AN ACCURATE COMPUTATIONAL SOLUTION OF TOTALLY POSITIVE CAUCHY LINEAR SYSTEMS

Abdoulaye Cissé
Computer Science Department
St. Cloud State University
St. Cloud, MN 56301
ciab0201@stcloudstate.edu

Andrew Anda
Computer Science Department
St. Cloud State University
St. Cloud, MN 56301
aanda@stcloudstate.edu

Abstract

The Cauchy linear systems $C(x,y)V = b$ where $C(x,y) = \left(\frac{1}{x_i - y_j}\right)$, $x = (x_i)$, $y = (y_i)$, $V = (v_i)$, $b = (b_i)$ can be solved very accurately regardless of condition number using Björck-Pereyra-type methods. We explain this phenomenon by the following facts. First, the Björck-Pereyra methods, written in matrix form, are essentially an accurate and efficient bidiagonal decomposition of the inverse of the matrix, as obtained by Neville Elimination with no pivoting. Second, each nontrivial entry of this bidiagonal decomposition is a product of two quotients of (initial) minors. We use this method to derive and implement a new method of accurately and efficiently solving totally positive Cauchy linear systems in time complexity $O(n^2)$.

Keywords: Accuracy, Björck-Pereyra-type methods, Cauchy linear systems, floating point architectures, generalized Vandermonde linear systems, Neville elimination, totally positive, LDU decomposition.

Note: In this paper we assume that the nodes x_i and y_j are pair wise distinct and C is totally positive

1 Background

Definition 1.1

A Cauchy matrix is defined as:

$$C(x,y) = \left(\frac{1}{x_i - y_j} \right), \quad for \ \ i,j = 1,2,...,n \tag{1.1}$$

where the nodes x_i and y_j are assumed pair wise distinct. *C(x,y)* is totally positive (TP) if

$$0 < y_1 < y_2 < \ldots < y_{n-1} < y_n < x_1 < x_2 < \ldots < x_{n-1} < x_n .$$

A matrix $\hat{C}(x,y) = (\hat{C}_{ij})_{i,j=1}^{n}$ is said to be of Cauchy-type if its entries are of the form

$$\hat{C}_{ij} = \frac{h_i f_j}{x_i - y_j}, \quad for \ \ i,j = 1,2,...,n \tag{1.2}$$

where the nodes satisfy the same conditions as in (1.1), and h_i, f_i are no vanishing scalars. For $U = diag(h_1, h_2,\ldots, h_n)$ and $V = diag(f_1, f_2, \ldots, f_n)$, $\hat{C}(x,y)$ can be factored according to

$$\hat{C}(x,y) = UC(x,y)V \tag{1.3}$$

where *C(x,y)* is defined as in (1.1).

Proposition 1.1

If fl(x) is the representable floating point number adjacent to *x* then for all real number *x*,

$$fl(x) = x(1 + \delta) \tag{1.4}$$

where $|\delta| < \varepsilon$ and ε is the unit round off, called machine precision.

A proof of this proposition is available in [7] p.38

1-1 The analysis of errors

A forward error analysis is the error analysis in which one keeps track of the errors as they are introduced. On the other hand a backward error analysis is the error analysis in which one takes the point of view that the computed solution to the problem must be the exact solution of a perturbed problem. To be more explicit, suppose that an approximation $\hat{y}$ to $y = f(x)$ is computed in an arithmetic of precision ε, where f is a real scalar function of a real scalar variable. How should we measure the "quality" of $\hat{y}$? In most computations we would be happy with a tiny relative error, $E_{rel}(\hat{y}) \approx \varepsilon$, but this cannot always be achieved. Instead, by focusing on the relative error of $\hat{y}$ we can ask, "for what set of data have we actually solved our problem?" That is, for what Δx do we have $\hat{y} = f(x + \Delta x)$? In general, there may be many such Δx, so we should ask for the smallest one. The value $|\Delta x|$ (or min $|\Delta x|$), possibly divided by $|x|$, is called the backward error. The absolute and relative errors of $\hat{y}$ are called forward errors, to distinguish them from the backward errors.

To illustrate backward error analysis, suppose we are solving the set of linear algebraic equations

$$Ay = b \tag{1.5}$$

Suppose we obtain the computed solution $\hat{y}$ and let h be the error vector

$$h = \hat{y} - y$$

If we take the point of view that $\hat{y} = y + h$ is the exact solution of a perturbation of the system (1.5) where the perturbation is in b then we may write

$$A(y + h) = b + k, \tag{1.6}$$

where k is the perturbation vector.

Wilkinson (1963) gives the following bound (we assume that the vector matrix norms used in this treatise are consistent, i.e. $\|Ay\| \leq \|A\| \cdot \|y\|$) on the relative error in y as a function of the relative error in b.

$$\|h\| / \|y\| \leq N \, \|k\| / \|b\| \tag{1.7}$$

where $\quad N = \|A\| \cdot \|A^{-1}\|$

is called the condition number for the system (1.5).

