Vandermonde systems on equidistant nodes in $[0,1]$: accurate computation

A. Eisinberg, G. Fedele, C. Imbrogno

Dip. Elettronica Informatica e Sistemistica,
Università degli Studi della Calabria,
87036, Rende (Cs), Italy

Abstract

This paper deals with Vandermonde matrices $V$ whose nodes are the equidistant points in $[0,1]$. We give an analytic factorization and explicit formula for the entries of their inverse, and explore its computational issues. We also give asymptotic estimates of the Frobenius norm of both $V$ and its inverse and show that a new representation of the floating point number system allows one to build an accurate algorithm for the interpolation problem on equidistant nodes in $[0,1]$.

Key words: Vandermonde matrices, Polynomial interpolation, Conditioning, Floating point numbers

1 Introduction

Vandermonde matrices, defined by $\tilde{V}_{ij} = x_i^{j-1}$, $x_i \in \mathbb{C}$, are still a topical subject in matrix theory and numerical analysis. The interest arises as they occur in applications, for example in polynomial and exponential interpolation, and because they are ill-conditioned, at least for positive or symmetric real nodes: Gautschi et al [6]. The primal system $\tilde{V}x = b$ represents a moment problem, which arises when determining the weights for a quadrature rule, while the matrix $V = \tilde{V}^T$ involved in the dual system $Vc = f$ plays an important role in polynomial interpolation: Higham [9]. The special structure of $V$ allows us to use ad hoc algorithms that require $O(n^2)$ elementary operations for solving a Vandermonde system. The most celebrated of them is the one by Björck and Pereyra: Björck et al [1]; the high accuracy of the solutions it gives has been justified theoretically: Higham [8]. Many explicit formulas and computational schemes for the entries of $V^{-1}$ have also been given: Knuth [10] and Eisinberg et al [5]. Bounds or estimates of the norm of both $V$ and $V^{-1}$ are also interesting, for example to investigate the condition of the polynomial
interpolation problem. Answer to these problems have been given first for special configurations of the nodes and recently for general ones: Tyrtyshnikov [12] and Golub et al. [7]. In this paper we consider a Vandermonde matrix \( V \) on the set of \( n \) equidistant nodes in the interval \([0, 1]\). Although Runge [11] proved that the set of equally spaced points represents a “bad” choice for Lagrange interpolation, a lot of research has been made for the design of accurate algorithms for polynomial interpolation on such set of nodes. Such an interest is due to the fact that the choice of equidistant nodes frequently occurs in many applications [2]. By using arguments from integer number theory, we find an explicit factorization for the entries of the inverse of \( V \) and show some recursions for the practical implementation of this formula. We also give formulas for the Frobenius norm of both \( V \) and \( V^{-1} \), and simple asymptotic estimates of these norms. Finally we report numerical simulations and show an accurate interpolation algorithm by using a new representation of the floating point number system.

2 Main results

Let

\[
X_n = \left\{ \frac{i - 1}{n - 1}, \ i = 1, 2, \ldots, n \right\}
\]

be the set of equidistant nodes in the interval \([0, 1]\). The matrix \( V \) on \( X_n \) is:

\[
V_{i,j} = \left( \frac{i - 1}{n - 1} \right)^{j-1}, \ i, j = 1, 2, \ldots, n.
\]

We first give an explicit formula for the inverse of \( V \), namely \( W \), using Stirling numbers as the main tool. Let \( m, n \in \mathbb{N}, x \in \mathbb{R} \); we will denote Stirling number of the first kind by \([m \ n] \) defined by

\[
\sum_{k} (-1)^{n+k} \binom{n}{k} x^k = n! \binom{x}{n}
\]

and it is zero if \( m < n \). Relevant properties used throughout the paper are:

\[
\sum_{k=1}^{n} (-1)^k (m - 1)^{k-1} \binom{n - 1}{k - 1} = (-1)^n (n - 1)! \binom{m - 1}{n - 1}
\]

\[
\sum_{k} \binom{m - 1}{k - 1} \binom{n - 1}{k - 1} = \binom{m + n - 2}{m - 1}
\]

\[
\sum_{k} (-1)^k \binom{m}{k} \binom{k}{n} = (-1)^m \delta_{mn}
\]
where $\delta_{mn}$ is the well-known Kronecher $\delta$-function. Moreover, Stirling numbers of the first kind and binomials satisfy the following recursion:

$$
\begin{bmatrix} i \\ j \end{bmatrix} = (i - 1) \begin{bmatrix} i - 1 \\ j \end{bmatrix} + \begin{bmatrix} i - 1 \\ j - 1 \end{bmatrix}, \quad \begin{bmatrix} i \\ 1 \end{bmatrix} = (i - 1)!
$$

