an interesting relation between T_m and its companion matrix. Premultiplying $\breve{R}^{-1}$ to (4.18) we get

$$\breve{R}^{-1}T_m\breve{R} = \tilde{H} + \breve{R}^{-1}t_1 e_{m+1}^T = \begin{bmatrix} 0 & 0 & \dots & -c_{m+1} \\ 1 & 0 & \dots & \vdots \\ & \ddots & \ddots & \vdots \\ & & 1 & -c_1 \end{bmatrix} =: C,$$

where

$$\begin{bmatrix} c_{m+1} \\ \vdots \\ c_1 \end{bmatrix} = -\breve{R}^{-1} t_1. \tag{4.19}$$

The matrix C is the companion matrix of T_m, since C and T_m are similar. The characteristic polynomial of T_m is

$$\det(\lambda I - T_m) = \det(\lambda I - C) = \lambda^{m+1} + c_1 \lambda^m + c_2 \lambda^{m-1} + \ldots + c_m \lambda + c_{m+1}.$$

When $n = m$, we have the relation for T,

$$\breve{R}^{-1} T \breve{R} = \tilde{H} + \breve{R}^{-1} t e_{n+1}^T =: C, \tag{4.20}$$

with

$$\begin{bmatrix} c_{n+1} \\ \vdots \\ c_1 \end{bmatrix} = -\breve{R}^{-1} t = -\breve{V}^{-1} \begin{bmatrix} x_1^{n+1} \\ \vdots \\ x_{n+1}^{n+1} \end{bmatrix}.$$

4.4 Krylov Matrix Connection

As in previous chapters, define

$$\Lambda = \begin{bmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_{n+1} \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n+1} \end{bmatrix}, \quad e = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

$$W = \begin{bmatrix} w_1 & 0 & \dots & 0 \\ \vdots & w_2 & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_{n+1} \end{bmatrix}.$$

Then $\breve{V}$ can be written as

$$\breve{V} = \left[e, \Lambda e, \Lambda^2 e, \dots, \Lambda^n e\right].$$

Using (4.4) and (4.5), i.e., $Q^T \Lambda Q = T$ and $Qe_1 = \frac{1}{\|w\|_2} w$, and by Theorem 3,

$$W\breve{V} = Q\breve{R}, \quad \breve{R} = \|w\|_2 \left[e_1, Te_1, T^2 e_1, \dots, T^n e_1\right].$$

Clearly, this $\breve{R}$ is identical to the one computed in previous section. By (4.8), $Q = W\Pi$. So $W\breve{V} = W\Pi\breve{R}$, which implies $\Pi = \breve{V}\breve{R}^{-1}$.

Let

$$\breve{R}^{-1} = \begin{bmatrix} \breve{r}_{11} & \dots & \dots & \breve{r}_{1,n+1} \\ & \ddots & & \vdots \\ & & & \breve{r}_{n+1,n+1} \end{bmatrix}.$$

Then

$$\pi_j(x) = \breve{r}_{1,j+1} + \breve{r}_{2,j+1} x + \dots + \breve{r}_{j+1,j+1} x^j, \quad j = 0, 1, \dots, n$$

Therefore, the elements of upper triangular matrix $\breve{R}^{-1}$ give the coefficients of $\pi_j(x)$. Now for a polynomial sequence $p_0(x), p_1(x), \dots, p_m(x)$ with $\deg p_j = j, \quad j = 0, 1, \dots, m$, we have (4.12),

$$\Lambda WV = WVH + \tilde{t} e_{m+1}^T,$$

where V and H are same as defined in Section 4.1.

For each j, $p_j(x)$ is a linear combination of $1, x_1, \ldots, x^j$. So

$$V = \breve{V} \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix},$$

where $\hat{R} \in \mathbb{R}^{(m+1)\times(m+1)}$ is upper triangular and non-singular. So

$$WV = W\breve{V} \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} = Q\breve{R} \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} = Q \begin{bmatrix} \breve{R}_m\hat{R} \\ 0 \end{bmatrix},$$

where $\breve{R}_m$ is the leading principal $(m+1)\times(m+1)$ submatrix of $\breve{R}$. Comparing it with (4.10), $R = \breve{R}\hat{R}$.

Chapter 5

Inverse of Vandermonde Matrices

5.1 Overview

This chapter takes a peek into various methods developed by a few selected authors to compute the inverse of a Vandermonde matrix, and a comparison of these methods is made for speed and accuracy. Further, a Krylov matrix connection is made with the inverse of a Vandermonde matrix.

5.2 The Traub Algorithm

The Traub algorithm as given in [12] gives a fast method to compute the inverse of a Vandermonde matrix in $O(n^2)$ flops. At the same time this algorithm is said to be numerically unstable.

The general idea of Traub algorithm is discussed as follows. Let $P(x)$ be a master polynomial whose zeros are the nodes of the Vandermonde matrix $\check{V}$. Define $P(x)$ as

$$P(x) = \textstyle\prod_{k=1}^{n+1}(x - x_k) = x^{n+1} + \displaystyle\sum_{k=0}^{n} a_k x^k. \tag{5.1}$$

for distinct $\{x_1,\ldots,x_{n+1}\}$. Consider the divided difference

$$P[t,x] = \frac{P(t)-P(x)}{t-x}. \tag{5.2}$$

Clearly, $P[t,x]$ is a polynomial in both t and x, with degree n. Define the polynomials

$$\{q_k(x)\}_{0\le k\le n} \tag{5.3}$$

that satisfy

$$P[t,x] = q_n(x) + tq_{n-1}(x) + \ldots + t^{n-1}q_1(x) + t^n q_0(x). \tag{5.4}$$

The polynomials given by (5.3) are also known as associated polynomials or Horner's polynomials. Substituting (5.1) into (5.2)

$$P[t,x] = \sum_{i=1}^{n+1} a_i \frac{t^i - x^i}{t-x} = \sum_{i=1}^{n+1} a_i(t^{i-1} + t^{i-2}x + \ldots + tx^{i-2} + x^{i-1}), \tag{5.5}$$

which implies that associated polynomials given by (5.3) can also be written as

$$q_k(x) = x^k + a_n x^{k-1} + \ldots + a_{n-k+2}x + a_{n-k+1}. \tag{5.6}$$

The polynomials above also satisfy the following recurrence relations

$$q_0(x) = 1, \qquad q_k(x) = xq_{k-1}(x) + a_{n-k+1}, \quad k = 1,2,\ldots,n, \tag{5.7}$$

and $q_{n+1}(x) = P(x)$. From (5.1)

$$P(x_j) = 0, \quad j = 1,\ldots,n+1.$$

So, by (5.2),

$$P[x_j, x_k] = \begin{cases} 0 & j \neq k \\ P'(x_k) & j = k. \end{cases}$$

Using (5.4) we have the so called orthogonality relation:

$$\frac{P[x_j, x_k]}{P'(x_k)} = \sum_{i=0}^{n} x_j^i \frac{q_{n-i}(x_k)}{P'(x_k)} = \delta_{jk} = \begin{cases} 0 & j \neq k \\ 1 & j = k. \end{cases} \tag{5.8}$$

