FAST ALGORITHMS TO COMPUTE MATRIX-VECTOR PRODUCTS FOR PASCAL MATRICES*

ZHIHUI TANG†, RAMANI DURAISWAMI‡, AND NAIL GUMEROV§

Abstract. The Pascal matrix arises in a number of applications. We present a few ways to decompose the Pascal matrices of size $n \times n$ into products of matrices with structure. Based on these decompositions, we propose fast algorithms to compute the product of a Pascal matrix and a vector with complexity $O(n \log n)$. We also present a strategy to stabilize the proposed algorithms. Finally, we also present some interesting properties of the Pascal matrices that help us to compute fast the product of the inverse of a Pascal matrix and a vector, and fast algorithms for generalized Pascal Matrices.

Key words. Matrix-vector product, Pascal matrices, matrix decomposition, structured matrices, Toeplitz matrices, Hankel matrices, Vandermonde matrices

Introduction. If we arrange the binomial coefficients

$$C_r^n = \frac{r!}{n!(r-n)!},\ r = 0, 1, 2, \cdots,\quad n = 0, 1, \cdots, r \tag{0.1}$$

in staggered rows,

$$\begin{array}{ccccccccccccc}
 & & & & & & 1 & & & & & & \\
 & & & & & 1 & & 1 & & & & & \\
 & & & & 1 & & 2 & & 1 & & & & \\
 & & & 1 & & 3 & & 3 & & 1 & & & \\
 & & 1 & & 4 & & 6 & & 4 & & 1 & & \\
 & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & \\
1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1
\end{array}, \tag{0.2}$$

we obtain the famous Pascal triangle. However, Pascal was not the first to discover this triangle, it had been described about 500 years earlier by Chinese mathematician Yang Hui. Therefore in China, it is known as "Yang Hui's triangle" . It also has been discovered in India, Persia and Europe. For a fascinating introduction, see [Edwards02].

The Pascal matrix arises in many applications such as in binomial expansion, probability, combinatorics, linear algebra [Edelman03], electrical engineering [Biolkova99] and order statistics [Aggarwala2001]. As described in previous work [Brawer92, Call93, Zhang97, Zhang98, Edelman03], three different types of $n \times n$ matrices can be obtained by arranging the Pascal triangle into a matrix and truncating appropriately.

*This work was supported by NSF Awards 0086075 and 0219681.

†Department of Mathematics and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (ztang@umiacs.umd.edu)

‡Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (ramani@umiacs.umd.edu)

§Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 (gumerov@umiacs.umd.edu)

These include the lower triangular matrix

$$PL_n = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 2 & 1 & 0 & \cdots & 0 \\ 1 & 3 & 3 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{n-1}^0 & C_{n-1}^1 & C_{n-1}^2 & C_{n-1}^3 & \cdots & C_{n-1}^{n-1} \end{bmatrix}, \tag{0.3}$$

the upper triangular matrix

$$PU_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 2 & 3 & \cdots & C_{n-1} \\ 0 & 0 & 1 & 3 & \cdots & C_{n-1}^2 \\ 0 & 0 & 0 & 1 & \cdots & C_{n-1}^3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & C_{n-1}^{n-1} \end{bmatrix}, \tag{0.4}$$

or the symmetric matrix

$$PS_n = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & C_{n-1}^0 \\ 1 & 2 & 3 & 4 & \cdots & C_n^1 \\ 1 & 3 & 6 & 10 & \cdots & C_{n+1}^2 \\ 1 & 4 & 10 & 20 & \cdots & C_{n+2}^3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{n-1}^0 & C_n^1 & C_{n+1}^2 & C_{n+2}^3 & \cdots & C_{2n-2}^{n-1} \end{bmatrix}. \tag{0.5}$$

Since all these matrices have strong structure, they have many interesting and useful properties [Brawer92, Call93, Zhang97, Zhang98, Edelman03]. We encountered it while working to develop fast translation operators in two dimensions and rotation operators in three dimensions for the fast multipole method [Tang03]. The fast translation operators are of crucial importance for the fast multipole method.

The main results of this paper are fast algorithms, with computational complexity $O(n \log n)$, for computing the matrix-vector product

$$y = Px, \tag{0.6}$$

where P is one of these Pascal matrices or a matrix related to them of size $n \times n$, x and y are vectors of length n.

The paper is organized as follows. In Section 1 we describe some classical results about fast matrix-vector products for matrices with Toeplitz, Hankel and Vandermonde structure. In Section 2 we present our results on decompositions of the Pascal matrices into products of structured matrices, which allow fast computation of the matrix-vector product. If implemented naively, these algorithms are not numerically stable. We provide a modification to stabilize the algorithms in Section 3. Numerical examples are given in Section 4 to demonstrate the effectiveness of the stabilizing technique and accuracy of the algorithm. In Section 5 we extend the algorithms to the so-called generalized Pascal matrices. Some concluding remarks are provided in Section 6.

1. Some classical results. The multiplication of a matrix and a vector arises in many problems in engineering and applied mathematics, and is the fundamental operation in much of scientific computation. For a dense matrix A of size $n \times n$, its product Ax with an arbitrary input vector x requires $O(n^2)$ work by standard matrix-vector multiplication. In many applications, n is very large, and moreover, for the same matrix, the multiplication has to be done over and over again with different input vectors, for example, in iterative methods for solving linear systems. In such cases, one seeks to identify special properties of the matrices in order to reduce the computational work. One special class of matrices are the structured matrices, which are dense but depend on only $O(n)$ parameters. They often appear in communications, control, optimization, and signal processing, etc. The multiplication of any of these matrices with any arbitrary input vector can often be done in $O(n \log^k n)$ time, where usually $0 \leq k \leq 2$, depending on the structure. Since the structured matrices depend on only $O(n)$ parameters, they require reduced storage of only $O(n)$ memory. Examples of structured matrices are Fourier matrices, circulant matrices, Toeplitz matrices, Hankel matrices, Vandermonde matrices, etc. To make the paper self-contained we provide the following resume of fast matrix-vector product algorithms for these matrices.

1.1. Fourier matrices. The most important class of matrices in all fast algorithms are the Fourier matrices.

DEFINITION 1.1. *A Fourier matrix of order n is defined as the following*

$$F_n = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix}, \tag{1.1}$$

where

$$\omega_n = e^{-\frac{2\pi i}{n}}, \tag{1.2}$$

is an nth root of unity.

It is well known that the product of this matrix with any vector is the so-called discrete Fourier transform, which can be done efficiently using the fast Fourier transform (FFT) algorithm [Cooley65]. Notice that the Fourier matrix is a unitary matrix, that is, $F_n F_n^* = I$, therefore, the conjugate transpose F_n^* is also a unitary matrix. The corresponding efficient matrix-vector product is the inverse fast Fourier transform (IFFT) [Van92, Golub96].

THEOREM 1.2. *The FFT and IFFT can be done in $O(n \log n)$ time and $O(n \log n)$ memory references.*

A proof can be found in [Cooley65] or [Van92]. This theorem is the basis for a number of other efficient algorithms, for example, the product of a circulant matrix and a vector.