1-2 Cancellation

Cancellation is what happens when two nearly equal numbers are subtracted. It is often, but not always, a bad thing. [7]

Consider the function $f(x) = (1 - \cos x)/x^2$. With $x = 1.2 \times 10^{-5}$ the value of $\cos x$ rounded to *10* significant figures is $c = 0.9999\,9999\,99$, so that $1 - c = 0.\,0000\,0000\,01$. Then $(1 - c)/x^2 = 10^{-10}/(1.44 \times 10^{-10}) = 0.6944\ldots$, which is clearly wrong given the fact that $0 \leq f(x) \leq ½$ for all $x \neq 0$. A *10-significant-figure* approximation to $\cos x$ is therefore not sufficient to yield a value of $f(x)$ with even one correct figure. The problem is that $1 - c$ has only *1* significant figure. The subtraction $1 - c$ is exact, but this subtraction produces a result of the same size as the error in c. In the other words, subtraction elevates the importance of the earlier error. In this particular example it is easy to rewrite $f(x)$ to avoid the cancellation. Since $\cos x = 1 - 2\sin^2(x/2)$,

$$f(x) = \frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2.$$

Evaluating this second formula for $f(x)$ with a *10-significant-figure* approximation to $\sin(x/2)$ yields $f(x) = 0.5$, which is correct to *10* significant figures.

It is important to realize that cancellation is not always a bad thing. There are several reasons. First, the numbers being subtracted may be error free, as when they are from initial data that is known exactly. Second, the effect of cancellation depends on the role that the result plays in the remaining computation. For example, if $x \gg y \approx z > 0$ then the cancellation in the evaluation of $x + (y - z)$ is harmless.

1-3 Ill conditioned problems

Let consider a mapping of the form

$$f: D \subset R^p \rightarrow R^q$$
where R is the set of real numbers. The p components of a vector $d \in D$ are the data that determine the problem, and the q components of the vector $f(d)$ constitute the solution to the problem. An interesting mathematical question to ask about any problem is how sensitive is the solution $f(d)$ to small perturbations in d?
Let δ be a small perturbation in d. The sensitivity question now becomes how much does $f(d + \delta)$ differ from $f(d)$? To put it in more mathematical terms, suppose $\|\cdot\|$ is a vector norm and suppose $\|\delta\|$ is a small positive number. Can we assume that $\|f(d + \delta) - f(d)\|$ is also a small positive number? When the answer is yes, the solution is not sensitive to small perturbations in the data. Such a problem is called well conditioned. However, for some problems the answer is no, in which case the solution is sensitive to small perturbations in the data. Such a problem is called ill conditioned.

A system of equations ($Ay = b$) is said to be ill-conditioned if a relatively small change in the coefficients of matrix A causes a relatively large change in the solution. Conversely a system of equations ($Ay = b$) is said to be well-conditioned if a relatively small change in the coefficients of matrix A causes a relatively small change in the solution.

In addition to its interpretation as the perturbation vector in (1.6), the vector k can be interpreted as the residual vector
$$k = A\hat{y} - b$$
However, $\hat{y}$ can be a poor approximation to y even with a relatively small residual vector k. This illustrates the well-known fact that for ill-conditioned problems, a poor approximate solution $\hat{y}$ can give a small residual vector k. Consequently, the size of the residual vector is not always a good test of how good a computed solution might be. It is inequality (1.7) which provides the clue for understanding what is happening here. If the condition number N is small, then a small perturbation vector k guarantees that h is small resulting in $\hat{y}$ being a good approximation to the true solution y. However, if N is large, we cannot be certain. Certainly if $\hat{y}$ is a poor approximation to x, that is, if h is large, than N will have to be large; but if N is large, it does not necessary follow that h is large nor that $\hat{y}$ is a poor approximation to y.

1-4 Model of arithmetic used

We will use a standard model of floating point arithmetic. We assume that the basic arithmetic operations $op = +, -, *, /$ satisfy
$$fl(x \; op \; y) = (x \; op \; y)(1 + \delta) \quad |\delta| < \varepsilon \qquad (1.8)$$
that is, we assume the relative error of any arithmetic operation is small. Here, ε is the unit round off, called machine precision, which is typically of order 10^{-8} or 10^{-16} in single and double precision computer arithmetic, respectively. In IEEE standard floating point arithmetic, $\varepsilon=2^{-53} \cong 1.1\text{x}10^{-16}$ in double precision. In addition, we will assume that neither overflow nor underflow occur, since both can destroy the relative accuracy of the result. This floating point model implies that products, quotients, and sums of like-signed quantities can be computed accurately (i.e. with low relative error) but expressions involving cancellation may not be.

It should be noted that with this floating point model, the determinant of any standard Vandermonde matrix can be evaluated accurately for any floating point data x_i, since it only involves differences of input data and products. In other words we have a way to accurately evaluate any minor of TP standard Vandermonde matrix.