The orthogonality relation can be used to express the inverse of Vandermonde matrix $\check{V}$ as follows

$$\delta_{jk} = \frac{P[x_j, x_k]}{P'(x_k)} = \frac{q_n(x_k)}{P'(x_k)} + x_j \frac{q_{n-1}(x_k)}{P'(x_k)} + \ldots + x_j^n \frac{q_0(x_k)}{P'(x_k)}$$

$$= \begin{bmatrix} 1 & x_j & x_j^2 & \ldots & x_j^n \end{bmatrix} \begin{bmatrix} \frac{q_n(x_k)}{P'(x_k)} \\ \frac{q_{n-2}(x_k)}{P'(x_k)} \\ \vdots \\ \frac{q_0(x_k)}{P'(x_k)} \end{bmatrix},$$

for $j, k = 1 \ldots n+1$. In matrix form

$$I = \check{V} \begin{bmatrix} \frac{q_n(x_1)}{P'(x_1)} & \frac{q_n(x_2)}{P'(x_2)} & \ldots & \frac{q_n(x_{n+1})}{P'(x_{n+1})} \\ \frac{q_{n-1}(x_1)}{P'(x_1)} & \frac{q_{n-1}(x_2)}{P'(x_2)} & \ldots & \frac{q_{n-1}(x_{n+1})}{P'(x_{n+1})} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{q_0(x_1)}{P'(x_1)} & \frac{q_0(x_2)}{P'(x_2)} & \ldots & \frac{q_0(x_{n+1})}{P'(x_{n+1})} \end{bmatrix},$$

or

$$I = \check{V} \begin{bmatrix} q_n(x_1) & q_n(x_2) & \dots & q_n(x_{n+1}) \\ q_{n-1}(x_1) & q_{n-1}(x_2) & \dots & q_{n-1}(x_{n+1}) \\ \vdots & \vdots & \vdots & \vdots \\ q_0(x_1) & q_0(x_2) & \dots & q_0(x_{n+1}) \end{bmatrix} \begin{bmatrix} \frac{1}{P'(x_1)} & & & \\ & \frac{1}{P'(x_2)} & & \\ & & \ddots & \\ & & & \frac{1}{P'(x_{n+1})} \end{bmatrix}.$$

Hence

$$\check{V}^{-1} = \begin{bmatrix} q_n(x_1) & \dots & q_n(x_{n+1}) \\ q_{n-1}(x_1) & \dots & q_{n-1}(x_{n+1}) \\ \vdots & \vdots & \vdots \\ q_0(x_1) & \dots & q_0(x_{n+1}) \end{bmatrix} \begin{bmatrix} \frac{1}{P'(x_1)} & & & \\ & \frac{1}{P'(x_2)} & & \\ & & \ddots & \\ & & & \frac{1}{P'(x_{n+1})} \end{bmatrix}. \qquad (5.9)$$

Traub used (5.9) to derive a fast algorithm for the inverse of Vandermonde matrices. The Traub algorithm is given as follows:

1. Compute the coefficients of $P(x)$ in (5.1) using the nested polynomial multiplication:

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \end{bmatrix} = \begin{bmatrix} -x_1 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} a_0^{(k)} \\ a_1^{(k)} \\ \vdots \\ a_k^{(k)} \end{bmatrix} = \begin{bmatrix} 0 \\ a_0^{(k-1)} \\ a_1^{(k-1)} \\ \vdots \\ a_{k-1}^{(k-1)} \end{bmatrix} - x_k \begin{bmatrix} a_0^{(k-1)} \\ a_1^{(k-1)} \\ \vdots \\ a_{k-1}^{(k-1)} \\ 0 \end{bmatrix}$$

   with $a_j = a_j^{(n)}$.

2. For $j = 1, 2, \dots, n+1$ do:

(a) Compute $q_k(x_j)$ by (5.7) for $k = 0, 1, \ldots, n$.

(b) Compute $P'(x_j) = q_k'(x_j)$ using

$$q_1'(x_j) = a_n, \qquad q_k'(x_j) = q_{k-1}(x_j) + x_j q_{k-1}'(x_j), \qquad k = 2, 3, \ldots, n+1.$$

(c) Compute the j^{th} column of $\check{V}^{-1}$, using (5.9).

Remark 1:- The Traub algorithm can compute $(n+1)^2$ entries of V^{-1} in $6(n+1)^2$ flops, but it propagates round-off error, thus leading to inaccurate solutions. The next algorithm has better numerical behavior.

5.3 The Parker Algorithm

Parker described an inversion formula based on Lagrange polynomials [9]. Let $L_j(x) = \dfrac{\prod_{k \neq j}(x - x_k)}{\prod_{k \neq j}(x_j - x_k)}$. be the j^{th} Lagrange polynomial of degree n. Then

$$L_j(x_k) = \begin{cases} 1 & k = j \\ 0 & k \neq j \end{cases}.$$

Express

$$L_j(x) = l_{n,j}x^n + l_{n-1,j}x^{n-1} + \ldots + l_{1,j}x + l_{0,j}. \tag{5.10}$$

For any function $f(x)$, the polynomial $p(x)$ satisfies $p(x_j) = f(x_j), \quad j = 1, \ldots, n+1$, is

$$p(x) = \sum_{j=1}^{n+1} f(x_j) L_j(x) = [1, x, \ldots, x^n] \begin{bmatrix} l_{0,1} & l_{0,2} & \ldots & l_{0,n+1} \\ l_{1,1} & l_{1,2} & \ldots & l_{1,n+1} \\ \vdots & \vdots & \vdots & \vdots \\ l_{n,1} & l_{n,2} & \ldots & l_{n,n+1} \end{bmatrix} \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_{n+1}) \end{bmatrix}.$$

On the other hand the coefficients of p are given by $\breve{V}^{-1} \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_{n+1}) \end{bmatrix}$. Since both are true for any function f, we have

$$\breve{V}^{-1} = \begin{bmatrix} l_{0,1} & l_{0,2} & \ldots & l_{0,n+1} \\ l_{1,1} & l_{1,2} & \ldots & l_{1,n+1} \\ \vdots & \vdots & \vdots & \vdots \\ l_{n,1} & l_{n,2} & \ldots & l_{n,n+1} \end{bmatrix} \tag{5.11}$$

The formula (5.11) can be easily derived from (5.1). Using $P(x_j) = 0$, we have

$$L_j(x) = \frac{P(x)}{(x - x_j)P'(x_j)} = \frac{P(x) - P(x_j)}{(x - x_j)P'(x_j)} = \frac{P[x, x_j]}{P'(x_j)}.$$

From (5.4) and (5.10) we can say that

$$q_{n-k}(x_j) = P'(x_j) l_{k,j}.$$

Basically the two formulas are two faces of the same coin. However the derivatives in this section shows the explicit relations between the Lagrange's polynomials $L_j(x)$ and polynomials $q_j(x), \quad j = 0, \dots, n$. The algorithm is given as follows

1. Compute the coefficients $a_0, \dots, a_n$ of $P(x)$ given by (5.1) as computed in step 1 of the algorithm in section 2.

2. For $j = 1, 2, \dots, n+1$ do

   (a) Compute $q_k(x_j)$ by (5.7) for $k = 0, 1, \dots, n$.

   (b) Compute $P'(x_j)$ from

   $$P'(x_j) = (x_j - x_1)\dots(x_j - x_{j-1})(x_j - x_{j+1})\dots(x_j - x_n)$$

   (c) Compute the j^{th} column of $\breve{V}^{-1}(x)$ using (5.9).

Remark 2:- Comparing the Traub algorithm with the Parker algorithm, one notices that they differ only in step 2(b), thus they are just different versions of the same algorithm. The computational complexity of the algorithm is $6n^2$ flops. By a proper implementation of step 2(b), it can be further reduced slightly. For details see [5]. There is a numerically stable method available for computing Lagrange's polynomials in [3].