1.2. Circulant matrices. DEFINITION 1.3. *A matrix of the form*

$$C_n = C(x_1, ..., x_n) = \begin{bmatrix} x_1 & x_n & x_{n-1} & \cdots & x_2 \\ x_2 & x_1 & x_n & \cdots & x_3 \\ x_3 & x_2 & x_1 & \cdots & x_4 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_n & x_{n-1} & x_{n-2} & \cdots & x_1 \end{bmatrix} \tag{1.3}$$

is called a circulant matrix.

It is easy to see that a circulant matrix is completely determined by the entries in the first column. All other columns are obtained by a shift of the previous column. It has the following important property.

THEOREM 1.4. *Circulant matrices $C_n(x)$ can be diagonalized by the Fourier matrix,*

$$C_n(x) = F_n^* \cdot diag(F_n x) \cdot F_n, \tag{1.4}$$

where $x = (x_1, ..., x_n)'$.

A proof can be found in [Bai2000]. Given this theorem, we have the following fast algorithm for a matrix-vector product for a circulant matrix.

Given a circulant matrix C_n, and a vector y, the product

$$C_n y \tag{1.5}$$

can be computed efficiently using the following four steps:

1. compute f =FFT(y),
2. compute g =FFT(x),
3. compute the element wise vector-vector product $h = f. * g$,
4. compute z =IFFT(h) to obtain $C_n y$

Since the FFT and the IFFT can be done in $O(n \log n)$ time and memory references, $C_n y$ can be obtained in $O(n \log n)$ time and memory references [Bai2000, Lu98].

1.3. Toeplitz matrices. Given an algorithm for a fast matrix-vector product for circulant matrices, it is easy to see the algorithm for a Toeplitz matrix, since a Toeplitz matrix can be embedded into a circulant matrix.

DEFINITION 1.5. *A matrix of the form*

$$T_n = T(x_{-n+1}, \cdots, x_0, ..., x_{n-1}) = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_{n-1} \\ x_{-1} & x_0 & x_1 & \cdots & x_{n-2} \\ x_{-2} & x_{-1} & x_0 & \cdots & x_{n-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{-n+1} & x_{-n+2} & x_{-n+3} & \cdots & x_0 \end{bmatrix} \tag{1.6}$$

is called a Toeplitz matrix.

A Toeplitz matrix is completely determined by its first column and first row, and thus depends on $2n-1$ parameters. The entries of T_n are constant down the diagonals parallel to the main diagonal. It arises naturally in problems involving trigonometric moments. Sometimes we denote a Toeplitz matrix using its first column vector and row vector

$$c = \begin{bmatrix} c_0 & c_1 & c_2 & ... & c_{p-1} \end{bmatrix}', r = \begin{bmatrix} c_0 & r_1 & r_2 & ... & r_{p-1} \end{bmatrix} \tag{1.7}$$

by

$$Toep(c, r') = Toep \begin{bmatrix} c_0 & c_0 \\ c_1 & r_1 \\ c_2 & r_2 \\ \vdots & \vdots \\ c_{p-1} & r_{p-1} \end{bmatrix} \tag{1.8}$$

THEOREM 1.6. *[Bai2000, Kailath99] The product of any Toeplitz matrix and any vector can be done in $O(n \log n)$ time and memory references.*

Proof. Given a Toeplitz matrix T_n and a vector y, to compute the product $T_n y$, a Toeplitz matrix can first be embedded into a $2n \times 2n$ circulant matrix C_{2n} as follows

$$C_{2n} = \begin{bmatrix} T_n & S_n \\ S_n & T_n \end{bmatrix}, \tag{1.9}$$

where

$$S_n = \begin{bmatrix} 0 & x_{-n+1} & x_{-n+2} & \cdots & x_{-1} \\ x_{n-1} & 0 & x_{-n+1} & \cdots & x_{-2} \\ x_{n-2} & x_{n-1} & 0 & \cdots & x_{-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_1 & x_2 & x_3 & \cdots & 0 \end{bmatrix}. \tag{1.10}$$

Then $T_n y$ can be multiplied as

$$C_{2n} \cdot \begin{bmatrix} y \\ 0_{n\times n} \end{bmatrix} = \begin{bmatrix} T_n & S_n \\ S_n & T_n \end{bmatrix} \cdot \begin{bmatrix} y \\ 0_{n\times n} \end{bmatrix} = \begin{bmatrix} T_n y \\ S_n y \end{bmatrix}, \tag{1.11}$$

which can be implemented to be done in $O(n \log n)$ time and memory references. ∎

1.4. Hankel matrices. DEFINITION 1.7. *A matrix of the form*

$$H_n = H(x_{-n+1}, \cdots, x_0, \cdots, x_{n-1}) = \begin{bmatrix} x_{-n+1} & x_{-n+2} & x_{-n+3} & \cdots & x_0 \\ x_{-n+2} & x_{-n+3} & x_{-n+4} & \cdots & x_1 \\ x_{-n+3} & x_{-n+4} & x_{-n+5} & \cdots & x_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_0 & x_1 & x_2 & \cdots & x_{n-1} \end{bmatrix} \tag{1.12}$$

is called a Hankel matrix.

A Hankel is completely determined by its first column and last row and thus depends on $2n-1$ parameters. The entries of T_n are constant along the diagonals that are perpendicular to the main diagonal. It arises naturally in problems involving power moments. It has the following property [Golub96].

THEOREM 1.8. *The product of a Hankel matrix and any vector can be done in $O(n \log n)$ time and memory references.*

Proof. Notice that if

$$I_p = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}, \tag{1.13}$$

is the backward identity permutation matrix, then $I_p H_n$ is a Toeplitz matrix for any Hankel matrix H_n, and $I_p T_n$ is a Hankel matrix for any Toeplitz matrix T_n. The product $H_n y$ for any vector y can be computed as follows [Kailath99]: first compute the product $\dot{(I_p H_n)} y$ of a Toeplitz matrix $I_p H_n$ and vector y as in (1.11), then apply the permutation to the vector $\dot{(I_p H_n)} y$ to have $P\dot{(PH_n)} y$, which is what we want since $I_p = I_p^t = I_p^{-1}$. ∎

1.5. Vandermonde matrices. DEFINITION 1.9. *Suppose* $\{x_i,\ i = 0, 1, ..n\} \in \mathbb{C}^{n+1}$*, a matrix of the form*

$$V = V(x_0, x_1, ..., x_n) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \cdots & \cdots & \cdots & \cdots \\ x_0^n & x_1^n & \cdots & x_n^n \end{bmatrix} \tag{1.14}$$

is called a Vandermonde matrix.

A Vandermonde matrix is completely determined by its second row, and so depends on $n + 1$ parameters. All rows are powers of the second row from the power 0 to power n. It is a fact that

$$\det A = \prod_{i,j=0, i>j}^{n} (x_i - x_j) \tag{1.15}$$

so a Vandermonde matrix is nonsingular if and only if all the $(n + 1)$ parameters $x_0, x_1, ..., x_n$ are distinct. In this paper we assume this is the case whenever we need the inverse of this matrix.