2 Cauchy Method

The problem of solving a Cauchy linear system with the special case of the right-hand side being $[1\ 1\ \dots\ 1]^t$ was considered in the year 1837 by Binet [2]. Four years later Cauchy solved the problem for a general right-hand side, arriving at his well-known formula [2]:

$$\det C = \frac{\prod_{j \succ k} (x_j - x_k) \prod_{j \prec k} (y_j - y_k)}{\prod_{j,k} (x_j - y_k)} \qquad 1 \le j, k \le n \qquad (2.1)$$

During Cauchy's time determinants were the main tool to express the solution of a linear system. Formula (2.1) provides an explicit expression for the entries of the solution $V=[v_1\ v_2 \dots v_n]^t$ for a Cauchy linear system with right-hand side $W=[w_1\ w_2 \dots w_n]^t$:

$$v_j = \frac{f(y_i)}{g'(y_i)} \left(\sum_{i=1, j=1}^{n} \frac{w_i}{x_j - y_i} \right) \frac{g(x_j)}{f'(x_j)} \qquad 1 \le i, j \le n \qquad (2.2)$$

with $f(x) = \prod_{i=1}^{n} (x - x_i)$ and $g(x) = \prod_{i=1}^{n} (x - y_i)$

From $B = \left(\frac{1}{x_j - y_i} \cdot \frac{f(y_i)}{g'(y_i)} \cdot \frac{g(x_j)}{f'(x_j)} \right)_{1 \le i, j \le n}$ we have $V=BW$, therefore $C^{-1} = B$.

Also from (2.2) we have $f'(x_j) = \prod_{k=1, k \ne j}^{n} (x_j - x_k)$ and $g'(y_i) = \prod_{k=1, k \ne i}^{n} (y_i - y_k)$.

So a reformulation of (2.2) yields the formula:

$$C^{-1} = \left(\frac{1}{x_j - y_i} \cdot \frac{\prod_{k=1}^{n} (y_i - x_k)}{\prod_{k=1, k \ne i}^{n} (y_i - y_k)} \cdot \frac{\prod_{k=1}^{n} (x_j - y_k)}{\prod_{k=1, k \ne j} (x_j - x_k)} \right)_{1 \le i, j \le n} \qquad (2.3)$$

Since there is no subtractive cancellation in (2.3), the relative accuracy will be preserved and an accurate solution of the linear system $CV=W$ can be computed in $O(n^3)$ by forming $V = C^{-1}W$ using the following algorithm in MATLAB syntax and semantics.

Algorithm 2.1 (Solving Cauchy Systems using Cauchy method) If C is a Cauchy matrix determined by the vectors $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$, the following algorithm computes the solution to $Cv = w$

```
10 function cauchy([$x_1, x_2, \ldots, x_n$], [$y_1, y_2, \ldots, y_n$], [$b_1, b_2, \ldots, b_n$])
20 for i = 1 : n
30     for j = 1 : n
40         p = 1
50         q = 1
60         t = 1
70         for k = 1 : n
80             $p = p \cdot (x_k - y_i) \cdot (x_j - y_k)$
90             if $k \neq i$
100                $q = q \cdot (y_k - y_i)$
110            if $k \neq j$
120                $t = t \cdot (x_j - x_k)$
130            $s_{ij} = p/(q \cdot t \cdot (x_j - y_i))$
140 for i = 1 : n
150     sum = 0
160     for j = 1 : n
170         $sum = sum + s_{ij} \cdot w_j$
180     $v_i = sum$
190 return v
```

3 Bidiagonal decomposition of inverses of TP matrices

3-1 Neville elimination

Neville elimination is similar to the process of Gaussian elimination except that the zeros in columns are created by subtraction of a multiple of row i from row $i+1$ instead of subtracting a multiple of a fixed row from all the other rows below it. For a nonsingular matrix A of order n, the Neville elimination procedure consists of $n-1$ successive steps, resulting in a sequence of matrices as follows:
$A = \tilde{A}^{(1)} \to A^{(1)} \to \tilde{A}^{(2)} \to A^{(2)} \to \ldots \to \tilde{A}^{(n)} = A^{(n)} = U$ where U is an upper triangular matrix. For each k, $1 \leq k \leq n$, the matrix $A^{(k)} = (a_{ij}^{(k)})_{1 \leq i,j \leq n}$ has zeros below its main diagonal in the first $k-1$ columns, and one also has the property that

$$\text{If } a_{ik}^{(k)} = 0, \text{ for some } i \geq k, \text{ then } a_{hk}^{(k)} = 0 \text{ for all } h \geq i \qquad (3.1)$$

$A^{(k)}$ is obtained from the matrix $\tilde{A}^{(k)}$ by moving the rows with a zero entry in the column k to the bottom, if necessary. In order to get (3.1) the rows are moved and placed with the same relative order as they appear in $A^{(k)}$. To get $A^{(k+1)}$ we produce zeros in the column k below the main diagonal by subtracting a multiple of row i from row $i+1$ *for* $i = n-1, n-2, \ldots, k$, according to the following formula:

$$\tilde{a}_{ij}^{(k+1)} = \begin{cases} a_{ij}^{(k)} & \text{if } i \le k \\ a_{ij}^{(k)} - \dfrac{a_{ik}^{(k)}}{a_{i-1,k}^{(k)}} a_{i-1,j}^{(k)} & \text{if } i \ge k+1 \text{ and } a_{i-1,k}^{(k)} \ne 0 \\ a_{ij}^{(k)} & \text{if } i \ge k+1 \text{ and } a_{i-1,k}^{(k)} = 0 \end{cases} \tag{3.2}$$

When A is a nonsingular TP matrix, no row exchanges are needed so $\tilde{A}^{(k)} = A^{(k)}$ *for all k.*

3-2 Accurate and Efficient Computation of TP generalized Vandermonde matrices using a Schur function

Most of the results in this section are from Koev's dissertation [9].