5.4 The Bjorck-Pereyra Algorithm

This algorithm has been discussed in Chapter 3. Based on the LU factorization of $\breve{V}$, we have

$$\breve{V}^{-1} = N_0 \dots N_{n-1} D_{n-1}^{-1} M_{n-1} \dots D_0^{-1} M_0.$$

Remark 3:-This method requires $O(n^3)$ flops because of explicit matrix multiplications, but it may provide more accurate results if the nodes x_k are specially ordered, like for example monotone ordering, leja ordering [7].

5.5 Krylov Connection To The Inverse of Vandermonde Matrix

It is straightforward to show

$$\Lambda\breve{V} = \breve{V}C,$$

where C is the companion matrix of Λ. Taking transpose of both sides,

$$\breve{V}^T\Lambda = C^T\breve{V}^T.$$

Multiply both sides by $\breve{V}^{-T}$ to get

$$\breve{V}^{-T}\breve{V}^T\Lambda\breve{V}^{-T} = \breve{V}^{-T}C^T\breve{V}^T\breve{V}^{-T},$$

or

$$\Lambda\breve{V}^{-T} = \breve{V}^{-T}C^T.$$

Define $P = [e_{n+1}, e_n, \ldots, e_1] \in \mathbb{R}^{(n+1)\times(n+1)}$. Note P is also known as the shuffle matrix and $P = P^{-1} = P^T$.

Then

$$\Lambda\breve{V}^{-T}P^{-1} = \breve{V}^{-T}P^{-1}PC^TP^{-1}.$$

Denote $\tilde{C} = PC^TP^{-1}$, we have

$$\Lambda\breve{V}^{-T}P^{-1} = \breve{V}^{-T}P^{-1}\tilde{C}.$$

Because C^T and C are similar, so are $\tilde{C}$ and C. In fact

$$C = \begin{bmatrix} 0 & & & -a_n \\ 1 & \ddots & & \vdots \\ & 1 & \ddots & \vdots \\ 0 & & 1 & -a_0 \end{bmatrix}, \quad \tilde{C} = \begin{bmatrix} -a_0 & \dots & \dots & -a_n \\ 1 & \ddots & & \vdots \\ & 1 & \ddots & \vdots \\ 0 & & 1 & 0 \end{bmatrix}.$$

where $a_0, \dots, a_n$ are coefficients of $P(x)$ defined in (5.1).

For

$$R = \begin{bmatrix} 1 & a_n & a_{n-1} & \dots & \dots & a_0 \\ 0 & 1 & a_n & a_{n-1} & \dots & \vdots \\ & & & \ddots & & \vdots \\ \vdots & & \ddots & & \ddots & a_{n-1} \\ 0 & & & \ddots & & a_n \\ 0 & 0 & \dots & \dots & & 1 \end{bmatrix},$$

it can be easily verified that $\tilde{C} = R^{-1}CR$.

Hence

$$\Lambda\breve{V}^{-T}P^{-1} = \breve{V}^{-T}P^{-1}\tilde{C} \Rightarrow \Lambda\breve{V}^{-T}P^{-1}R^{-1} = \breve{V}^{-T}P^{-1}R^{-1}C.$$

Set $A = \breve{V}^{-T}P^{-1}R^{-1}$. Then

$$\Lambda A = AC.$$

Let $d = Ae_1 = [d_1, d_2, \ldots, d_{n+1}]^T$. By comparing the column on both sides, A can be written as a Krylov matrix given by

$$A = \left[d, \Lambda d, \Lambda^2 d, \ldots, \Lambda^n d\right],$$

which implies A can be written as the product of following matrices

$$A = D\check{V},$$

where $D = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_{n+1} \end{bmatrix}$. So

$$\check{V}^{-T} P^{-1} R^{-1} = D\check{V},$$

and we have

$$\check{V}^{-1} = PR^T \check{V}^T D. \tag{5.12}$$

Still we need to derive the formula for D, i.e. we need to derive $d_1, d_2, \ldots, d_{n+1}$. From the above equation,

$$(\check{V} PR^T \check{V}^T) D = I, \tag{5.13}$$

which clearly implies that $\check{V} PR^T \check{V}^T$ is diagonal . Writing it out, it has the following form

$$\begin{bmatrix} a_1 + 2a_2 x_1 + \ldots + (n+1)x_1^n & 0 & \ldots & 0 \\ 0 & a_1 + 2a_2 x_2 + \ldots + (n+1)x_2^n & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & \ldots & a_1 + 2a_2 x_n + \ldots + (n+1)x_n^n \end{bmatrix}.$$

From (5.1) it is easily verified that for $j = 1 \ldots n+1$,

$$P'(x_j) = a_1 + a_2 x_j + a_3 x_j^2 + \ldots + (n+1)x_j^n.$$

So

$$\check{V} P R^T \check{V}^T = \begin{bmatrix} P'(x_1) & & & \\ & P'(x_2) & & \\ & & \ddots & \\ & & & P'(x_{n+1}) \end{bmatrix},$$

and

$$D = \begin{bmatrix} \frac{1}{P'(x_1)} & & & \\ & \frac{1}{P'(x_2)} & & \\ & & \ddots & \\ & & & \frac{1}{P'(x_{n+1})} \end{bmatrix}.$$

By (5.12),

$$\check{V}^{-1} = \begin{bmatrix} 0 & 0 & \ldots & 1 \\ 0 & \ldots & 1 & 0 \\ \ldots & & & \\ 1 & & & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & \ldots & 0 \\ a_n & \ddots & \ddots & \vdots \\ \vdots & \ddots & & 0 \\ \vdots & \ldots & \ldots & 0 \\ a_0 & \ldots & a_n & 1 \end{bmatrix} \begin{bmatrix} 1 & \ldots & 1 \\ x_1 & & x_{n+1} \\ \vdots & & \vdots \\ x_1^n & \ldots & x_{n+1}^n \end{bmatrix} \begin{bmatrix} \frac{1}{P'(x_1)} & & & \\ & \frac{1}{P'(x_2)} & & \\ & & \ddots & \\ & & & \frac{1}{P'(x_{n+1})} \end{bmatrix}$$

$$= \begin{bmatrix} q_n(x_1) & q_n(x_2) & \ldots & q_n(x_{n+1}) \\ q_{n-1}(x_1) & q_{n-1}(x_2) & \ldots & q_{n-1}(x_{n+1}) \\ \vdots & \vdots & \vdots & \vdots \\ q_0(x_1) & q_0(x_2) & \ldots & q_0(x_{n+1}) \end{bmatrix} \begin{bmatrix} \frac{1}{P'(x_1)} & & & \\ & \frac{1}{P'(x_2)} & & \\ & & \ddots & \\ & & & \frac{1}{P'(x_{n+1})} \end{bmatrix},$$

which is in congruence with results obtained in previous sections.

Chapter 6

Preconditioner

6.1 Deriving Preconditioner

In Chapter 4, we considered the least squares problem $\min \|WVc - Wf\|_2$ for $c \in \mathbb{R}^{m+1}$, where $\{w_i\} > 0$ and $\{x_j\}$ are distinct nodes, W and V are the same as defined in Chapter 4, $f(x)$ is a function and $f = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_{n+1}) \end{bmatrix}$. The solution c can be computed in the following way.