A Fourier matrix is a special case of Vandermonde matrix. The transpose of a Vandermonde matrix arises naturally in polynomial evaluations or polynomial interpolations. There exist efficient algorithms for fast matrix-vector product for a Vandermonde matrix, its transpose, its inverse, and the transpose of its inverse. All of them are of complexity $O(n \log^2 n)$, although there are associated stability problems [Driscoll97, Moore93]. The basic idea is to factor the matrices into products of sparse matrices, Toeplitz matrices and the like, so that the FFT can be applied to speed up the computations. We state these facts as a theorem below. The details can be found in [Driscoll97, GohbergL94, GohbergC94, Lu98, Moore93, Pan92].

THEOREM 1.10. *The product of any Vandermonde matrix, its transpose, its inverse, or the transpose of its inverses with any vector is of complexity* $O(n \log^2 n)$.

In later sections, we give representations of the Pascal matrices in factored forms in terms of Vandermonde matrices. There exist a number of algorithms for the product of a Vandermonde matrix and a vector and techniques to overcome the instability problems associated with the algorithm [Driscoll97, Moore93]. However, in this paper we do not recommend use of those factored representations involving Vandermonde matrices when alternate representations are available due to the inferior complexity and the reported instability of the Vandermonde algorithms.

2. Fast algorithms. In this section we present decompositions of the Pascal matrices which allow fast computation of the matrix-vector product.

2.1. Decomposition of the lower triangular Pascal matrix. We will use the following identity in our implementation of the fast algorithms.

THEOREM 2.1. *The Pascal matrix* PL *can be decomposed as,*

$$PL = diag(v_1) \cdot T \cdot diag(v_2), \tag{2.1}$$

where the vectors v_1 and v_2 are

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 2! \\ 3! \\ \vdots \\ (p-1)! \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ \frac{1}{1!} \\ \frac{1}{2!} \\ \frac{1}{3!} \\ \vdots \\ \frac{1}{(p-1)!} \end{bmatrix}, \tag{2.2}$$

and the matrix

$$T = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ \frac{1}{2!} & \frac{1}{1!} & 1 & \cdots & 0 \\ \frac{1}{3!} & \frac{1}{2!} & \frac{1}{1!} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{(p-1)!} & \frac{1}{(p-2)!} & \frac{1}{(p-3)!} & \cdots & 1 \end{bmatrix} \tag{2.3}$$

is a Toeplitz matrix.

Proof. Notice that the (n, m) entry PL_{nm} of the Pascal matrix is

$$PL_{nm} = \begin{cases} C_{n-1}^{m-1} & \text{if } n \geq m \\ 0 & \text{if } n < m \end{cases}, \tag{2.4}$$

where $C_{n-1}^{m-1} = \frac{(n-1)!}{(n-m)!(m-1)!}$. That is, every entry in n-th row of the Pascal matrix has a common factor $(n-1)!$, and every entry in m-th column of the Pascal matrix has a common factor $\frac{1}{(m-1)!}$. We can take out the common factor $(n-1)!$ of the n-th row and common factor $\frac{1}{(m-1)!}$ of the m-th column, and multiply from left side by a diagonal matrix which is the identity, except that the n-th entry in the diagonal is $(n-1)!$, and multiply from right side by a diagonal matrix which is the identity, except that the m-th entry in the diagonal is $\frac{1}{(m-1)!}$. This can be done for every row and column.

$$PL = \begin{bmatrix} \frac{0!}{0!0!} & 0 & 0 & \cdots & 0 \\ \frac{1!}{1!0!} & \frac{1!}{0!1!} & 0 & \cdots & 0 \\ \frac{2!}{2!0!} & \frac{2!}{1!1!} & \frac{2!}{0!2!} & \cdots & 0 \\ \frac{3!}{3!0!} & \frac{3!}{2!1!} & \frac{3!}{1!2!} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{(p-1)!}{(p-1)!0!} & \frac{(p-1)!}{(p-2)!1!} & \frac{(p-1)!}{(p-3)!2!} & \cdots & 1 \end{bmatrix} \tag{2.5}$$

Therefore we have factored the Pascal matrix into products of matrices with a Toeplitz matrix T in the middle and p diagonal matrices on the left of T, and p diagonal matrices on the right of T. Multiplying the diagonal matrices on the left and the right respectively, we end up with the diagonal matrices $diag(v_1)$ and $diag(v_2)$. □

With the notation for Toeplitz matrix introduced earlier, T can be written as

$$T = \text{Toep} \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ \frac{1}{2} & 0 \\ \frac{1}{6} & 0 \\ \vdots & \vdots \\ \frac{1}{(p-1)!} & 0 \end{bmatrix} \tag{2.6}$$

From Theorem 2.1, It is clear that the multiplication of a Pascal matrix PL and a vector x can be done in three steps: first calculate the element-wise multiplication of $u = v_1 .* x$, which requires p multiplications. Then calculate the product $w = Tu$ of Toeplitz matrix T and vector u as (1.11), which requires $O(p \log p)$ work by Theorem 1.6. And finally calculate another element-wise multiplication of $v_1 .* w$ to obtain the product PLx. Therefore we have the following.

THEOREM 2.2. *The multiplication of the $p \times p$ lower triangular Pascal matrix and a p vector can be done in $O(p \log p)$ operations.*

While we usually use the above decomposition to build fast algorithms for the product of the Pascal matrix and a vector, we have found some other ways to factor the matrix which we state here. There may be useful in other contexts, such as in analytical work.

2.1.1. Alternate decomposition 1. LEMMA 2.3. *The Pascal matrix PL can be decomposed as the following,*

$$PL = V2 * V1^{-1}, \tag{2.7}$$

where

$$V_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ x_1 & x_2 & x_3 & x_4 & \cdots & x_p \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & \cdots & x_p^2 \\ x_1^3 & x_2^3 & x_3^3 & x_4^3 & \cdots & x_p^3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{p-1} & x_2^{p-1} & x_3^{p-1} & x_4^{p-1} & \cdots & x_p^{p-1} \end{bmatrix} \tag{2.8}$$

$$V_2 = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ (x_1+1) & (x_2+1) & (x_3+1) & \cdots & (x_p+1) \\ (x_1+1)^2 & (x_2+1)^2 & (x_3+1)^2 & \cdots & (x_p+1)^2 \\ (x_1+1)^3 & (x_2+1)^3 & (x_3+1)^3 & \cdots & (x_p+1)^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (x_1+1)^{p-1} & (x_2+1)^{p-1} & (x_3+1)^{p-1} & \cdots & (x_p+1)^{p-1} \end{bmatrix} \tag{2.9}$$

are Vandermonde matrices, and $\{x_i, i = 1, 2, \cdots, p\}$ are distinct numbers.