(7)

$$
\begin{pmatrix} x \\ n \end{pmatrix} = \begin{pmatrix} x - 1 \\ n \end{pmatrix} + \begin{pmatrix} x - 1 \\ n - 1 \end{pmatrix}, \quad \begin{pmatrix} x \\ 0 \end{pmatrix} = 1
$$

(8)

**Theorem 1** The entries of the inverse of the Vandermonde matrix $W$ on the set $X_n$ is:

$$W_{i,j} = (-1)^{i+j} (n-1)^{i-1} \sum_{k=1}^{n} \frac{1}{(k-1)!} \begin{bmatrix} k - 1 \\ i - 1 \end{bmatrix}, \quad i, j = 1, 2, \ldots, n.$$  

(9)

**Proof.** The proposition is proved if the following relationship holds:

$$\sum_{q=1}^{n} \sum_{k=1}^{n} (-1)^{q+j} (i-1)^{q-1} \frac{1}{(k-1)!} \begin{bmatrix} k - 1 \\ j - 1 \end{bmatrix} = \delta_{ij}, \quad i, j = 1, 2, \ldots, n.$$  

(10)

By property (4) we have:

$$\sum_{k=1}^{n} (-1)^{j+k} \begin{bmatrix} k - 1 \\ j - 1 \end{bmatrix} = \delta_{ij}, \quad i, j = 1, 2, \ldots, n$$

and by using (6), (10) follows.

The proposed formula can also be proved from Eisenberg et al [4] through an affine transformation on the set of nodes, however the above direct proof is much simpler. The following factorization comes from (9) and it can be verified by inspection.

**Corollary 1** An analytic factorization of $W$ is:

$$W = \frac{1}{(n-1)!} D_1 U D_2 L$$

(11)

where:

$$D_1 = \text{diag}\left\{ (n-1)^{i-1} \right\}_{i=1,2,\ldots,n},$$

$$U_{i,j} = (-1)^{i+1} \begin{bmatrix} j - 1 \\ i - 1 \end{bmatrix}, \quad i = 1, 2, \ldots, n, \quad j = i, i+1, \ldots, n,$$

$$D_2 = \text{diag}\left\{ (n-1)! \right\}_{(i-1)!}_{i=1,2,\ldots,n},$$

$$L_{i,j} = (-1)^{j+1} \begin{bmatrix} i - 1 \\ j - 1 \end{bmatrix}, \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, i.$$
Therefore the polynomial which interpolates the function \( f(x) \) on the set of nodes (1) is:

\[
p(x) = \sum_{k=1}^{n} c_k x^{k-1}
\]

(12)

where the coefficient vector \( c \) can be calculated as:

\[
c = \frac{1}{(n-1)!} D_1 U D_2 L \cdot f
\]

(13)

and \( f_k = f \left( \frac{k-1}{n-1} \right), \quad k = 1, 2, \ldots, n. \)

3 Computational aspects

The next result gives recursive formulas for the entries of the matrices \( U \) and \( L \).

**Proposition 1** The matrix \( U \) satisfies the recursion

\[
U_{1,1} = 1,
\]

\[
U_{1,j} = 0, \quad j = 2, 3, \ldots, n,
\]

\[
U_{i,j} = (j - 2) U_{i,j-1} - U_{i-1,j-1}, \quad i = 2, 3, \ldots, n; \quad j = i, i + 1, \ldots, n
\]

(14)

while the entries of \( L \) satisfy

\[
L_{i,1} = 1, \quad i = 1, 2, \ldots, n,
\]

\[
L_{i,i} = (-1)^{i+1}, \quad i = 1, 2, \ldots, n,
\]

\[
L_{i,j} = L_{i-1,j} - L_{i-1,j-1}, \quad i = 2, 3, \ldots, n; \quad j = 2, 3, \ldots, i
\]

(15)

**Proof.** The above recursions follow directly from (7) and (8). □

The next proposition enables to calculate the product \( L \cdot f \) in (13) without building the matrix \( L \).