1. Compute the reduced QR factorization and d.

$$WV = \tilde{Q}R, \quad d = \tilde{Q}^T Wf.$$

2. Solve $Rc = d$ for c.

   Obviously different weights $\{w_i\}_{i=1}^{n+1}$ give different inner products $\langle f, g\rangle_w$, in $\mathbb{P}_n$. So the corresponding least squares solution c depends on $\{w_i\}_{i=1}^{n+1}$. From matrix factorization point of view, the reduced QR factorization of WV depends on W. When $m = n$, the least squares problem becomes the system of equations

$$WVc = Wf, \tag{6.1}$$

or equivalently

$$Vc = f. \tag{6.2}$$

Comparing (6.1) and (6.2), we take the diagonal matrix W as a preconditioner [6]. In this case the solution to (6.2) is unique and c is independent of W. From numerical point of view, however one may determine a preconditioner W such that $\kappa(WV)$ is much smaller than $\kappa(V)$, or simply we look for W that solves $\min \kappa(WV)$, over all diagonal W. For detailed reference, see [6] and [8].

It has been established in previous chapters that WV has a unique QR factorization $WV = QR$, where orthogonal Q satisfies $Q^T \Lambda Q = T$, $Qe_1 = \frac{1}{\|w\|_2} w$, and

$$w = [w_1, w_2, \ldots, w_{n+1}]^T, \text{ and } R = \|w\|_2 [e_1, Te_1, \ldots, T^n e_1].$$

So the choice of W is equivalent to the choice of the first column of Q. This gives rise to another related question. How the diagonal matrix W will affect Q and T? Here we basically consider the minimization problem $\min \kappa(WV)$. We use the Frobenius norm for condition number. In principle other norms can be used, but then the computation of $\{w_i\}_{i=1}^{n+1}$ becomes difficult. Denote

$$V^{-1} = \begin{bmatrix} b_{1,1} & b_{1,2} & \ldots & b_{1,n+1} \\ b_{2,1} & b_{2,2} & \ldots & b_{2,n+1} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n+1,1} & b_{n+1,2} & \ldots & b_{n+1,n+1} \end{bmatrix}.$$

Then from the definitions of W and V we have

$$WV = \begin{bmatrix} w_1 p_0(x_1) & w_1 p_1(x_1) & \ldots & w_1 p_n(x_1) \\ w_2 p_0(x_2) & w_2 p_1(x_2) & \ldots & w_2 p_n(x_2) \\ \vdots & \vdots & \ldots & \vdots \\ w_{n+1} p_0(x_{n+1}) & w_{n+1} p_1(x_{n+1}) & \ldots & w_{n+1} p_n(x_{n+1}) \end{bmatrix}, \tag{6.3}$$

and

$$V^{-1}W^{-1} = \begin{bmatrix} b_{1,1}w_1^{-1} & b_{1,2}w_2^{-1} & \dots & b_{1,n+1}w_{n+1}^{-1} \\ b_{2,1}w_1^{-1} & b_{2,2}w_2^{-1} & \dots & b_{2,n+1}w_{n+1}^{-1} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n+1,1}w_1^{-1} & b_{n+1,2}w_2^{-1} & \dots & b_{n+1,n+1}w_{n+1}^{-1} \end{bmatrix}. \tag{6.4}$$

Set $A_i = \sqrt{\sum_{j=0}^{n} p_j^2(x_i)} = \|e_i^T V\|_2$ and $B_i = \sqrt{\sum_{j=1}^{n+1} b_{ji}^2} = \|V^{-1}e_i\|_2$. The square of the Frobenius norms of both the matrices are given as:

$$\|WV\|_F^2 = w_1^2 \sum_{j=0}^{n} p_j^2(x_1) + w_2^2 \sum_{j=0}^{n} p_j^2(x_2) + \ldots + w_{n+1}^2 \sum_{j=0}^{n} p_j^2(x_{n+1}) \tag{6.5}$$

$$= (w_1^2 A_1^2 + w_2^2 A_2^2 + \ldots + w_{n+1}^2 A_{n+1}^2), \tag{6.6}$$

$$\|V^{-1}W^{-1}\|_F^2 = \frac{1}{w_1^2} \sum_{j=1}^{n+1} b_{1j}^2 + \frac{1}{w_2^2} \sum_{j=1}^{n+1} b_{2j}^2 + \ldots + \frac{1}{w_{n+1}^2} \sum_{j=1}^{n+1} b_{n+1,j}^2 \tag{6.7}$$

$$= \left(\frac{1}{w_1^2} B_1^2 + \frac{1}{w_2^2} B_2^2 + \ldots + \frac{1}{w_{n+1}^2} B_{n+1}^2 \right). \tag{6.8}$$

From (6.6) and (6.7),

$$\kappa_F^2(WV) = (w_1^2 A_1^2 + w_2^2 A_2^2 + \ldots + w_{n+1}^2 A_{n+1}^2) \left(\frac{1}{w_1^2} B_1^2 + \frac{1}{w_2^2} B_2^2 + \ldots + \frac{1}{w_{n+1}^2} B_{n+1}^2 \right).$$

Let

$$x = \begin{bmatrix} w_1 A_1 \\ w_2 A_2 \\ \vdots \\ w_{n+1} A_{n+1} \end{bmatrix}, \qquad y = \begin{bmatrix} \frac{B_1}{w_1} \\ \frac{B_2}{w_2} \\ \vdots \\ \frac{B_{n+1}}{w_{n+1}} \end{bmatrix}.$$

By the Cauchy-Schwarz inequality, $\|x\|_2^2 \|y\|_2^2 \geq |x^T y|^2$. Hence

$$\|x\|_2^2 \|y\|_2^2 = \kappa_F^2(WV) \geq \left(w_1 A_1 \frac{B_1}{w_1} + w_2 A_2 \frac{B_2}{w_2} + \ldots + w_{n+1} A_{n+1} \frac{B_{n+1}}{w_{n+1}} \right)^2.$$

$$= (A_1B_1 + A_2B_2 + \ldots + A_{n+1}B_{n+1})^2. \tag{6.9}$$

The equality is attained when $x = ty$, for some constant t. Setting $x = y, (t = 1)$, we have

$$w_iA_i = \frac{B_i}{w_i}, \quad i = 1, \ldots, n+1,$$

$$\Longrightarrow w_i = \sqrt{\frac{B_i}{A_i}}. \tag{6.10}$$

So when $w_1, \ldots, w_{n+1}$ are taken as in (6.10), $\kappa_F(WV)$ has the minimum value, which is

$$\kappa = \min \kappa_F(WV) = (A_1B_1 + A_2B_2 + \ldots + A_{n+1}B_{n+1}). \tag{6.11}$$

Therefore for $t > 0$, the minimizer is

$$W = t \begin{bmatrix} \sqrt{\frac{B_1}{A_1}} & & & \\ & \sqrt{\frac{B_2}{A_2}} & & \\ & & \ddots & \\ & & & \sqrt{\frac{B_{n+1}}{A_{n+1}}} \end{bmatrix}. \tag{6.12}$$

Once the optimal preconditioner is determined, we take

$$w = t \begin{bmatrix} \sqrt{\frac{B_1}{A_1}} \\ \vdots \\ \sqrt{\frac{B_{n+1}}{A_{n+1}}} \end{bmatrix}, \tag{6.13}$$

and use the GR algorithm to solve (6.1) with first column of the orthogonal factor Q parallel to w. The parameter t won't change κ_{min} as defined in (6.11), but it can be used to scale the weights w_i. For instance we may take $t = 1$. We could also set $t = \sqrt{\frac{B_1}{A_1}}$ so that $w_1 = 1$, or set

$$t = \frac{1}{\sqrt{\frac{B_1}{A_1} + \ldots + \frac{B_{n+1}}{A_{n+1}}}},$$

such that $\|w\|_2 = 1$. Although (6.11) provides a formula for $\min \kappa_F(WV)$, it is not clear how small it is in comparison with $\kappa_F(V)$. The next section provides some observations obtained from running numerical tests.