Proof. Because PL is a matrix of binomial coefficients, it is easy to see that

$$PLV_1 = V_2. \tag{2.10}$$

Notice that $\{x_i, i = 1, 2, \cdots, p\}$ are distinct numbers and can be arbitrary. Hence V_1 is nonsingular and its inverse exists. Thus we have

$$PL = V_2 * V_1^{-1}. \tag{2.11}$$

□

If we let V_1 be a matrix with one single column, the formula (2.10) provides us a tool to obtain the matrix-vector product V_2 of PL and V_1, which can be used as a true value to test against results from other method. In addition, from Theorem 1.10, we know that a Vandermonde matrix and its inverse can be multiplied by vectors in $O(p \log^2 p)$ time, this decomposition also allows fast matrix-vector product for the Pascal matrix. However, it is slower than the previous decomposition. Furthermore, many existing algorithms for Vandermonde matrices are not stable (see [Moore93] and [Driscoll97]). Therefore it is not generally the preferred algorithm.

2.1.2. Alternate decomposition 2. LEMMA 2.4. *A Pascal matrix can be decomposed as the product of $p-1$ matrices*

$$PL = A_1 * A_2 * \cdots * A_{p-1} \tag{2.12}$$

where

$$A_i = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 & 1 \end{bmatrix} \tag{2.13}$$

with p number of 1's as its diagonal entries, and i number of 1's as its sub-diagonal entries starting from the position $(p, p-1)$.

Proof. We will prove this by mathematical induction. It is trivial for $p = 2$. Now assume that for $p = n$,

$$PL^{(n)} = A_1^{(n)} * A_2^{(n)} * \cdots * A_{n-1}^{(n)}, \tag{2.14}$$

where $PL^{(n)}$ is the Pascal matrix of size $n \times n$, and $A_i^{(n)}$ is A_i as defined in (2.13) of size $n \times n$. For $p = n+1$, we need to prove that

$$PL^{(n+1)} = A_1^{(n+1)} * A_2^{(n+1)} * \cdots * A_n^{(n+1)}. \tag{2.15}$$

It is easy to see that

$$A_i^{(n+1)} = \begin{bmatrix} 1 & 0 \\ 0 & A_i^{(n)} \end{bmatrix} \quad \text{for } i = 1, 2, \cdots, n-1. \tag{2.16}$$

That is, we need to prove

$$PL^{(n+1)} = \begin{bmatrix} 1 & 0 \\ 0 & A_1^{(n)} \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & A_2^{(n)} \end{bmatrix} * \cdots * \begin{bmatrix} 1 & 0 \\ 0 & A_{n-1}^{(n)} \end{bmatrix} * A_n^{(n+1)}. \tag{2.17}$$

By assumption, we have

$$\begin{bmatrix} 1 & 0 \\ 0 & A_1^{(n)} \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & A_2^{(n)} \end{bmatrix} * \cdots * \begin{bmatrix} 1 & 0 \\ 0 & A_{n-1}^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & PL^{(n)} \end{bmatrix}. \tag{2.18}$$

Therefore, we need only to prove that

$$PL^{(n+1)} = \begin{bmatrix} 1 & 0 \\ 0 & PL^{(n)} \end{bmatrix} * A_n^{(n+1)}. \tag{2.19}$$

It is easy to see that the entries in the first column of both sides are one's, the entries in the first row of both sides are the same. For all other entries, we need to prove that

$$PL_{ij}^{(n+1)} = PL_{(i-1)(j-1)}^{(n)} + PL_{(i-1)j}^{(n)} \tag{2.20}$$

This is trivial for all entries in the upper triangular part of the matrices since all entries are zeroes. This is also true for the entries of the diagonal since $PL^{(n)}_{(i-1)j} = 0$, and $PL^{(n+1)}_{ij}$ and $PL^{(n)}_{(i-1)(j-1)}$ are all one's. What is left to prove is the lower triangular part of the matrices. We know

$$PL^{(n+1)}_{ij} = C^{j-1}_{i-1}, \tag{2.21}$$

and

$$PL^{(n)}_{(i-1)(j-1)} + \ PL^{(n)}_{(i-1)j} = C^{j-2}_{i-2} + C^{j-1}_{i-2} = C^{j-1}_{i-1}. \tag{2.22}$$

Therefore for $p = n + 1$, we have

$$PL^{(n+1)} = A^{(n+1)}_1 * A^{(n+1)}_2 * \cdots * A^{(n+1)}_n. \tag{2.23}$$

This completes the proof. □

This decomposition would require $O(p^2)$ operations for matrix-vector product. *However, all these are additions and there are no multiplications.* This may be suitable for some computer architectures.

2.2. The upper triangular Pascal matrix. We have shown some fast multiplication algorithm for a lower triangular Pascal matrix and a vector. It is easy to see that the upper triangular Pascal matrix PU is the transpose of the lower triangular Pascal matrix PL. Therefore, it can be decomposed in a similar way to the lower triangular Pascal matrix. Indeed, applying the transpose to different decompositions (2.1), (2.7), and (2.12) of the lower triangular Pascal matrix, we would obtain decompositions which allow fast matrix-vector products. Therefore we also have the following theorem similar to Theorem 2.2.

THEOREM 2.5. *The multiplication of the upper triangular Pascal matrix and a vector can be done in $O(p \log p)$ operations.*

2.3. The symmetric Pascal matrix. The following lemma gives the Cholesky decomposition of the matrix PS [Edelman03].

LEMMA 2.6.

$$PS = PL * PU \tag{2.24}$$

A number of proofs can be found in [Edelman03]. This identity implies all decompositions that admit fast matrix-vector product for the Pascal matrices PL and PU can be utilized to do fast matrix-vector product for the Pascal matrix PS, since we can first multiply PU by the vector to obtain the product, which is a vector, and then multiply by PL.

We also have the following factorization.

LEMMA 2.7. *The matrix PS can be decomposed as the following,*

$$PS = diag(v) \cdot H \cdot diag(v), \tag{2.25}$$

where

$$v = \begin{bmatrix} 1 \\ \frac{1}{1!} \\ \frac{1}{2!} \\ \frac{1}{3!} \\ \vdots \\ \frac{1}{(p-1)!} \end{bmatrix}, \text{ and } H = \begin{bmatrix} 0! & 1! & 2! & \cdots & (p-1)! \\ 1! & 2! & 3! & \cdots & p! \\ 2! & 3! & 4! & \cdots & (p+1)! \\ 3! & 4! & 5! & \cdots & (p+2)! \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (p-1)! & p! & (p+1)! & \cdots & (2p-2)! \end{bmatrix} \tag{2.26}$$

is a Hankel matrix.