Definition 3.1

Let consider $G = [x_i^{a_j}]_{1\le i,j\le n}$ be a TP generalized Vandermonde matrix where $0 \le a_1 < a_2 < \ldots < a_n$ are integers and $V = [x_i^{j-1}]_{i,j=1}^{n}$ is a standard Vandermonde matrix. We define the partition λ associated with G as the nonincreasing sequence of nonnegative integers $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$, where $\lambda_i = a_{n+1-i} - (n-i)$. A Schur function corresponding to a partition λ is defined as:

$$S_\lambda(x_1, x_2, \ldots, x_m) = \frac{\det\left(\left[x_i^{j-1+\lambda_{m-j}}\right]_{i,j=1}^{m}\right)}{\det\left(\left[x_i^{j-1}\right]_{i,j=1}^{m}\right)} \tag{3.3}$$

This Schur function is the ratio of the determinant of a generalized Vandermonde matrix corresponding to a partition λ and the determinant of a standard Vandermonde matrix. In his paper Koev describes a new approach to compute the Schur function to high relative accuracy [9]. In fact we need to compute the function accurately because we cannot compute *det(G)* accurately using traditional algorithms. The need for the Schur function is to accurately compute all the needed minors of G needed to obtain an accurate bidiagonal decomposition of G^{-1} and needed by the new Bjröck-Pereyra-type algorithm for the accurate solution of TP generalized Vandermonde linear systems.

3-2-1 Bidiagonal decomposition of inverses of TP matrices

The following result is due to Gasca and Pena [4].

Proposition 3.1

If A is TP, then $A^{-1} = U^{(1)}U^{(2)}\ldots U^{(n-1)}D^{-1}L^{(n-1)}L^{(n-2)}\ldots L^{(1)}$, where $L^{(k)}$ and $U^{(k)}$, $k=1,2,\ldots,n$ are unit lower and upper bidiagonal matrices respectively with the property that $L_{i+1,i}^{(k)} = 0$

and $U^{(k)}_{i+1,i} = 0$ *for* $i \leq k\text{-}1$, and D^{-1} is a diagonal matrix. Each entry of $L^{(k)}$ and $U^{(k)}$ is a negated product of two quotients of minors of A. Each entry of D is a quotient of minors of A.

The proof of this proposition is based on the process of Neville elimination and the general form of the LDU decomposition of A, where L and U are unit lower and unit upper triangular respectively. More details can be found in [9]. However the following notation will be convenient. For $\alpha_1, \beta_1, \alpha_2, \beta_2$ natural numbers such as $1\leq \alpha_1\leq \alpha_2 \leq n$, $1\leq \beta_1 \leq \beta_2 \leq n$ and $m - 1 = \alpha_1 - \alpha_2 = \beta_1 - \beta_2$, we denote by $A(\alpha_1{:}\alpha_2, \beta_1{:}\ \beta_2)$ the m x m submatrix of A where A is an n x n real matrix. We have the following results from the proof of proposition 3.1; *for* $k = 1,2,\ldots,n$ *and* $i \geq k$

$$L^{(k)}_{i+1,i} = -\frac{a^{(k-1)}_{i+1,k}}{a^{(k-1)}_{i,k}} = -\frac{\det(A(i-k+2:i+1,1:k))}{\det(A(i-k+2:i,1:k-1))}\cdot\frac{\det(A(i-k+1:i-1,1:k-1))}{\det(A(i-k+1:i,1:k))} \tag{3.4}$$

$$U^{(k)}_{i,i+1} = -\frac{a^{(k-1)}_{k,i+1}}{a^{(k-1)}_{k,i}} = -\frac{\det(A(1:k,i-k+2:i+1))}{\det(A(1:k-1,i-k+2:i))}\cdot\frac{\det(A(1:k-1,i-k+1:i-1))}{\det(A(1:k,i-k+1:i))} \tag{3.5}$$

$$D_{ii} = \frac{\det(A(1:i,1:i))}{\det(A(1:i-1,1:i-1))}\ \textit{for } i = 2,\ldots,n \text{ and } \quad D_{11} = a_{11} \tag{3.6}$$

Note that *for* $k=1$, $L^{(1)}_{i+1,i} = -\dfrac{a_{i+1,1}}{a_{i,1}}$ and $U^{(1)}_{i,i+1} = -\dfrac{a_{1i+1}}{a_{1i}}$

For the TP Vandermonde matrices, the decomposition yields the Björck-Pereyra algorithms.

3-2-2 Björck-Pereyra-Type algorithm for TP generalized Vandermonde Matrices

In this section we describe an accurate bidiagonal decomposition of the inverse of a generalized Vandermonde matrix.

Let $G = [x_i^{j-1+\lambda_{n-j+1}}]_{1\leq i,j\leq n}$ be a TP generalized Vandermonde matrix corresponding to a partition $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ where $0 < x_1 < x_2 < \ldots < x_n$, and let $V = [x_i^{j-1}]^{n}_{i,j=1}$ be a standard Vandermonde matrix. For any $i \leq j$, $k \leq l \in \{1,2,\ldots,n\}$ such that $j\text{-}i = l\text{-}k$ we have from (3.3) that $det(G_{i:j,k:l}) = det(V_{i:j,1:l-k+1}).s_{(\lambda_{n-l+1}, \lambda_{n-l+2}, \ldots, \lambda_{n-k+1})}(x_i, x_{i+1}, \ldots, x_j).\prod_{s=i}^{j} x_s^{k-1}$.