6.2 Numerical Tests

In this section we report some numerical results for several groups Vandermonde-like matrices V and the associated system of linear equations $Vc = f$.

Experimental Objectives

The main purpose for the numerical experiments includes the following objectives

1. Comparison of the optimal condition number of κ_{min} with $\kappa_F(V)$
2. Observation of the change of both κ_{min} and $\kappa_F(V)$ with respect to nodes $\{x_i\}_{i=1}^{n+1}$
3. Comparison of the errors and the residual of numerical solutions based on $Vc = f$ and $WVc = Wf$, respectively, where W is the optimal preconditioner defined in (6.12) with $t = 1$.

Equipment

Computer:- Processor: Intel(R) Pentium(R) 4CPU 2.80 GHz

Memory: 1001.78 MB

Operating System: Kernel Linux 2.6.24-23-generic

Machine Precision: 1.1102×10^{-16}

Software:- MATLAB and Simulink Version 7.8.0.347(R2009a) by The MathWorks.

Research System: Stewie, Math Department KU.

Setup and parameters

To ensure higher accuracy residuals and errors are computed using Variable Precision Arithmetic(vpa) to 32 digits. The parameters, which we chose are

1. Dimension of the matrix, $n+1$
2. Nodes $\{x_i\}$
3. Polynomials $p_j(x)$
4. Function $f(x)$ for right hand side vector f of the system. Sometimes f maybe just a vector.

Quantities displayed

1. Condition numbers: κ_{min} and $\kappa_F(V)$
2. Optimal relative error
$$E_{opt} = \frac{||c_{opt} - c_{exact}||_2}{||c_{exact}||_2}$$
and error
$$E = \frac{||c - c_{exact}||_2}{||c_{exact}||_2}$$
3. Residues:
$$Res_{opt} = ||Vc_{opt} - f||_2$$
$$Res = ||Vc - f||_2$$
c_{opt} is the numerical solution to $WVc = Wf$ with optimal W, and c is the numerical solution to $Vc = f$, and c_{exact} is the solution to $Vc = f$, with data generated by MATLAB vpa(H,32) code.

κ_{min} is computed with the formula (6.11), where $B_1, \ldots, B_{n+1}$ are computed using the elements of V^{-1}, which are computed using MATLAB command "inv(V)". c_{opt} is computed by employing GR algorithm with W defined in (6.12) with $t = 1$. c and c_{exact} are computed by GR algorithm with $w = [1, 1, \ldots, 1]^T$. Again the latter one is computed using vpa. In order to compute $\kappa(V)$, $\kappa(R)$ is computed, where R is the upper triangular factor of V in QR factorization, computed by GR algorithm. All the MATLAB codes are provided in appendix A.

6.2.1 Computations Using Legendre Polynomials to Construct Vandermonde-like Matrices

Each of the following tables show the behavior of preconditioner when tested on various nodes and for different dimensions of V. Here the Vandermonde-like matrix is constructed using polynomials which obey Legendre three term recurrence relation. From (4.15) we have

$$b_0 p_0(x) = 1,$$

$$b_1 p_1(x) = (x - a_1) p_0(x),$$

$$b_j p_j(x) = (x - a_j) p_{j-1}(x) - b_{j-1} p_{j-2}(x),$$

$j = 2, 3....., m$, where a_j and $b_j > 0 \in \mathbb{R}$.

Example 1

Choose $b_1 = 2, b_j = \dfrac{2j}{\sqrt{4j^2 - 1}}$, for $j \geq 2$, and $a_j = 0$ for $j \geq 1$ then $p_j(x)$ are Legendre polynomials for the interval [-2,2].

Nodes:-

$$x_k = 2\cos\big(\pi \frac{2k-1}{2n+2}\big), \quad 1 \leq k \leq n+1,$$

are the zeros of Chebyshev polynomials on [-2,2] and

$$x_k = -2 + 4\big(\frac{k-1}{n}\big)^2 \quad 1 \leq k \leq n+1,$$

are equidistant on [-2,2].

Dimension $n + 1 = 4, 6, 8$.

The results are reported in Table 6.1.

Example 2

Table 6.1: Comparison of $\kappa(V)$ and κ_{min} for $p_j(x)$=Legendre Polynomials

Mat Dim=n+1	κ_F	Nodes	
		Zeros of Chebyshev Polynomial for [-2,2]	Equidistant on [-2,2]
4	κ_{min}	4.08	4.37
	$\kappa_F(R)$	4.65	5.93
6	κ_{min}	6.10	6.93
	$\kappa_F(R)$	7.47	10.64
8	κ_{min}	8.12	11.02
	$\kappa_F(R)$	10.40	18.90

$p_j(x)$ are Legendre polynomials on [-2,2], with the same coefficients as in Example 1.

Nodes:-

$$x_k = -2+4(\frac{k-1}{n})^2 \quad 1 \le k \le n+1$$

are clustered on [-2,2]. The second choice of nodes $\{x_k\}_{i=1}^{n+1}$ are generated randomly using MATLAB command ”randn(n,1)”.

Dimension $n+1 = 4,6,8$.

Results are shown in Table 6.2. The results in Table 6.1, show no significant improvement

Table 6.2: Comparison of $\kappa_F(R)$ and κ_{min} for $p_j(x)$=Legendre Polynomials

Mat Dim	κ_F	Nodes	
		Clustered in [-2,2]	Randomly generated
4	κ_{min}	7.20	15.03
	$\kappa_F(R)$	9.86	36.04
6	κ_{min}	43.21	26.47
	$\kappa_F(R)$	61.14	1.18e+02
8	κ_{min}	4.20e+02	2.12e+03
	$\kappa_F(R)$	6.22e+02	2.70e+04

of the condition number with optimal preconditioner, nor does clustered nodes on [-2,2] shown in Table 6.2. However a significant improvement is noticed for randomly generated nodes, especially when the dimension increases. In Figure 6.1, the graph plots the ratios $\frac{\kappa_{min}}{\kappa_F(\mathrm{R})}$ of Vandermonde-like matrices of dimension $n+1 = 8$, with Legendre polynomials and random nodes. This ratio is computed by running the algorithm 50 times. Each time, a new set of 50