Proof. This lemma can be proved in a way similar to the proof of (2.1). Notice that the (n, m) entry PS_{nm} of the matrix PS is

$$PS_{nm} = C^n_{n+m}. \tag{2.27}$$

where $C^n_{n+m} = \frac{(n+m)!}{n!m!}$. That is, every entry in n-th row of the matrix has a common factor $\frac{1}{n!}$, and every entry in m-th column of the Pascal matrix has a common factor $\frac{1}{m!}$. We can take out the common factor $\frac{1}{n!}$ of the n-th row and common factor $\frac{1}{m!}$ of the m-th column, and multiply from left side by a diagonal matrix which is the identity, except that the n-th entry in the diagonal is $\frac{1}{n!}$, and multiply from right side by a diagonal matrix which is the identity, except that the m-th entry in the diagonal is $\frac{1}{m!}$. This can be done for every row and column.

$$PS = \begin{bmatrix} \frac{0!}{0!0!} & \frac{1!}{1!0!} & \frac{2!}{2!0!} & \cdots & \frac{(p-1)!}{(p-1)!0!} \\ \frac{1!}{0!1!} & \frac{2!}{1!1!} & \frac{3!}{2!1!} & \cdots & \frac{p!}{(p-1)!1!} \\ \frac{2!}{0!2!} & \frac{3!}{1!2!} & \frac{4!}{2!2!} & \cdots & \frac{(p+1)!}{(p-1)!2!} \\ \frac{3!}{0!3!} & \frac{4!}{1!3!} & \frac{5!}{2!3!} & \cdots & \frac{(p+2)!}{(p-1)!3!} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{(p-1)!}{0!(p-1)!} & \frac{p!}{1!(p-1)!} & \frac{(p+1)!}{2!(p-1)!} & \cdots & \frac{(2p-2)!}{(p-1)!(p-1)!} \end{bmatrix}. \tag{2.28}$$

Therefore we have factored the Pascal matrix into products of matrices with a Hankel matrix H in the middle and p diagonal matrices on the left, and p diagonal matrices on the right. Multiplying the diagonal matrices on the left and the right respectively, we end up with the diagonal matrices $diag(v)$ and $diag(v)$. □

It is clear from the lemmas above that the multiplication of the matrix PS and any vector x can be either done by successively apply PU and PL to x, or first calculate the element-wise vector product of $v. * x$, then apply Hankel matrix H to the product, finally with the obtained vector, do another element-wise vector product with v to arrive the result of $PS \cdot x$. In either process, the involved matrices are either Toeplitz matrices or Hankel matrices. By Theorems 1.6, and 1.8, we have the following.

THEOREM 2.8. *The multiplication of matrix PS and any vector can be done in $O(p \log p)$ operations.*

Although we can use both decompositions to do the matrix-vector product, the first one is less efficient in comparison with the second one. The reason is that the cost of the product of a Toeplitz matrix and a vector is the same as that of a Hankel matrix and a vector, and the first one requires two multiplications of a Toeplitz matrix and a vector. We recommend use of the second decomposition.

3. Stability and Implementation Issues. We have discussed how to multiply the Pascal matrices with a vector efficiently through matrix decomposition. Notice that the entries of the Toeplitz or Hankel matrices in the decompositions have very different magnitudes of numbers, and there can exist instability problems if the decomposition is implemented naively. We discuss the implementation of these fast algorithms and provide modifications required to achieve numerical stability in this section.

Fast algorithms for computing the product of a Pascal matrix and a vector are based on the decomposition (2.1) and (2.25). We analyze the stability problem of

the fast algorithm for the lower triangular Pascal matrix as an example and present techniques to stabilize it.

Given a Pascal matrix PL of size $p \times p$, we can factor it into

$$PL = \mathrm{diag}(v_1) \cdot T(c_p, r_p') \cdot \mathrm{diag}(v_2), \tag{3.1}$$

where

$$v_1 = \begin{bmatrix} 1 & 1! & 2! & 3! & \cdots & (p-1)! \end{bmatrix}', \tag{3.2}$$

$$v_2 = \begin{bmatrix} 1 & 1! & \frac{1}{2!} & \frac{1}{3!} & \cdots & \frac{1}{(p-1)!} \end{bmatrix}', \tag{3.3}$$

$$c_p = \begin{bmatrix} 1 & 1! & \frac{1}{2!} & \frac{1}{3!} & \cdots & \frac{1}{(p-1)!} \end{bmatrix}', \tag{3.4}$$

$$r_p = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{3.5}$$

For a vector

$$a = \begin{bmatrix} a_0 & a_1 & a_2 & ... & a_{p-1} \end{bmatrix}', \tag{3.6}$$

the product

$$PLa = \mathrm{diag}(v_1) \cdot T(c_p, r_p') \cdot \mathrm{diag}(v_2) \cdot a \tag{3.7}$$

requires three matrix-vector products, of which two involve diagonal matrices, one involves a Toeplitz matrix. Therefore the product Px can be done in $O(p \log p)$ time instead of $O(p^2)$ time required by straightforward matrix-vector product. A naive implementation of the above method shows that the precision gets worse as p gets larger. The reason for this is that the entries in the Toeplitz matrix and the vector v_2 are of very different magnitudes, varying approximately from 1 to $\frac{1}{(p-1)!}$. When we compute the matrix-vector product, we need to compute the FFT of two vectors

$$u = \begin{bmatrix} 1 & 1! & \frac{1}{2!} & \frac{1}{3!} & \cdots & \frac{1}{(p-1)!} & 0 & 0 & \cdots & 0 \end{bmatrix}', \tag{3.8}$$

$$x = \begin{bmatrix} a_0 & 1!a_1 & \frac{1}{2!}a_2 & \frac{1}{3!}a_3 & \cdots & \frac{1}{(p-1)!}a_{p-1} & 0 & 0 & \cdots & 0 \end{bmatrix}'. \tag{3.9}$$

For a large p, when we compute the FFT of u and x, the final result would be the same if we simply treated the entries such as $\frac{1}{(p-1)!}$ as zeros, and this causes the numerical instability. Therefore we need to find a way to increase the effect of entries of smaller magnitude by bringing all nonzero terms in u and x to the same magnitude. This can be done by multiplying or dividing the entries by some constant factors and still preserving the same structure, viz. a Toeplitz matrix. Indeed, the Pascal matrix can be expressed by introducing a new parameter α as follows,

$$P(\alpha) = \mathrm{diag}(v_1(\alpha)) \cdot Toep\left[c_p(\alpha), r_p'\right] \cdot \mathrm{diag}(v_2(\alpha)), \tag{3.10}$$

where

$$v_1(\alpha) = \begin{bmatrix} 1 & \frac{1}{\alpha} & \frac{2}{\alpha^2} & \frac{6}{\alpha^3} & ... & \frac{(p-1)!}{\alpha^{p-1}} \end{bmatrix}', \tag{3.11}$$

$$v_2(\alpha) = \begin{bmatrix} 1 & \frac{\alpha}{1} & \frac{\alpha^2}{2} & \frac{\alpha^3}{6} & ... & \frac{\alpha^{p-1}}{(p-1)!} \end{bmatrix}', \tag{3.12}$$

$$c_p(\alpha) = \begin{bmatrix} 1 & \frac{\alpha}{1} & \frac{\alpha^2}{2} & \frac{\alpha^3}{6} & ... & \frac{\alpha^{p-1}}{(p-1)!} \end{bmatrix}', \tag{3.13}$$

$$r_p(\alpha) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{3.14}$$

With this factorization, it is possible to implement a fast, numerically stable algorithm by selecting a proper value of α.