Now for the bidiagonal decomposition of G^{-1}, we have from (3.4), (3.5), (3.6) *for* $k = 2,\ldots,n$ *and* $i \geq k$

$$L^{(k)}_{i+1,i} = -\frac{s_{(\lambda_{n-k+1:n})}(x_{i-k+2:i+1})}{s_{(\lambda_{n-k+2:n})}(x_{i-k+2:i})}\cdot\frac{s_{(\lambda_{n-k+2:n})}(x_{i-k+1:i-1})}{s_{(\lambda_{n-k+1:n})}(x_{i-k+1:i})}\cdot\prod_{j=i-k+1}^{i-1}\frac{x_{i+1}-x_{j+1}}{x_i - x_j} \tag{3.7}$$

$$D^{-1}_{ii} = \frac{s_{(\lambda_{n-i+2:n})}(x_{1:i-1})}{s_{(\lambda_{n-i+1:n})}(x_{1:i})}\cdot\prod_{j=1}^{i-1}\frac{1}{x_i - x_j} \tag{3.8}$$

$$U_{i,i+1}^{(k)} = - x_k \cdot \frac{s_{(\lambda_{n-i:n-i+k-1})}(x_{1:k})}{s_{(\lambda_{n-i+1:n-i+k-1})}(x_{1:k-1})} \cdot \frac{s_{(\lambda_{n-i+2:n-i+k})}(x_{1:k-1})}{s_{(\lambda_{n-i+1:n-i+k})}(x_{1:k})} \qquad (3.9)$$

$L_{i+1,i}^{(1)} = - \left(\frac{x_{i+1}}{x_i} \right)^{\lambda_n}$, **Error! Objects cannot be created from editing field codes.**=- $(x_1)^{1+\lambda_{n-i}-\lambda_{n-i+1}}$ and $D_{11}^{-1} = (x_1)^{-\lambda_n}$

This bidiagonal decomposition of G^{-1} leads the solution of the TP generalized Vandermonde system $Gy = b$. The accuracy of this method takes place in the following proposition.

Proposition 3.2

If the relative error in every entry of $L^{(i)}$, $U^{(i)}$, D^{-1} is small (i.e. does not exceed a value $\eta = h(n)\varepsilon$ where $h(n)$ is a polynomial in n), $\hat{y}$ is the computed solution of $Gy = b$ and $\hat{G}$ is such that $\hat{G}\hat{y} = b$, then the following error bounds are valid:

1. $|y - \hat{y}| \leq O(nh(n))\varepsilon|G^{-1}||b|$
2. If b has alternating sign pattern (i.e. $(-1)^i f_i$ have the same sign for all i = 1,2,…,n) then solution is computed with a small componentwise relative forward error
$$|y - \hat{y}| \leq O(nh(n))\varepsilon|y|;$$
3. The componentwise relative backward error is small
$$|G - \hat{G}| \leq O(n^2h(n))\varepsilon|G| + O(\varepsilon^2)$$
4. The componentwise residual error is also small
$$|b - G\hat{y}| \leq O(n^2h(n))\varepsilon|G||\hat{y}| + O(\varepsilon^2)$$

A proof of this proposition is available in [1, 8].

4 Accurate solution of TP Cauchy linear system

For a TP matrix, the decomposition from Proposition 3.1 yields the well known Björck-Pereyra algorithm. As long as we can compute quotients of minor accurately and efficiently, the algorithms from Proposition 3.1 for solving $Ax = b$ are applicable to any TP matrix and the computed solutions will satisfy the same error bounds as the solutions for the Björck-Pereyra methods [9]. We propose a new fast $O(n^2)$ algorithm for accurately and efficiently solving TP Cauchy linear systems. In fact, Cauchy linear systems can be solved very accurately using a Björck-Pereyra algorithm this exemplifies how the exploitation of the structure allows not only for the speed-up of computation, but also for more accuracy in the computed solution than when using standard numerically methods. In section 3, we presented Koev's method [9] to obtain an accurate solution of TP generalized Vandermonde systems. The results indicate a close similarity between the numerical proprieties of Cauchy matrices and Vandermonde matrices. We exploit Koev's algorithm to derive a new algorithm for accurately solving, linear systems of equations with TP Cauchy matrices where the time of computation should be bounded by a polynomial function of the matrix dimension n; $O(n^2)$.

4-1 Theorem of V. Olshevsky [5]

Let $x = (x_i)_{i=1}^n$ and $y = (y_i)_{i=1}^n$ be vectors such that $\forall i,j\ x_i \neq y_j$ and $\forall i \neq j\ y_i \neq y_j$

If $h_i = \prod_{j=1}^{n} (x_i - y_j)$ and $f_i^{-1} = \prod_{j=1, j\neq i}^{n} (y_i - y_j)$ then $V(x) = \hat{C}(x,y).V(y)$

where $\hat{C}$ is a Cauchy-type matrix of the form $\hat{C}(x,y) = \left(\frac{h_i f_j}{x_i - y_j} \right)$ *for i,j = 1, 2, …,n* and *V(y)* and *V(x)* are polynomial Vandermonde matrices.