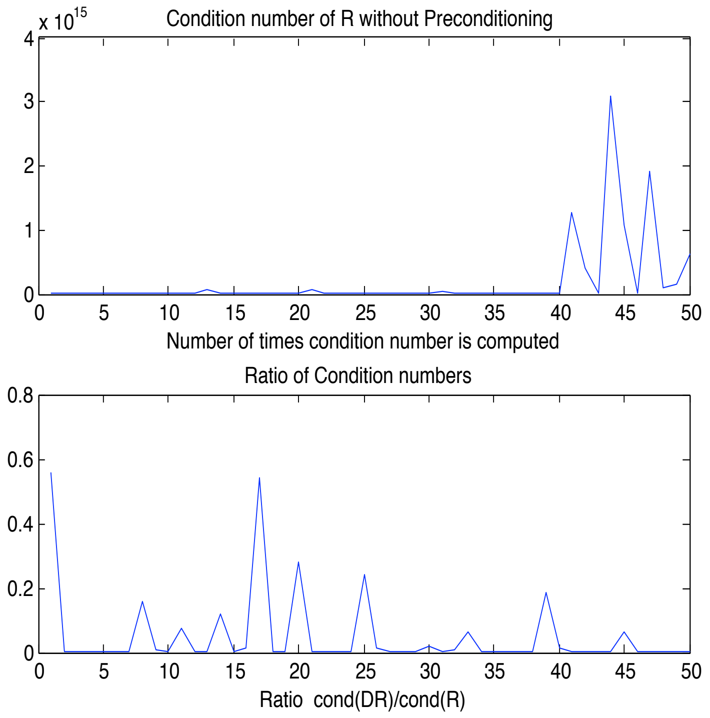

Figure 6.1: Comparison of condition number for Vandermonde-like matrix of dimension 8

nodes, $\{x_k\}_{k=1}^{50}$ is generated and the corresponding ratio is computed. The choice of random nodes actually help us to get insight into the behaviour of the preconditioner, that for some random choice of nodes $\frac{\kappa_{min}}{\kappa_F(\mathrm{R})} \leq 1$.

The optimal residual, Res_{opt} and the optimal error, E_{opt}, was found to be of the order of 10^{-15} for most choice of nodes. However as the matrix dimension increased, Res_{opt}, E_{opt} were found to be of the same order, as the residual and error computed without the application of preconditioner, that being 10^{-14}.

6.2.2 Computations Using Chebyshev Polynomials to Construct Vandermonde-like Matrices

This section uses usual Chebyshev polynomials $p_j(x) = T_j(x/2)$ for $j \geq 1$ to generate Vandermonde-like matrices, where

$$T_j(x) = \cos(j \arccos x).$$

Here we provide only one table to show that for the Vandermonde-like matrices, generated by Chebyshev polynomials, even the choice of randomly generated nodes do not show significant improvement, as was the case in previous section. It seems that the optimal preconditioner won't change the condition number very much.

Example 3

For choice $b_1 = b_2 = \sqrt{2}$, $b_j = 1$ for $j \geq 3$ and $a_j = 0$ for $j \geq 1$, $p_j(x)$ are Chebyshev polynomials for the interval [-2,2].

Nodes:-

$$x_k = -2 + 4\left(\frac{k-1}{n}\right)^2 \quad 1 \leq k \leq n+1$$

are clustered on [-2,2]. The second choice of nodes $\{x_k\}_{i=1}^{n+1}$ are generated randomly using MATLAB command "randn(n,1)".

Dimension $n+1 = 4, 6, 8$.

The numerical results are shown in Table 6.3

Table 6.3: Comparison of $\kappa_F(V)$ and κ_{min} for $p_j(x)$=Chebyshev Polynomials

Mat Dim	$\kappa_F()$	Nodes	
		Clustered in [-2,2]	Randomly generated
4	κ_{min}	5.79	11.63
	$\kappa_F(R)$	6.57	78.72
6	κ_{min}	32.12	1.80e+03
	$\kappa_F(R)$	41.25	5.56e+04
8	κ_{min}	3.04e+02	1.66e+06
	$\kappa_F(R)$	4.34e+02	2.19e+06

6.2.3 Preconditioning of Classical Vandermonde Matrices

In this section the focus is on classical Vandermonde matrices and the behavior of optimal preconditioner with different nodes.

Example 4

Polynomials $p_j(x) = x^{j-1}$

Nodes:- Random, generated by MATLAB command "randn(1,n)".

Dimension $n+1 = 5, 10, 15, 20, 25$.

The numerical results are shown in Table 6.4

From Table 6.4 it can be seen that κ_{min} is one order smaller than $\kappa_F(\check{V})$, thus implying that the

Table 6.4: Comparison of $\kappa_F(\check{V})$ and κ_{min} for $p_j(x) = x^j$ at random nodes.

Matrix Dimensions	$\kappa_F(\check{V})$	κ_{min}
5	508.68	167.39
10	4.07e+04	6.93e+03
15	1.5851e+10	3.66e+09
20	2.50e+15	2.41e+14
25	1.86e+19	1.84e+17

preconditioner works well for Vandermonde matrices also. Another test run is conducted, and

Table 6.5: Comparison of $\kappa_F(\check{V})$ and κ_{min} for $p_j(x) = x^j$ for given nodes.

Nodes x_k	$\kappa_F(\check{V})$	κ_{min}
Zeros of Chebyshev polynomial on [-2,2]	125.6782	48.5591
Equidistant on [-2,2]	175.0491	64.9210
Clustered on [-2,2]	706.7711	289.2667
Extremely placed $\left[-10^4, -10^3, -100, 100, 10^3, 10^4\right]$	1.0103e+20	2.0799e+04
All positive $[.09, .9, 9, 99, 999, 9999]$	2.2822e+20	4.6703e+10
Around zero $\left[-10^{-3}, -10^{-2}, -10^{-1}, 10^{-1}, 10^{-2}, 10^{-3}\right]$	1.7617e+09	2.0799e+04

the results are presented in Table 6.5.

Table 6.5 considers a Vandermonde matrix of fixed dimension, $n = 6$ and compares the Frobenius norm condition numbers. Here the choice of nodes is diverse, the idea is to study the behavior when nodes are not restricted to some interval but are placed extremely. Some of such choices were made and tested. In this example, the ratio $\dfrac{\kappa_{min}}{\kappa(\breve{V})}$ can be as small as of order 10^{-10}.

Chapter 7

Conclusions and Future Work

The study reviewed LU and QR factorizations of Vandermonde and Vandermonde-like matrices. The Factorizations were viewed in context with Krylov matrix with the related Hessenberg reduction.

The study maybe considered as the first step to interpret the behavior of Krylov matrices with polynomial theory, which potentially provides better understanding of Krylov subspace methods for linear systems and eigenvalue problems of large sparse matrices.

In the latter part of study, the behavior of diagonal preconditioner was investigated. The GR algorithm did not show a major improvement when applied to Vandermonde-like Matrices with Legendre and Chebyshev polynomials. It means that the algorithm works with optimal weights but may require some additional tweaking. Also when preconditioner algorithm was applied to classical Vandermonde Matrices, the results were in congruence with expected, i.e. a significant lowering of condition number was observed.

Future work in analyzing the GR algorithm, involves finding the reasons behind failure of preconditioner when applied to Vandermonde-like matrices with Legendre and Chebyshev polynomials. One may start with the knowledge that for generating preconditioner, in our test, the inverse of Vandermonde-like matrices was computed by the conventional way. That definitely decreases efficiency and possibly also accuracy. It would be interesting to look into methods, for instance, the Parker algorithm, which computes the inverse of these matrices in a faster and more efficient way. A faster matrix inverse method, if included in preconditioner algorithm, may possibly give better results on the application of GR algorithm.

Choice of nodes is another point which should be mentioned here. The results showed that the improvement in condition number was different for different nodes, with the exception of Vandermonde-like matrices with Chebyshev polynomials. It can be hereby said the choice of nodes might also play a vital role in the behavior of preconditioner.