We need to select a proper α so that the magnitude of maximum and minimum of the nonzero entries in the vector $v_2(\alpha)$ and $c_p(\alpha)$ are approximately the same. The Fast Fourier transform is applied to the two vectors

$$x(\alpha) = \left[\begin{array}{cccccccccc} a_0 & \frac{\alpha a_1}{1} & \frac{\alpha^2 a_2}{2} & \frac{\alpha^3 a_3}{6} & ... & \frac{\alpha^{p-1} a_{p-1}}{(p-1)!} & 0 & 0 & ... & 0 \end{array} \right]' \tag{3.15}$$

and

$$u(\alpha) = \left[\begin{array}{cccccccccc} 1 & \alpha & \frac{1}{2}\alpha^2 & \frac{1}{6}\alpha^3 & ... & \frac{1}{(p-1)!}\alpha^{p-1} & 0 & 0 & ... & 0 \end{array} \right]'. \tag{3.16}$$

Assuming all entries of $a = \left[\begin{array}{cccc} a_0 & a_1 & a_2 & ... \quad a_{p-1} \end{array} \right]'$ are of the same magnitude, the entries in $x(\alpha)$ and the entries in $u(\alpha)$ are of the same magnitude.

We want to choose one value α so that all nonzero entries of $x(\alpha)$ and $u(\alpha)$ are as close to each other as possible. Let us consider a typical entry

$$f(m) = \frac{\alpha^m}{m!},\ m = 0, 1, 2, ..., p-1. \tag{3.17}$$

We want the maximum and minimum of this function to be as close as possible. We will iteratively find an α which satisfies this criterion. To start the iteration we need an approximate guess for α. The following analysis provides this guess.

If $\alpha \geq p-1$, then

$$f_{\min} = 1, \quad \text{and} \quad f_{\max} = \frac{\alpha^{p-1}}{(p-1)!}. \tag{3.18}$$

In this case, we should choose $\alpha = p-1$. If $1 \leq \alpha < p-1$, then when $0 \leq m \leq \alpha$,

$$f_{\min} = 1, \quad \text{and} \quad f_{\max} = \frac{\alpha^{[\alpha]}}{([\alpha])!}; \tag{3.19}$$

when $\alpha < m \leq p-1$,

$$f_{\min} = \frac{\alpha^{p-1}}{(p-1)!}, \quad \text{and} \quad f_{\max} = \frac{\alpha^{[\alpha]}}{([\alpha])!}. \tag{3.20}$$

Comparing these two cases, it is easy to see that the proper value of α should be $1 \leq \alpha < p-1$ and we need to select α such that

$$\min_{\alpha}(\max(\frac{\alpha^\alpha}{\alpha!}, \frac{\alpha^\alpha (p-1)!}{\alpha^{p-1}\alpha!})). \tag{3.21}$$

Using Stirling's formula,

$$\frac{\alpha^\alpha}{\alpha!} \approx \frac{\alpha^\alpha e^\alpha}{(2\pi)^{0.5}\alpha^{\alpha+0.5}} = \frac{e^\alpha}{(2\pi\alpha)^{0.5}}; \tag{3.22}$$

and

$$\frac{\alpha^\alpha (p-1)!}{\alpha^{p-1}\alpha!} \approx \frac{\alpha^\alpha (2\pi)^{0.5}(p-1)^{p-1+0.5}e^\alpha}{\alpha^{p-1}(2\pi)^{0.5}\alpha^{\alpha+0.5}e^{p-1}} = \sqrt{\frac{p-1}{\alpha}}\left(\frac{p-1}{\alpha e}\right)^{p-1} e^\alpha. \tag{3.23}$$

So when $\frac{p-1}{\alpha e} \approx 1$, we would achieve our objective, that is, when

$$\alpha \approx \frac{p-1}{e}, \tag{3.24}$$

the magnitude of the nonzero entries of $x(\alpha)$ and $u(\alpha)$ are about the closest.

This can provide us an initial value for the proper value of α. We obtaqin the best α by searching in this vicinity. For each fixed p, we can pre-compute and test a few values of α to build a look-up table containing the best values of $\alpha(p)$ that achieve numerical stability.

We would also like to note that the modification does not have much effect on the complexity of the algorithm: once p is known, we can select an α from the look-up table and compute $c_p(\alpha)$ and the FFT of $u(\alpha)$ and store it before we start the computation of the matrix-vector product. The vectors $x(\alpha)$ and $v_1(\alpha)$ can be computed from $c_p(\alpha)$. Note that if the multiplication is to be done with several vectors, the FFT of $u(\alpha)$ only needs to be computed once. This reduces the number of FFTs to two each time, which naturally speeds up the multiplication even further.

This technique will eventually stop working since the entries will be of very different magnitudes with very large number p even by utilizing this technique. We suggest a technique to address the stability problem in case of high precision requirements and for large p. In this case the corresponding Toeplitz matrix could be subdivided into blocks of smaller Toeplitz matrices, each of which consists elements of similar magnitude, therefore the above method could be used *on each block* to stabilize the computation with the same complexity. Furthermore, if we view each Toeplitz block as one entry in the whole matrix, the whole matrix is a Toeplitz matrix again as the following,

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & A_0 & A_{-1} & A_{-2} & A_{-3} & A_{-4} & A_{-5} & \cdots \\ \cdots & A_1 & A_0 & A_{-1} & A_{-2} & A_{-3} & A_{-4} & \cdots \\ \cdots & A_2 & A_1 & A_0 & A_{-1} & A_{-2} & A_{-3} & \cdots \\ \cdots & A_3 & A_2 & A_1 & A_0 & A_{-1} & A_{-2} & \cdots \\ \cdots & A_4 & A_3 & A_2 & A_1 & A_0 & A_{-1} & \cdots \\ \cdots & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{3.25}$$

where $\{A_i, i = ..., -2, -1, 0, 1, 2...\}$ are Toeplitz matrices. Therefore the fast algorithm could be applied to each of the individual matrix as well as to the whole matrix.