**Proposition 2** Let \( w \) be the product \( L \cdot f \) and let \( H \) be the matrix defined as:

\[
H_{i,j} = \sum_{k=j}^{i+j-1} (-1)^{k+j} \binom{i-1}{k-j} f_k \quad i, j = 1, 2, \ldots, n
\]

then

\[
w_i = H_{i,1}, \quad i = 1, 2, \ldots, n.
\]

(16)
Proof. The matrix $H$ satisfies the following recursive property:

$$H_{i,j} = H_{i-1,j} - H_{i-1,j+1}, \quad i = 2, 3, ..., n; \quad j = 1, 2, n + 1 - i.$$ 

By noting that

$$w_i = \sum_{k=1}^{i} (-1)^{k+1} \binom{i-1}{k-1} f_k \quad i = 1, 2, ..., n,$$

(16) follows. ■

For the previous proposition building the $L$ matrix for the dual problem is not necessary. Such matrix instead turns out necessary for solving the primal problem.

4 Estimating Frobenius norm of $V$ and $W$

As concerns the Frobenius norm of $V$, we can write

$$||V||^2_F = 1 + \sum_{j=1}^{n} \frac{1}{(n-1)^{2j-2}} S_{2j-2}(n)$$

where

$$S_m(n) = \sum_{k=1}^{n-1} k^m.$$  

(18)

Taking into account only the first three terms in (18), that is:

$$S_m(n) \sim \frac{1}{n+1} m^{n+1} + \frac{1}{2} m^n + \frac{1}{12} nm^{n-1}$$

the Frobenius norm of $V$ becomes:

$$||V||_F \sim \sqrt{1 + \frac{7}{12} n + \frac{1}{2} (n-1) \left[ \gamma + \ln(4) + \psi \left( n + \frac{1}{2} \right) \right]}$$

(19)

where $\gamma = 0.577215665$ is the Euler-Mascheroni constant and the function $\psi(x)$ is defined as:

$$\psi(x) = \frac{d}{dx} \ln \Gamma(x)$$

where $\Gamma(x)$ is the gamma function. The term $\psi \left( n + \frac{1}{2} \right)$ in (19) can be written as:

$$\psi \left( n + \frac{1}{2} \right) = -\gamma + 2 \sum_{k=1}^{n} \frac{1}{2k-1} - \ln(4)$$
where
\[ \sum_{k=1}^{n} \frac{1}{2k-1} = \frac{1}{2} (\gamma + \ln(n)) + \ln(2) + \frac{B_2}{8n^2} + ... \]
and \( B_k \) is the \( k^{th} \) Bernoulli number, then (19) has the following asymptotic expression:
\[ ||V||_F \sim \sqrt{\frac{1}{2} n \ln(n)}. \]
By Theorem 1, using the binomial identity (5) we have:
\[ ||W||_F^2 = \sum_{q=1}^{n} \sum_{k=q}^{n} \sum_{s=q}^{n} (n-1)^{2q-2} \frac{1}{(k-1)!(s-1)!} \left( \frac{k+s-2}{k-1} \right) \left[ \frac{k-1}{q-1} \right] \left[ \frac{s-1}{q-1} \right]. \tag{20} \]
Taking into account only the terms in (20) where \( k = s = n \) we conjecture that:
\[ ||W||_F \sim \frac{4}{3} \frac{1}{(n-1)!} \sqrt{\left( \frac{2n-2}{n-1} \right) \sum_{q=1}^{n} (n-1)^{2q-2} \left[ \frac{n-1}{q-1} \right]^2}. \tag{21} \]
Figures 1 and 2 show the accuracy of the estimates of the Frobenius norm of both \( V \) and \( W \) in term of relative error for \( n \) in the interval \([1, 100]\).

5 Numerical experiments

This section shows some numerical experiments, aimed at investigating the accuracy of the proposed factorization. We have solved several dual systems \( Vc = f \) and primal systems \( Va = b \) and have compared our results with those obtained by the Björck-Pereyra algorithms. We have used package Matlab [13] to compute the approximate solutions \( \hat{c} \) and \( \hat{a} \) and the package Mathematica.
Fig. 2. Relative errors estimating $\|W_n\|_F$ by (21).