Appendix A

Programming Codes in MATLAB

A.1 Main code

```
% This M file uses various functions to compute and compare
% the condition number of R ,the upper triangular matrix
% generated by Lothar Reichel algorithm also known as
% GR algorithm.The purpose of this code is to
% generate a preconditioner in order to obtain
% improvement over the condition number of Vandermonde
% like matrix .
function [x,Res,R_without_scaling,Err,Err_without_scaling,P]
=preconditioner(choice_of_nodes,n,m)
format long
 x = nodes(choice_of_nodes,n),
xv = vpa(x);\\
choice_of_poly=
input('Enter 1 for Legendre coefficients,\\2 for Chebyshev coefficients-:');
[a,b,P] = orthopolyn(choice_of_poly,x,n);
     av = vpa(a);
     bv = vpa(b);
      w = diagmatr(x,P,n,m);
```

```
% The GR algorithm uses  reichel’s method to compute constants
% alpha,beta and least square solution c.
   [alpha,beta,c] = GR_algorithm(x,w,n,m);
[alpha1,beta1,c1] = GR_algorithm(x,ones(1,n),n,m);
[alphav,betav,cv] = GR_algorithm(xv,ones(1,n),n,m);
%The constants alpha ,beta so obtained are further used in getting a QR
%decomposition of Vanderminde like matrices.The following function just
%computes R explicitly and then its condition number.Q which is stored in
%form of Givens rotations is not derived explicitly by the algorithm.
% R is a upper triangular matrix obtained by using optimal weights ’w’
%in GR algorithm by   Reichel
                   R = cnd_no_of_R(alpha,beta,a,b,n);
%R_without_scaling is the upper triangular matrix obtained with using all
%weights =1 in GR algorithm
   R_without_scaling = cnd_no_of_R(alpha1,beta1,a,b,n);
% CNR and CNR_without_scaling are the condition numbers or  R and
% unscaled R.The condition number of R is smaller than unscaled R.Its
% equivalent to the procedure of multiplying unscaled R by a
% preconditioner D such that, D is diagonal matrix consisiting of weights.
                 CNR = cond(R,’fro’)
 CNR_without_scaling = cond(R_without_scaling,’fro’)
% By switching alpha and beta with a and b,the inverse of R can be
% calculated.
                RINV = cnd_no_of_R(a,b,alpha,beta,n);
R_without_scalingINV = cnd_no_of_R(a,b,alpha1,beta1,n);
                  Rv = cnd_no_of_R(av,bv,alphav,betav,n);
%Let us assume x_exact to be exact solution .The residual is compared by
%finding RINV*h-x_exact for scaled R and R_without_scalingINV*h-x_exact.h
%here is the smooth function exp(x) as used in GR algorithm.
                   h = exp(x);
                   h = h’;
                 Sol = RINV*c';
 Sol_without_scaling = R_without_scalingINV*c1';
                 Res = P*Sol-h;
   R_without_scaling = P*Sol_without_scaling-h;
                Solv = Rv*cv';
                 Err = double(Sol-Solv);
 Err_without_scaling = double(Sol_without_scaling-Solv);
```

A.2 Code for GR algorithm

```
function [alpha,beta,c]=GR_algorithm(x,w,n,m)
format long
%We define h(x) to be a continuous function
%h(x)=exp(x);
alpha(1) = x(1);
 beta(1) = w(1);
    c(1) = w(1)*exp(x(1));
     tau = beta(1);
 beta(1) = sqrt(beta(1)^2+w(2)^2);
   gamma = tau/beta(1);
   sigma = -w(2)/beta(1);
     tau = gamma*sigma*(alpha(1)-x(2));
 beta(2) = abs(tau);
    c(2) = sign(double(tau))*(sigma * c(1)+gamma*w(2)*exp(x(2)));
    c(1) = gamma*c(1)-sigma*w(2)*exp(x(2));
alpha(2) = sigma^2*alpha(1)+gamma^2*x(2);
alpha(1) = gamma^2*alpha(1)+sigma^2*x(2);
for j = 2:m-1
        tau = beta(1);
    beta(1) = sqrt(beta(1)^2+w(j+1)^2);
      gamma = tau/beta(1);
      sigma = -w(j+1)/beta(1);
     c(j+1) = sigma*c(1)+gamma*w(j+1)*exp(x(j+1));
       c(1) = gamma*c(1)-sigma*w(j+1)*exp(x(j+1));
         v1 = gamma*sigma*(alpha(1)-x(j+1));
         v2 = sigma*beta(2);
         v3 = gamma^2*x(j+1)+sigma^2*alpha(1);
   alpha(1) = gamma^2*alpha(1)+sigma^2*x(j+1);
    beta(2) = gamma*beta(2);
          M = min(j-2,n-1);
   for k = 1:M
          tau = beta(k+1);
    beta(k+1) = sqrt(beta(k+1)^2+v1^2);
        gamma = tau/beta(k+1);
        sigma = -v1/beta(k+1);
          tau = sigma*c(k+1)+gamma*c(j+1);
       c(k+1) = gamma*c(k+1)-sigma*c(j+1);
       c(j+1) = tau;
          tau = alpha(k+1);
   alpha(k+1) = gamma^2*tau+sigma^2*v3-2*gamma*sigma*v2;
           v1 = (gamma-sigma)*(gamma+sigma)*v2+gamma*sigma*(tau-v3);
           v3 = gamma^2*v3+2*gamma*sigma*v2+sigma^2*tau;
           v2 = sigma*beta(k+2);
    beta(k+2) = gamma*beta(k+2);
   end
   if j-1<n
            tau = beta(j);
        beta(j) = sqrt(beta(j)^2+v1^2);
          gamma = tau/beta(j);
          sigma = -v1/beta(j);
            tau = (gamma-sigma)*(gamma+sigma)*v2+gamma*sigma*(alpha(j)-v3);
      beta(j+1) = abs(tau);
```

```
            tau = sign(double(tau))*(sigma*c(j)+gamma*c(j+1));
           c(j) = gamma*c(j)-sigma*c(j+1);
         c(j+1) = tau;
     alpha(j+1) = sigma^2*alpha(j)+2*sigma*gamma*v2+gamma^2*v3;
       alpha(j) = gamma^2*alpha(j)-2*gamma*sigma*v2+sigma^2*v3;
   end
end
```

A.3 Code For Generating Vandermonde-like Matrices From Polynomial Following Three Term Recurrence Relation.

```
function [a,b,P]=orthopolyn(choice_of_poly,x,n)
 format long
%The following function gives vandermonde like matrix P as an output
% for choice of polynomials 1 implying Legendre and if choice of polynomial
% is 2 then we have  Chebyshev Vandermonde-like matrix
if choice_of_poly==1
        a = (zeros(1,n));
     b(1) = 2;
   b(2:n) = 2*(1:n-1)./sqrt(4*(1:n-1).^2-1);

        P = zeros(n);
   P(:,1) = ones(n,1)/b(1);
   P(:,2) = (x-a(1))/(b(1)*b(2));

   for k=3:n
     P(:,k)=((x'-a(k-1)).*P(:,k-1)-b(k-1)*P(:,k-2))/b(k);
   end
else
        a = (zeros(1,n));
     b(1) = sqrt(2);
     b(2) = sqrt(2);
   b(3:n) = 1;
 P(1:n,1) = ones(n,1)/sqrt(2);
 P(1:n,2) = (x-a(1))*1/(b(1)*b(2));

  for k=3:n
   P(:,k) = ((x'-a(k-1)).*P(:,k-1)-b(k-1)*P(:,k-2))/b(k);
  end
end
```