4-2 Partition and Schur function with standard Vandermonde matrix

In definition 3.1, to label the generalized Vandermonde matrix, we used the partition $\lambda = (\lambda_i)_{i=1}^n$ instead of $(a_i)_{i=1}^n$ where $\lambda_i = a_{n+1-i} - (n-i)$. For the standard Vandermonde matrix, $a_i = i-1$. Therefore $\lambda_i = 0\ \ \forall i$; in other words $\lambda = (0)$ for the standard Vandermonde matrix. Also if $\forall i\ \lambda_i = 0$ then $a_i = i-1$. Therefore, $S_{(0)}(x_1, x_2, \ldots, x_n) = 1\ \ \forall (x_1, x_2, \ldots, x_n)$. Consequently the expressions (3.7), (3.8), (3.9) yield immediately the following results:

$$L_{i+1,i}^{(k)} = - \prod_{j=i-k+1}^{i-1} \frac{x_{i+1} - x_{j+1}}{x_i - x_j}\ ,\quad L_{i+1,i}^{(1)} = -1 \tag{4.1}$$

$$D_{ii}^{-1} = \prod_{j=1}^{i-1} \frac{1}{x_i - x_j}\ ,\quad D_{11}^{-1} = 1 \tag{4.2}$$

$$U_{i,i+1}^{(k)} = -x_k\ ,\quad \forall k \tag{4.3}$$

From (4.1), (4.2) and (4.3), we deduce the following algorithm.

Algorithm 4.1 (Solving TP standard Vandermonde system) If *V* is a TP standard Vandermonde matrix determined by a vector $x = (x_1, x_2, \ldots, x_n)$ the following algorithm computes the solution of $Vy = b$.

```
10 function z=vand([$x_1, x_2, \ldots, x_n$], [$b_1, b_2, \ldots, b_n$])
20 d = ones(n)
30 for i = 2:n
40    for j = i-1:-1:1
50          $d_{ij} = d_{i,j+1}.(x_i - x_j)$
60 for k = 1: n-1
70          for i = n-1:-1:k
80             $L_{i+1,i}^{(k)} = \frac{d_{i+1,i-k+2}}{d_{i,i-k+1}}$
90       $b_{i+1} = b_{i+1} - b_i.L_{i+1,i}^{(k)}$
100 for i = 1:n
```

```
110   $D_{ii}^{-1} = \frac{1}{d_{i,1}}$
120   $b_i = b_i . D_{ii}^{-1}$
130 for k = n-1: -1:1
140       for i = k:n-1
150             $U_{i,i+1}^{(k)} = x_k$
160             $b_i = b_i - b_{i+1} . U_{i,i+1}^{(k)}$
170 return b
```

The following proposition is a restriction of proposition 3.2 to the standard Vandermonde systems

Corollary 4.1

If the relative error in every entry of $L^{(i)}$, $U^{(i)}$, D^{-1} is small (i.e. does not exceed a value $\eta = h(n)\varepsilon$ where $h(n)$ is a polynomial in n), $\hat{y}$ is the computed solution of TP Vandermonde system $Vy = b$ and $\hat{W}$ is such that $\hat{W}\hat{y} = b$, then the following error bounds are valid:

1. $|y - \hat{y}| \leq O(nh(n))\varepsilon|V^{-1}||b|$
2. If b has alternating sign pattern (i.e. $(-1)^i f_i$ have the same sign for all i = 1,2,…,n) then solution is computed with a small componentwise relative forward error
   $$|y - \hat{y}| \leq O(nh(n))\varepsilon|y|;$$
3. The componentwise relative backward error is small
   $$|V - \hat{W}| \leq O(n^2h(n))\varepsilon|V| + O(\varepsilon^2)$$
4. The componentwise residual error is also small
   $$|b - V\hat{y}| \leq O(n^2h(n))\varepsilon|V||\hat{y}| + O(\varepsilon^2)$$

The cost of the Algorithm 4.1 is $O(n^2)$ and the accuracy of the computed solution is justified by the Corollary 4.1. Indeed, when $0 < x_1 < x_2 < \ldots < x_n, V(x)$ is TP. The totally positive of the coefficients of $V(x)$ implies that the entries $L_{i+1,i}^{(k)}$ and D_{ii}^{-1} can be evaluated accurately in floating point model (1.8) since they involve only products and quotients of positive quantities. Also, it was observed in [2, 10] that if the components of the right-hand side are sign-alternating, i.e. $(-1)^i \cdot b_i$ have the same sign $(1 \leq i \leq n)$, then the solution will be computed with no subtractive cancellation with Algorithm 4.1.

4-3 Our new approach

Consider a Cauchy linear system $C(x,y)X = b$ where $C(x,y)$ is a TP Cauchy matrix.

Define $(h_i)_{i=1}^{n}$ and $(f_i)_{i=1}^{n}$ by $h_i = \prod_{j=1}^{n}(x_i - y_j)$ and $f_i^{-1} = \prod_{j=1, j\neq i}^{n}(y_i - y_j)$.

We then have $\hat{C}(x,y) = \left(\frac{h_i f_j}{x_i - y_j} \right)_{i,j=1}^{n} = HC(x,y)F$ where $H = diag(h_1, h_2, \ldots, h_n)$ and $F = diag(f_1, f_2, \ldots, f_n)$.