Since for the transpose matrix PU of the lower triangular Pascal matrix and the symmetric Pascal matrix PS, the corresponding decompositions are very similar, the process to introduce the parameter α is the same. For the matrix, PU, we have

$$PU(\alpha) = \mathrm{diag}(v_3) \cdot Toep\,[c_p, {r_p}'] \cdot \mathrm{diag}(v_4), \tag{3.26}$$

where

$$v_3(\alpha) = \begin{bmatrix} 1 & -\frac{\alpha}{1} & \frac{\alpha^2}{2} & -\frac{\alpha^3}{6} & ... & \frac{\alpha^{p-1}}{(p-1)!} \end{bmatrix}', \tag{3.27}$$

$$v_4(\alpha) = \begin{bmatrix} 1 & -\frac{1}{\alpha} & \frac{2}{\alpha^2} & -\frac{6}{\alpha^3} & ... & \frac{(p-1)!}{\alpha^{p-1}} \end{bmatrix}', \tag{3.28}$$

$$c_p(\alpha) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}', \tag{3.29}$$

$$r_p(\alpha) = \begin{bmatrix} 1 & \frac{\alpha}{1} & \frac{\alpha^2}{2} & \frac{\alpha^3}{6} & ... & \frac{\alpha^{p-1}}{(p-1)!} \end{bmatrix}; \tag{3.30}$$

p	Naive algorithm	Stabilized algorithm	Stabilizing α	α from Eq. (3.24)
6	$2.5352e-016$	$1.8608e-016$	1.15	1.84
9	$4.5658e-015$	$5.0705e-016$	2	2.94
12	$1.2296e-012$	$1.3944e-015$	2.2	4.05
15	$1.4068e-010$	$2.3761e-015$	3.9	5.15
18	$4.3558e-008$	$1.2296e-014$	3.95	6.25
21	$7.2048e-005$	$4.9564e-014$	4.7	7.36
24	0.2388	$1.4088e-013$	5.1	8.46
27	262.46	$2.5018e-013$	5.8	9.56
30	$7.0563e+005$	$3.8519e-013$	6.4	10.67
33	$8.4567e+008$	$2.0082e-012$	6.15	11.77
36	$8.6394e+012$	$6.9394e-012$	7.1	12.88

TABLE 4.1

Accuracy comparison of the naive algorithm and the stabilized algorithm. The initial guess is seen to overestimate the correct value.

and for the matrix, PS, we have

$$PS(\alpha) = \mathrm{diag}(v_5) \cdot I_p \cdot Toep[c_p, r_p'] \cdot \mathrm{diag}(v_6), \tag{3.31}$$

where

$$v_5(\alpha) = \left[\begin{array}{ccccccc} 1 & \frac{\alpha}{1} & \frac{\alpha^2}{2} & \frac{\alpha^3}{6} & \ldots & \frac{\alpha^{p-1}}{(p-1)!} \end{array} \right]', \tag{3.32}$$

$$v_6(\alpha) = \left[\begin{array}{ccccccc} -\frac{\alpha}{1} & \frac{\alpha^2}{1} & -\frac{\alpha^3}{2} & \ldots & \frac{\alpha^p}{(p-1)!} \end{array} \right]', \tag{3.33}$$

$$c_p(\alpha) = \left[\begin{array}{ccccccc} \frac{(p-1)!}{\alpha^p} & \frac{(p-2)!}{\alpha^{p-1}} & \frac{(p-3)!}{\alpha^{p-2}} & \ldots & \frac{1}{\alpha^2} & \frac{1}{\alpha} \end{array} \right]', \tag{3.34}$$

$$r_p(\alpha) = \left[\begin{array}{ccccccc} \frac{(p-1)!}{\alpha^p} & \frac{p!}{\alpha^{p+1}} & \frac{(p+1)!}{\alpha^{p+2}} & \ldots & \frac{(2p-3)!}{\alpha 2^{p-2}} & \frac{(2p-2)!}{\alpha^{2p-1}} \end{array} \right]. \tag{3.35}$$

We can also implement similarly the above Toeplitz block technique on these two Pascal matrices.

4. Numerical results. In this section we demonstrate with examples and numerical experiments the effectiveness of the stabilizing technique and the accuracy of the algorithm.

In formula (3.10), we introduced a parameter α to stabilize the algorithm to compute the product of a lower triangular Pascal matrix and a vector. In our first experiment, we set $p = 20$, that is, the lower triangular Pascal matrix is of size 20×20, and the multiplying vector is a vector randomly generated with uniform distribution. The direct matrix-vector product is used as the true value (As mentioned before, (2.10) provides a given test where an exact result is available). Then compute the product with our new proposed algorithm for each α that runs from one to ten with increase of 0.05 each step. The result is plotted in Fig. 4.1. It shows that as α changes, the accuracy of the algorithm changes. It also shows that choosing a proper value of α is necessary to obtain accurate result.

In our second experiment, we compare the matrix-vector product with the naive algorithm (without parameter α) and the stabilized algorithm (with proper value of parameter α). We let the size of the Pascal matrix changes, and all vectors of different sizes are uniformly randomly generated. We record the maximum relative errors of

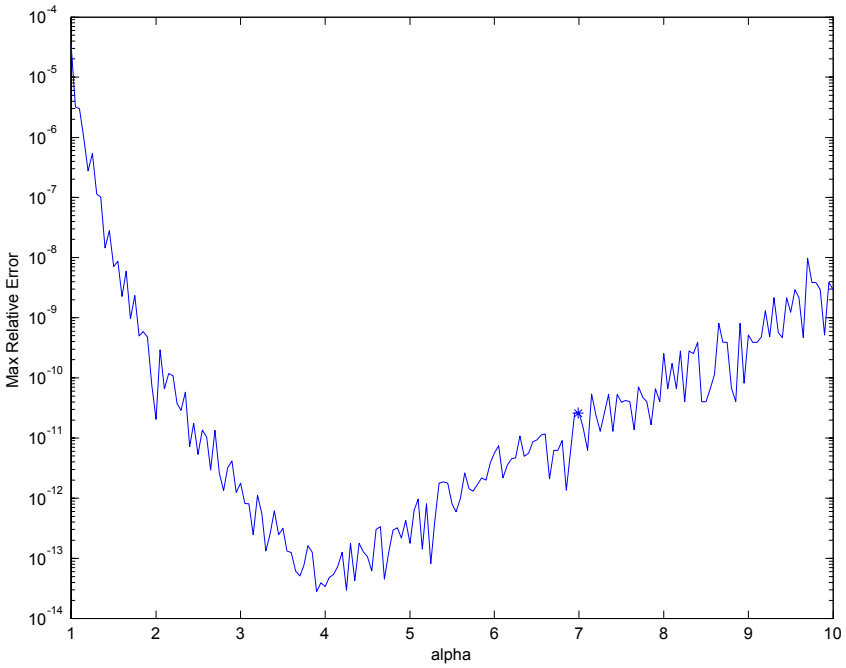

FIG. 4.1. *Stabilization of the computations using the scaling parameter α. The guess value for α according to equation (3.24) is 7.*

p	Stabilized algorithm
25	$2.2881e-013$
50	$1.7356e-013$
75	$6.1541e-014$
100	$2.3015e-013$
125	$2.6873e-013$
150	$1.3628e-013$
200	$2.6536e-013$

TABLE 4.2
Accuracy achieved by the block stabilized algorithm. The block size is taken to be 25×25.

the naive algorithm and the stabilized algorithm, and the values of α used in the stabilized algorithm in Table 4.1. It shows that when p is smaller than 9, there is no need to introduce the stabilizing parameter α; when p gets bigger, it is necessary to use the stabilizing parameter α to obtain accurate results. It also shows that as p getting bigger, the stabilized algorithm also gradually lose precision. This leads us to apply the block technique to maintain accuracy as p grows even bigger. In the next experiment, we increase the value of p and at the same time try to maintain the maximum relative error at 10^{-13}, so we choose the block size to be 25×25. We list the experimental result in Table 4.2. It is clear that the block technique is able to provide the desired accuracy.