A.4 Code for Getting Factor R

```
function [r]=cnd_no_of_R(alpha,beta,a,b,n)
%This code exclusively genrates the upper triangular factor R of Rechel's
%Method.
r=[];
r(1,1)=beta(1)/b(1);
r(2,2)=r(1,1)*beta(2)/b(2);
r(1,2)=r(1,1)*(alpha(1)-a(1))/b(2);
 for j=2:n-1
r(j+1,j+1)=r(j,j)*beta(j+1)/b(j+1);
r(j,j+1)=(r(j-1,j)*beta(j)+r(j,j)*(alpha(j)-a(j)))/b(j+1);
     for k=2:j-1
r(k,j+1)=(r(k-1,j)*beta(k)+r(k,j)*(alpha(k)-a(j))
         +r(k+1,j)*beta(k+1)-r(k,j-1)*b(j))/b(j+1);
     end
r(1,j+1)=(r(1,j)*(alpha(1)-a(j))+r(2,j)*beta(2)-r(1,j-1)*b(j))/b(j+1);
```

```
 end
r;
```

A.5 Codes For Computing Nodes and Finding Diagonal Preconditioner

```
function [x]=nodes(choice_of_nodes,n)
format long
switch (choice_of_nodes)
case 1
x = input('Enter a vector which has a dimension equal to the value of n  ');
case 2
x = 2*cos(pi*(2*(1:n)-1)/(2*n));
case 3
x = -2+4*(((1:n)-1)/(n-1)).^2;
case 4
x = 2-4*[0:n-1]/(n-1);
case 5
x = randn(1,n);
end
% The following codes generates the diagonal precondioner with outputs w.
function w=diagmatr(x,P,n,m)

 A=inv(P);
 for i=1:n
    d(i)=sqrt(norm(A(:,i))/norm(P(i,:)));
end
w=d;
```

A.6 Code for Plotting the Ratio $\frac{\kappa(R)}{\kappa_{min}}$

```
%This m file plots the Condition number of R  and ratio of condition number
% of DR and  R.The Vandermonde-like matrix P in this code is generated by
% Legendre polynomials.
function plotting_preconditioner(n,m)
format long
a=(zeros(1,n));
  b(1)=2;
for j=2:n
    b(j)=(2*j/((4*j^2-1)^.5));
end
% The nodes x on which the polynomials are evaluated are generated
% randomly.The loop runs for 50 iterations and each time new nodes are
% generated and henceforth a new P.
for i=1:50
    x=i*randn(n,1)
P(:,1) = ones(n,1)/b(1);
    P(:,2) = (x'-a(1))/(b(1)*b(2));
     for k=3:n

      P(:,k)=((x-a(k-1)).*P(:,k-1)-b(k-1)*P(:,k-2))/b(k);

    end
%Using new nodes and new P everytime a new preconditioner D is generated
%consisting of weights w.
    w=diagmatr(x,P,n,m);
   [alpha,beta]=GR_algorithm(x,w,n,m);
   [alpha1,beta1]=GR_algorithm(x,ones(1,n),n,m);
 R=cnd_no_of_R(alpha,beta,a,b,n);
%R_without_scaling is the upper triangular matrix obtained with using all
```

```
%weights =1 in GR algorithm
R_without_scaling=cnd_no_of_R(alpha1,beta1,a,b,n);
% CNR and CNR_without_scaling are the condition numbers or  R and
% unscaled R.The condition number of R is smaller than unscaled R.Its
% equivalent to the procedure of multiplying unscaled R by a preconditioner
% D such that D is diagonal matrix consisiting of weights.

% By switching alpha and beta with a and b,the inverse of R can be
% calculated.CNR and CNRUS store the condition number of R and unscaled R.
   CNRS(i)=cond(R,'fro');
  CNRUS(i)=cond(R_without_scaling,'fro');
end
subplot(2,1,1),plot(CNRUS)
title('Condition number of R without Preconditioning ')
xlabel('Number of times condition number is computed')

subplot(2,1,2),plot(CNRS./CNRUS)
title('Ratio of Condition numbers  ')
xlabel('Ratio  cond(DR)/cond(R)')
%ylabel('')
CNRUS
CNRS;
CNRS./CNRUS;
```

A.7 Code for Comparing $\kappa(V)$ and κ_{min}

```
function d=vcondition(v)
n=length(v);
for k=0:n-1
    V(k+1,:) =v.^k;
end
```

```
VIN=inv(V);
d=[];
for i=1:n
    d(i)=sqrt(norm(VIN(:,i))/norm(V(i,:)));
end
d=d/d(1)
D=diag(d);
L=cond(V,'fro')
N=cond(D*V,'fro')
N/L;
```

Bibliography

[1] B. Beckermann, *The condition number of real Vandermonde, Krylov and positive definite Hankel matrices.* Numer. Math. Vol 85 , No. 4, pp. 553-577, 2000. Cited on

[2] A. Bjorck and V. Pereyra, *Solution of Vandermonde systems of equations.* AMS Vol. 24, No. 112, pp. 893-903, 1970. Cited on 4, 14, 21, 23

[3] J. P. Berrut and L. N. Trefethen, *Barycentric Lagrange interpolation.* SIAM Review, Vol. 46, No. 3, pp. 501-517, 2004. Cited on 55

[4] C. K. Chui and M. K. Lai, *Vandermonde determinant and Lagrange interpolation in R^s.* In B. L Lin, editor, Nonlinear and Convex Analysis, Marcel Dekker, 1988. Cited on 13

[5] I. Gohberg and V. Olshevsky, *The fast generalized Parker-Traub algorithm for inversion of Vandermonde and related matrices.* AMS Vol. 13, No. 2, pp. 208-234, 1997. Cited on 55

[6] G. H. Golub and F. C. Van Loan, *Matrix Computations 3rd Edition.* The John Hopkins University press, Baltimore, Maryland, 1996. Cited on 16, 20, 61

[7] N. Higham, *Fast solution of Vandermonde-like systems involving orthogonal polynomials.* IMA pp. 473-486, 1988. Cited on 56

[8] N. Higham, *Accuracy and Stability of Numerical Algorithms.* SIAM, 1996. Cited on 22, 23, 61

[9] F. Parker, *Inverses of Vandermonde matrices.* Amer. Math Monthly 71, pp. 410-411, 1964. Cited on 53

[10] M. J. D. Powell, *Approximation Theory and Methods.* Cambridge University Press, 1981. Cited on 13

[11] L. Reichel, *Fast QR decomposition of Vandermonde-like matrices and polynomial least squares approximation.* SIAM J. Matrix Anal, Appl., Vol. 12, No. 3, pp.552-564, 1991. Cited on 4, 14, 30, 41, 44

[12] J. Traub, *Associated polynomials and uniform methods for the solution of linear problems.* SIAM, Review, 8, Vol. No. 3, 277-301, 1966. Cited on 14, 49

[13] L. N. Trefethen and D. Bau, *Numerical Linear Algebra.* SIAM, 1997. Cited on

[14] D. S. Watkins, *Fundamentals of Matrix Computations 2nd Edition.* Wiley Interscience, 2002. Cited on

[15] H. J. Wilkinson, *The Algebraic Eigenvalue Problem.* Oxford University Press, 1965. Cited on

[16] H. Xu, *Function of matrix and Krylov matrices.* Draft. Cited on 7