Letting $X' = F^{-1}X$ and $b' = Hb$, the system $C(x,y)X = b$ is therefore equivalent to the system $\hat{C}(x,y)X' = b'$. Therefore we solve the latter system.
By using the Theorem of V. Olshevsky [5], we have $V(x) = \hat{C}(x,y).V(y)$ where $V(x)$ and $V(y)$ are polynomial Vandermonde matrices. Consequently the system $\hat{C}(x,y)X'=b'$ can be solved in the following steps:

1. Solve $V(x)Z = b'$ for Z
2. Compute the matrix-vector product $V(y)Z = X'$
3. Compute the matrix-vector product $FX' = X$

Algorithm 4.2 (Solving TP Cauchy system) If C is a TP Cauchy matrix determined by the vectors $x = (x_1, x_2,\ldots, x_n)$ and $y = (y_1, y_2,\ldots, y_n)$ the following algorithm computes the solution to $CX = b$.

```
10 function newcauchy([$x_1, x_2, \ldots, x_n$], [$y_1, y_2, \ldots, y_n$],  [$b_1, b_2, \ldots, b_n$])
20 for i = 1 : n
30     p=1
40     for j = 1 : n
50         $p=p\cdot(x_i - y_j)$
60   $b_i = b_i \cdot p$
70 z = vand([$x_1, x_2, \ldots, x_n$], [$b_1, b_2, \ldots, b_n$])
80 v=vander([$x_1, x_2, \ldots, x_n$])
90 for i=1 : n
100    s=0
110    for j=1 : n
120        $s = s + v_{ij}\cdot z_j$
130    $X_i'$=s;
140 for i = 1 : n
150     p=1
160     for j = 1 : n
170         if $j \neq i$
180          $p=p\cdot(y_i - y_j)$
190   $X_i = X_i'/p$
200 return X
```

```
10 function v= vander([$x_1, x_2, \ldots, x_n$])
20  f=[];
30  for i=1:n
40      for j=1:n
50          $f_{ij}=(x_i)$^(j-1)
60  v=f
```

The accuracy of the Algorithm 4.2 is addressed in the Corollary 4.1 and the following two propositions from Nicholas (more details can be found in [7], 62-73).

Proposition 4.1

Consider the inner product x^ty, where $x, y \in R^n$, we will assume that the evaluation is from left to right. The following error bound is valid.

$$| x^ty - fl(x^ty) | \leq n\varepsilon | x^t || y | + O(\varepsilon^2) \qquad (4.5)$$

Proposition 4.2

Let $A \in R^{nxn}$, $x \in R^n$, and $y = Ax$. The vector y can be formed as n inner products, $y_i = a_i^tx, for\ i = 1,2,\ldots, n$, where a_i^t is the i^{th} row of A. From (4.5) we have

$\hat{y}_i = (a_i + \Delta a_i)^tx, \quad | \Delta a_i | \leq \gamma_n | a_i |$ where $\gamma_n = n\varepsilon/(1 - n\varepsilon)$.

This gives the backward error result

$$\hat{y} = (A + \Delta A)x, \quad | \Delta A | \leq \gamma_n | A |, \qquad (4.6)$$

which implies the forward error bound

$| y - \hat{y} | \leq \gamma_n | A || x |$

4-4 Numerical Experiments

Our challenge is to test the accuracy of a routine that we claim is more accurate than some other methods. We performed some numerical experiments and confirmed the correctness of our algorithms. We solved the TP Cauchy linear system using the MATLAB routine for solving linear systems of equations with a very high precision arithmetic. We compared the MATLAB results to the results when using our new algorithm (implemented in MATLAB and C++ on an INTEL processor, running IEEE double precision arithmetic with $\varepsilon = 2^{-53} \approx 10^{-16}$). We chose random nodes x_i and y_i, which are sorted in order to preserve the totally positive property of the matrices. Also we selected a right-hand side $b = (-1, 1, -1, 1, -1, 1, -1, 1, -1, 1)^t$ with alternating signs in order to avoid a subtraction cancellation.

We computed the solution to $Cz = b$ using Algorithm 4.1 and Algorithm 2.1, and also using the MATLAB routine of solving linear systems of equations with *15* decimals. We generated the random numbers in MATLAB, output them to a file, read them into MATLAB and C++, and then formed the explicit matrices (Cauchy and Vandermonde) . Then we solved the related systems to those matrices.

Clearly, we require a measure of the condition of the system of equations. We know that a system of equations without a solution – the very worst condition possible – has a coefficient matrix with determinant of zero. It is therefore tempting to think that the size of the determinant of A can be used as a measure of condition. However, if $Ax = b$ and A is an n x n diagonal matrix with each element on the leading diagonal equal to $s\ (s \neq 0)$ then A is perfectly conditioned, irrespective of the value of s. But the determinant of A in this case is s^n. Thus, the size of the determinant of A is not a suitable measure of condition because in this example, it changes with s even though the condition of the system is constant. Two of the functions MATLAB provides to estimate condition of a matrix are *cond( )* and *rcond( )*. Note that *rcond(A)* returns an estimate for the reciprocal of the condition of A in *1-norm* using the LAPACK condition estimator. If A is well