[14] (using extended precision) for the exact ones and for errors

$$
\epsilon_c = \max_{1 \leq i \leq n} \frac{|\hat{c}_i - c_i|}{|c_i|}, \quad (22)
$$

$$
\epsilon_a = \max_{1 \leq i \leq n} \frac{|\hat{a}_i - a_i|}{|a_i|}. \quad (23)
$$

A set of experiments has been run, for $n = 2 \div 10, 20, 30$. We have generated the right-hand sides $f$ and $b$ with random entries uniformly distributed in the interval $[-1, 1]$. Tables 1 and 2 report the maximum value of (22) and (23), respectively, over 10000 runs and also the fraction of trials in which the proposed algorithm (EFI) gives equal or more accurate result than Björck-Pereyra one (BP). EFI algorithm seems to perform better than the Björck-Pereyra one in terms of numerical accuracy. Figure 3 shows the empirical probability distribution of (22) obtained by 100000 runs of both the algorithms, with entries of $f$ uniformly distributed in $[-1, 1]$. $\epsilon_c$ is measured in term of the unit roundoff $u = 2^{-53}$. As to the computational cost, both algorithms require $2.5n^2$ flops for solving the dual problem. When the primal problem is considered, EFI algorithm costs $3.5n^2$ flops while BP algorithm costs $2.5n^2$. However the proposed factorization turns out useful when several interpolation problems must be solved on the same set of nodes: in fact if $m$ is the number of interpolation problems to be solved on a set of $n$ equidistant nodes in $[0, 1]$, the Björck-Pereyra’s algorithm costs $2.5n^2m$ flops while EFI costs $n^2 + 1.5n^2m$ flops. For Matlab code refer to Appendix A.
Fig. 3. Empirical probability density of $\epsilon_c$ for EFI and BP algorithms ($n = 10$, 100000 runs).

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP</td>
<td>1.11-16</td>
<td>2.00-14</td>
<td>2.06-15</td>
<td>6.84-13</td>
<td>2.15-12</td>
<td>1.39-11</td>
</tr>
<tr>
<td>EFI</td>
<td>1.11-16</td>
<td>2.00-14</td>
<td>4.61-16</td>
<td>6.52-13</td>
<td>5.88-13</td>
<td>2.93-12</td>
</tr>
<tr>
<td>s.r.</td>
<td>1</td>
<td>1</td>
<td>0.69</td>
<td>0.54</td>
<td>0.74</td>
<td>0.81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>20</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP</td>
<td>5.49-11</td>
<td>3.96-13</td>
<td>6.98-10</td>
<td>1.04-10</td>
<td>1.40-09</td>
</tr>
<tr>
<td>EFI</td>
<td>5.96-12</td>
<td>5.14-13</td>
<td>5.30-11</td>
<td>6.01-11</td>
<td>3.17-11</td>
</tr>
<tr>
<td>s.r.</td>
<td>0.81</td>
<td>0.64</td>
<td>0.90</td>
<td>0.98</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Table 1. Dual problem - Maximum value of $\epsilon_c$. Success rate of EFI algorithm over 10000 runs. $f \in [-1, 1]$.

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP</td>
<td>1.11-16</td>
<td>4.97-14</td>
<td>1.39-12</td>
<td>5.23-12</td>
<td>1.49-12</td>
<td>4.18-12</td>
</tr>
<tr>
<td>EFI</td>
<td>1.11-16</td>
<td>4.97-14</td>
<td>1.07-12</td>
<td>6.58-12</td>
<td>2.57-12</td>
<td>6.06-12</td>
</tr>
<tr>
<td>s.r.</td>
<td>1</td>
<td>0.95</td>
<td>0.78</td>
<td>0.61</td>
<td>0.85</td>
<td>0.90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>20</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>BP</td>
<td>2.00-12</td>
<td>1.67-12</td>
<td>1.93-12</td>
<td>6.72-10</td>
<td>9.68-12</td>
</tr>
<tr>
<td>EFI</td>
<td>3.80-12</td>
<td>9.76-13</td>
<td>3.78-12</td>
<td>6.94-10</td>
<td>1.07-11</td>
</tr>
<tr>
<td>s.r.</td>
<td>0.91</td>
<td>0.72</td>
<td>0.93</td>
<td>0.97</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Table 2. Primal problem - Maximum value of $\epsilon_a$. Success rate of EFI algorithm over 10000 runs. $f \in [-1, 1]$. 8
6 A new representation of the floating point number system