5. Generalization of Pascal matrices. As described in [Zhang97, Call93], the lower triangular Pascal matrix PL can be generalized as follows,

$$P_{ij}[z] = \begin{cases} z^{i-j}C_{i-1}^{j-1} & \text{if } i \geq j \\ 0 & \text{if } i < j \end{cases} . \tag{5.1}$$

It is easy to see that it can be decomposed as the following,

$$P[z] = diag \begin{bmatrix} 1 \\ z \\ z^2 \\ z^3 \\ \vdots \\ z^{p-1} \end{bmatrix} \cdot PL \cdot diag \begin{bmatrix} 1 \\ z^{-1} \\ z^{-2} \\ z^{-3} \\ \vdots \\ z^{1-p} \end{bmatrix} . \tag{5.2}$$

The product of a generalized Pascal matrix and a vector can be computed efficiently similar to that of a lower triangular Pascal matrix and a vector.

From [Call93], we know that

$$P[x]P[y] = P[x+y]. \tag{5.3}$$

It is obvious that

$$P[0] = I. \tag{5.4}$$

Therefore, we have (for proof, see [Call93])

$$P[-x] = P[x]^{-1}. \tag{5.5}$$

When $x = 1$, we have

$$P[-1] = P[1]^{-1}, \tag{5.6}$$

Combine with

$$P[1] = PL, \tag{5.7}$$

that is,

$$PL^{-1} = P[-1]. \tag{5.8}$$

Consider that we have

$$PU = PL' \text{ and } PS = PL * PU, \tag{5.9}$$

then

$$PL^{-1} = diag \begin{bmatrix} 1 \\ -1 \\ 1 \\ \vdots \\ (-1)^{p-1} \end{bmatrix} * PL * diag \begin{bmatrix} 1 \\ -1 \\ 1 \\ \vdots \\ (-1)^{p-1} \end{bmatrix} \tag{5.10}$$

$$PU^{-1} = P[-1]' = diag \begin{bmatrix} 1 \\ -1 \\ 1 \\ \vdots \\ (-1)^{p-1} \end{bmatrix} * PU * diag \begin{bmatrix} 1 \\ -1 \\ 1 \\ \vdots \\ (-1)^{p-1} \end{bmatrix}, \tag{5.11}$$

and

$$PS^{-1} = P[-1]' * P[-1] = diag \begin{bmatrix} 1 \\ -1 \\ 1 \\ \vdots \\ (-1)^{p-1} \end{bmatrix} * PU * PL * diag \begin{bmatrix} 1 \\ -1 \\ 1 \\ \vdots \\ (-1)^{p-1} \end{bmatrix}. \tag{5.12}$$

We have developed fast algorithms to compute the matrix-vector product for three different types of Pascal matrices. With the fast algorithm to compute the product of a generalized Pascal matrix and a vector, we can also compute efficiently the matrix-vector product for matrices including all inverses of these three Pascal matrices.

6. Conclusion. We have presented matrix decompositions of the Pascal matrices. Based on these decompositions, we have developed fast algorithms for computing the matrix-vector product of a Pascal matrix and a vector and provide techniques to stabilize the algorithms. We have also presented some interesting properties of the generalized Pascal matrices. Our algorithms have been shown to be stable and $O(p \log p)$.The algorithms have already been applied to create fast translation algorithms for the fast multipole method [Tang03].

REFERENCES

[Aggarwala2001] Rita Aggarwala and Michael P. Lamoureux, *Inverting the Pascal Matrix Plus One*, American Math. Monthly, 109, April 2001, pp371-377.

[Bai2000] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, editors, *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.

[Biolkova99] V. Biolkova and D. Biolek, *Generalized Pascal Matrix of First-Order S-Z Transforms*, in ICECS'99 Pafos, Cyprus 1999, pp. 929-931.

[Brawer92] Robert Brawer and Magnus Pirovino, *The Linear algebra of the Pascal matrix*, Linear Algebra and Its Applications, 174, 1992, pp. 13-23.

[Call93] Gregory S. Call and Daniel J. Velleman, *Pascal's matrices*, American Math. Monthly, 100, April, 1993, pp. 372-376.

[Cooley65] J. W. Cooley and J. W. Tukey *An algorithm for the machine calculation of complex Fourier series*, Math Comp. 19, 1965, pp. 297-301.

[Driscoll94] James R. Driscoll and Dennis M. Healy,Jr., *Computing Fourier transforms and convolutions on the 2-sphere*, Advances in Applied Mathematics, 15, 1994, pp.202-250.

[Driscoll97] J. R. Driscoll, D. M. Healy, JR., and D. N. Rockmore, *Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs*, SIAM J. COMPUT. 26(4), 1997, pp1066-1099.

[Edelman03] Alan Edelman and Gilbert Strang, *Pascal matrices*, MIT.

[Edwards02] A. W. F. Edwards, *Pascal's Arithmetical Triangle: The story of a mathematical idea*, Johns Hopkins University Press, 2002.

[GohbergL94] I. Gohberg and V. Olshevsky, *Complexity of multiplication with vectors for structured matrices*, Linear Algebra Appl., 202(1994), pp. 163-192.

[GohbergC94] I. Gohberg and V. Olshevsky, *Fast algorithms with preprocessing for matrix-vector multiplication problems*, J. Complexity, 10(4)(1994), pp. 411-427.

[Golub96] G. Golub and C. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, third edition, 1996.

[Kailath99] T. Kailath and A.H. Sayed, editors, *Fast Reliable Algorithms for Matrices with Structure*,SIAM, Philadelphia, 1999.

[Lu98] Hao Lu, *A generalized Hilbert matrix problem and confluent Chebyshev-Vandermonde system*, SIAM, J. MATRIX ANAL. APPL. 19(1), Jan. 1998, pp. 253-276.

[Moore93] Sean S. B. Moore, Dennis M. Healy, Jr., and Daniel N. Rockmore, *Symmetry stabilization for fast discrete monomial transforms and polynomial evaluation*, Linear Algebra Appl., 192, 1993, pp. 249-299.

[Pan92] Victor Pan, *Complexity of computations with matrices and polynomials*, SIAM Review, 34(2), June 1992, pp.225-262.

[Stein71] E. Stein and G. Weiss, *Fourier analysis on Euclidean Spaces*, Princeton University Press, Princeton, NJ, 1971.

[Tang03] Zhihui Tang, *Fast transforms based on structured matrices with applications to the fast multipole method*, University of Maryland at College Park, Ph.D. Thesis, 2003.

[Van92] Charles Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.

[Zhang97] Zhizheng Zhang, *The Linear algebra of the generalized Pascal matrix*, Linear Algebra and Its Applications, 250, 1997, pp. 51-60.

[Zhang98] Zhizheng Zhang and Maixue Liu, *An extension of the generalized Pascal matrix and its algebraic properties*, Linear Algebra and Its Applications, 271, 1998, pp. 169-177.