conditioned, *rcond(A)* is near *1.0*. If *A* is badly conditioned, *rcond(A)* is near *0.0*. For a perfect condition, *cond( )* is unity but gives a large value for a matrix which is ill-conditioned. Compared to *cond, rcond* is a more efficient, but less reliable, method of estimating the condition of a matrix. We observed that when *cond* is larger than *8.2e15* MATLAB estimates the system close to singular or badly scaled and may end up with an inaccurate solution. In order to have a "well conditioned" Cauchy matrix for the experiment, we kept track the condition of the explicit Cauchy matrix obtained from the random numbers x_i and y_i. For our experiment we only used a Cauchy matrix that was "well conditioned".
The results in table 4.1 and table 4.2 confirmed our claims that the problem is well-conditioned, even though $cond(C) = 4.6\cdot 10^{11}$ and that the MATLAB routine and Cauchy method computed a solution that was only good in the first digits.

Node x_i	Node y_i	MATLAB routine	Cauchy	NewCauchy
98.4509759201955	6.56495308392387	282043865261.052	282043695587.749	282043695587.749
100.053150319458	19.9982456896986	-916600568139.946	-916600016328.87	-916600016328.87
103.223225800457	30.6277256267298	847425607126.141	847425096631.954	847425096631.959
103.532590502816	51.5368171010196	-357927962745.76	-357927746828.442	-357927746828.576
111.784120506889	63.8562731455501	203875420964.974	203875297895.372	203875297896.338
119.469975362746	74.2147972831518	-94238488880.275	-94238431996.1953	-94238432008.0202
128.399551803366	78.3526125396556	36826167160.0719	36826144941.9119	36826144944.4418
129.771803821511	87.0529484972559	-1149427777.48736	-1149427086.53091	-1149427089.74589
133.12448404099	91.2927736207461	90485511.0790638	90485456.971817	90485468.2329705
142.227334657284	95.7299997399511	-613512.516887339	-613512.155255823	-613512.904359188

Table 4.1: Solving a 10 x 10 TP Cauchy linear system in MATLAB

Cauchy	NewCauchy
282043695587.7490200000	282043695587.7490200000
-916600016328.8701200000	-916600016328.8707300000
847425096631.9538600000	847425096631.9587400000
-357927746828.4415300000	-357927746828.5543200000
203875297895.3720100000	203875297896.3378300000
-94238431996.1952970000	-94238432005.5710140000
36826144941.9119420000	36826144951.3231740000
-1149427086.5309062000	-1149427098.3944476000
90485456.9718170020	90485459.1769653410
-613512.1552558227	-613513.6984067048

Table 4.2: Solving a 10 x 10 TP Cauchy linear system in C++

Conclusions

We have presented the Björck-Pereyra-type methods for TP matrices whose initial minors are accurately and efficiently computable. We presented Koev's algorithm for accurately and efficiently solving the TP generalized Vandermonde linear system along with his new method for computing the Schur function. This in turn allowed us to derive a Björck-Pereyra-type method for the standard Vandermonde linear system. We used the theorem of V. Olshevsky along with the Björck-Pereyra-type method for solving the standard Vandermonde linear system to present a new method for solving accurately and efficiently solving the TP Cauchy linear systems that is faster than the Cauchy method.

References

[1]. T. Boros, T. Kailath, and V. Olshevsky. *The fast parallel Björck – Pereyra - type algorithm for parallel solution of Cauchy linear equations*. Linear Algebra applications,302-303 (1999), 265-293.

[2]. T. Boros, T. Kailath, and V. Olshevsky. *A fast parallel Björck – Pereyra - type Algorithm for solving Cauchy linear equations*, AMS subject classification: 65F05, 65L20, 15A09, 15A23.

[3]. James Demmel, *Accurate SVDs of Structured Matrices* (Oct. 9, 1997), LAPACK working Note 130, University of Tennessee Computer Science Report ut-cs-97-375.

[4]. M. Gasca and J. M. Pena. *On factorizations of totally positive matrices. In Total Positivity and its Applications*. Kluwer Acad. Publ., Dordrecht, 1996.

[5]. I. Gohberg and V. Olshevsky, *Fast algorithms with preprocessing for matrix-vector multiplication problems*, Journal of Complexity, (1994)

[6]. Robert Todd Gregory, *Error Free Computation: Why It Is Needed and Methods For Doing It*, Robert E. Krieger publishing company Huntington, NY 1980 12-16, 24-30

[7]. Nicholas J. Higham, *Accuracy and Stability of numerical algorithms*, Second Edition, (2002) SIAM, ISBN 0-89871-521-0, 1-91, 157-192, 415-430, 489-571.

[8]. N. J. Highma. *Stability analysis of algorithms for solving confluent Vandermonde-like systems*, SIAM J. Mat. Anal. Appl., (1990), 11(1), 23-41.

[9]. Plamen Stefanov Koev, *Accurate and Efficient Computations with Structured Matrices*, the dissertation of his PhD Thesis, University of California, Berkeley (2002).

[10]. John Penny, *Numerical Methods Using MATLAB*, George Lindfield, Ellis Horwood Limited (1995), 1-82

[11]. Christoph W. Ueberhuber, *Numerical Computation 1 : Methods, Software, and Analysis*, 1997, Springer - Verlag Berlin Heid. NY, ISBN 3-540-62058-3, 117- 215.