In this section we give a new representation of the floating point number system. A floating number is represented as a linear combination of integers and the coefficients are powers of the base \([3]\). This formulation will be used in the next section to build an accurate algorithm for the polynomial interpolation problem on the set of equidistant nodes in \([0, 1]\). The floating point number system \(F\) is a subset of the real numbers defined as follows \([9]\):

\[
F = \left\{ y \in \mathbb{R} \mid y = \pm \beta^e \sum_{k=1}^{t} d_k \beta^{-k} \right\}. \tag{24}
\]

The system \(F\) is characterized by four integer parameters:

- the base \(\beta \geq 2\),
- the precision \(t \geq 1\),
- the exponent range \(e_{\min} \leq e \leq e_{\max}\).

Each digit \(d_k\) satisfies \(0 \leq d_k \leq \beta - 1\), and \(d_1 \neq 0\) for normalized numbers.

The following representation of \(F\) is proposed \([3]\):

\[
y = sgn(y) \sum_{j=1}^{q} y_j \beta^{e - A(j)} \tag{25}
\]

where:

\[
y_j = \sum_{k=A(j)-1+1}^{A(j)} d_k \beta^{A(j)-k}, \quad j = 1, 2, ..., q, \tag{26}
\]

\[
\begin{cases}
  A(j) = A(j-1) + \alpha_j, & j = 1, 2, ..., q \\
  A(0) = 0
\end{cases} \tag{27}
\]

\[
\begin{cases}
  \alpha_j \text{ integer} & \alpha_j \geq 1, \quad j = 1, 2, ..., q \\
  \sum_{j=1}^{q} \alpha_j = t
\end{cases} \tag{28}
\]

\[
sgn(y) = \begin{cases}
  +1, & y > 0 \\
  0, & y = 0 \\
  -1, & y < 0
\end{cases} \tag{29}
\]

Formula (25) can be easily proved by inspection. It should be noted that \(y_j\) is integer, as it can be see from (26). The integer quantities \(y_j\) can be obtained by using the following proposition:
Proposition 3 Let
\[ y = \pm \beta^e \sum_{k=1}^{t} d_k \beta^{-k} \] (30)
be a floating point number, then \( y_j \) can be expressed as:
\[ y_j = \left[ \beta^{A(j)} y_n - \sum_{k=1}^{t} \beta^{A(j)-A(i)} y_i \right] \] (31)
where:
\[ y_n = \frac{y}{\pm \beta^e} = \sum_{k=1}^{t} d_k \beta^{-k} \] (32)
and \( \lfloor x \rfloor \) is the greatest integer less than or equal to \( x \).

Proof. It is sufficient to show that (31) and (26) are equivalent. By substituting (32) and (26) in (31) we have:
\[ y_j = \left[ \sum_{k=1}^{t} d_k \beta^{A(j)-k} - \sum_{i=1}^{j-1} \beta^{A(j)-A(i)} \sum_{k=A(i)+1}^{A(j)} d_k \beta^{A(i)-k} \right] \] (33)
By simple algebraic manipulations [10] we obtain:
\[ y_j = \left[ \sum_{k=1}^{t} d_k \beta^{A(j)-k} - \sum_{i=1}^{j-1} \beta^{A(j)-A(i)} \right] = \]
\[ \left[ \sum_{k=A(j-1)+1}^{t} d_k \beta^{A(j)-k} \right] = \sum_{k=A(j-1)+1}^{A(j)} d_k \beta^{A(j)-k}. \] (34)

The following pseudo-code maps the generic floating-point number into the new representation (25), by using the IEEE double standard \( (t = 53, \beta = 2) \):

function fp2nfp
Input:
\( x = \) floating-point number
\( q \geq 2 = \) number of splitting
Output:
\( y = \) integer vector containing the \( y_j \) in (26)
\( exp = \) integer vector containing the quantities \( exp(i) = e - A(i) \)
\( \alpha(i) = \left\lfloor \frac{t}{q} \right\rfloor, \quad i = 1, 2, ..., q - 1 \)
\( \alpha(q) = t - \sum_{k=1}^{q-1} \alpha(k) \)
\( e = \lfloor \log_2(|x|) \rfloor + 1 \)
\( f = \frac{\alpha}{e} \)
\( aux = f \)
for \( i = 1 : q \)
\[
\begin{align*}
\text{aux} &= \text{aux} \cdot 2^{\alpha(i)} \\
y(i) &= \lfloor \text{aux} \rfloor \\
an\text{ux} &= \text{aux} - y(i)
\end{align*}
\]
end
\[
\exp(i) = e - \sum_{k=1}^{i} \alpha(k).
\]

7 Accurate interpolation

As we noted the inverse of the Vandermonde matrix on equidistant nodes in \([0, 1]\) can be factorized as a product of four matrix with integer entries that can be stored without rounding errors. By using such property and the fact that each entry of the vector \( f \) can be represented as a linear combination of integers where the coefficients are powers of the base, we build an accurate algorithm for the interpolation problem on equidistant nodes in \([0, 1]\). We consider the quantity
\[
c = \frac{1}{(n - 1)!} D_1 U D_2 L f
\]
and, by using (25) we write the vector as
\[
f = [f_1, f_2, ..., f_n]^T = Y \cdot e
\]
where
\[
Y = \begin{bmatrix}
y_1(1) & y_2(1) & y_3(1) \\
y_1(2) & y_2(2) & y_3(2) \\
y_1(3) & y_2(3) & y_3(3) \\
\vdots & \vdots & \vdots \\
y_1(n) & y_2(n) & y_3(n)
\end{bmatrix}
\]
and
\[
e = [2^{\exp(1)}, 2^{\exp(2)}, 2^{\exp(3)}]^T.
\]
Note that \( y_q(k), q = 1, 2, 3; k = 1, 2, ..., n \) are integer quantities multiplied by power of the base of the floating point number system. Then
\[
c = \frac{1}{(n - 1)!} D_1 U D_2 L Y e.
\]
(35)

The numerical experiments are aimed in showing the high accuracy of formula (35). 100000 experiments have been run for \( n = 2, 3, ..., 10 \). Table 3 reports the maximum and mean value of \( \epsilon_c \) in term of unit roundoff \( u = 2^{-53} \).
<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>1.00</td>
<td>1.00</td>
<td>2.16</td>
<td>2.00</td>
<td>2.88</td>
<td>2.91</td>
<td>2.76</td>
<td>2.60</td>
<td>4.02</td>
</tr>
<tr>
<td>mean</td>
<td>0.07</td>
<td>0.32</td>
<td>1.05</td>
<td>1.14</td>
<td>1.04</td>
<td>1.25</td>
<td>1.25</td>
<td>1.16</td>
<td>1.84</td>
</tr>
</tbody>
</table>

Table 3. Dual problem - Maximum and mean value of $\epsilon_c$ in term of unit roundoff for (35) over 100000 runs. $f \in [-1, 1]$.

8 Conclusion

In this paper we derived an explicit factorization of the inverse of the Vandermonde matrix $W$ on equidistant nodes in $[0, 1]$. Such factorization allows to design an efficient algorithm to solve Vandermonde systems. The numerical experiments indicate that our approach is more stable compared with existing Björck-Pereyra algorithm. We have showed that a new representation of the floating point number system and the factorization of $W$ in terms of matrices with integer entries, allow us to build an accurate algorithm for the interpolation problem on equidistant nodes in $[0, 1]$ with errors comparable to the unity roundoff of the floating point system.

Appendix A - Matlab code

function c=EFI(f)
[n,t]=size(f);
if t==1
    f=f';
end
% Matrix D1
%--------------------------------------------------------
D1=zeros(n,1);
D1(1)=1;
for i=2:n
    D1(i)=(n-1)*D1(i-1);
end
% Matrix U
%--------------------------------------------------------
U=zeros(n);
for i=1:n
    U(i,i)=(-1)^(i+1);
end
for j=3:n
    U(2,j)=(j-2)*U(2,j-1);
end
for i=3:n-1
    for j=i+1:n
        U(i,j)=(j-2)*U(i,j-1)-U(i-1,j-1);
    end
end
% Matrix D2
D2=zeros(n,1);
D2(n)=1;
for i=n-1:-1:1
    D2(i)=i*D2(i+1);
end
% Product L.f
w=zeros(n,1);
b=f';
for i=1:n
    w(i)=(-1)^(i+1)*b(1);
    b=diff(b);
end
%--------------------------------------------------------
c=(D1.*(U*(D2.*w)))/prod(1:n-1);

References


