On the Numerical Computation of the Determinant of a Bivariate Polynomial Matrix

Dimitris Varsamis

Department of Informatics Engineering
Technological Educational Institute of Central Macedonia - Serres
62124, Serres, Greece

Nicholas Karampetakis

Department of Mathematics, Aristotle University of Thessaloniki
54124 Thessaloniki, Greece

Copyright © 2015 Dimitris Varsamis and Nicholas Karampetakis. This article is distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

The paper, based on the technique evaluation-interpolation, proposes a new numerical algorithm for the computation of the determinant of a bivariate polynomial matrix. The numerical algorithm uses the Newton bivariate polynomial interpolation with equidistant points. The algorithm is applied to the programming environment of MATLAB and is compared to the corresponding build-in function of MATLAB \( \text{det()} \) which uses symbolic operations.

Keywords: Determinant, Bivariate polynomial interpolation, Evaluation-Interpolation, Numerical Algorithm

1 Introduction

Polynomial interpolation in several variables is a relatively new topic that has applications to many mathematical problems where the solution is a multivariate polynomial or polynomial matrix i.e. the computation of a) the inverse of a polynomial matrix that has applications in analysis and synthesis theory of
Control Systems [5, 8], b) the greatest common divisor [7], c) the determinant of a polynomial matrix [9], d) the solution of Diophantine equations [4], e) the transfer function of a multidimensional system [1], f) the generalized inverse of a polynomial matrix [6] e.t.c. The analytical solution of these problems leads to difficult and complex procedures that are very difficult to implement in a programming environment which must support symbolic operations. The evaluation–interpolation technique that is used in all the interpolation methods, avoids such kind of problems by using known numerical methods. More specifically, this technique evaluates the values of the polynomial solution that we are looking for at given interpolation points and then constructs the polynomial solution by using interpolation techniques.

The determinant of a bivariate polynomial matrix is a bivariate polynomial that you know their total degree. The total degree gives us the number of interpolation points for the evaluation. For the interpolation part of the technique evaluation–interpolation we use the Newton bivariate interpolation on a new set of equidistant points in triangular basis and a new numerical transformation for the representation of the determinant in monomial basis.

In this paper, we start by presenting in Section 2, the way that the Newton bivariate interpolation method in [10] is applied to equidistant points with aim to simplify the implementation. The numerical algorithmic implementation of the Newton bivariate interpolation method is given, by three alternative methods. A comparison between the proposed implementations is given in relation to the number of iterations that is used in each implementation. Finally, in Section 3 we present the proposed numerical algorithm which it computes the determinant of a bivariate polynomial matrix. Additionally, we perform a programming implementation of the proposed numerical algorithm by given focus in execution time and numerical precision instead to build-in function of MATLAB \texttt{det()}.

2 Two-variable Newton interpolation

Let the equidistant points $x_i = x_0 + i \cdot h_x$ and $y_j = y_0 + j \cdot h_y$ in the set

$$S_{\Delta}^{(n)} = \{(x_i, y_j) \mid i, j \in \mathbb{N}, i + j \leq n\}$$

of interpolation points, where the number of interpolation points is given by

$$N = \binom{n + 2}{n} = \frac{(n + 2)!}{n! \cdot 2!} = \frac{(n + 1)(n + 2)}{2}$$

and let the Newton bivariate polynomial passing through from these points is given by

$$p(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n-i} a_{i,j} x^i y^j$$
where $n$ is the total degree of the bivariate polynomial. An algorithm for the computation of the above polynomial is given (a) in [11] which is applied to random interpolation points in a triangular basis and (b) in [10] which is applied in the special case of equidistant interpolation points. The recursive type for the equidistant interpolation points is given by

$$d_{i,j}^{(k)} := \begin{cases} 
  d_{i,j}^{(k-1)} - d_{i-1,j}^{(k-1)} & \text{if } (i \geq k \land j < k) \\
  d_{i,j}^{(k-1)} - d_{i,j-1}^{(k-1)} & \text{if } (i < k \land j \geq k) \\
  d_{i,j}^{(k-1)} + d_{i-1,j-1}^{(k-1)} - d_{i,j}^{(k-1)} - d_{i,j-1}^{(k-1)} & \text{if } (i \geq k \land j \geq k) \\
  d_{i,j}^{(k-1)} & \text{if } (i < k \land j < k)
\end{cases}$$  \hspace{1cm} (1)

Based on [10] the interpolation polynomial can be written as follows

$$p(x, y) = X_d^T \cdot D \cdot Y_d$$  \hspace{1cm} (2)

where

$$D = \begin{pmatrix}
  d_{0,0}^{(0)} & d_{0,1}^{(1)} & d_{0,2}^{(2)} & \cdots & d_{0,n-1}^{(n-1)} & d_{0,n}^{(n)} \\
  d_{1,0}^{(1)} & d_{1,1}^{(1)} & d_{1,2}^{(2)} & \cdots & d_{1,n-1}^{(n-1)} & 0 \\
  d_{2,0}^{(2)} & d_{2,1}^{(2)} & d_{2,2}^{(2)} & \cdots & 0 & 0 \\
  \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
  d_{n-1,0}^{(n-1)} & d_{n-1,1}^{(n-1)} & 0 & \cdots & 0 & 0 \\
  d_{n,0}^{(n)} & 0 & 0 & \cdots & 0 & 0
\end{pmatrix} \in \mathbb{R}^{(n+1) \times (n+1)}$$  \hspace{1cm} (3)

and

$$X_d = \begin{pmatrix}
  \frac{x-x_0}{h_x} \\
  \frac{(x-x_0)(x-x_1)}{2h_x^2} \\
  \vdots \\
  \frac{(x-x_0)(x-x_1) \cdots (x-x_{n-1})}{n! h_x^n}
\end{pmatrix} \in \mathbb{R}[x]^{(n+1)}$$

$$Y_d = \begin{pmatrix}
  \frac{y-y_0}{h_y} \\
  \frac{(y-y_0)(y-y_1)}{2h_y^2} \\
  \vdots \\
  \frac{(y-y_0)(y-y_1) \cdots (y-y_{n-1})}{n! h_y^n}
\end{pmatrix} \in \mathbb{R}[y]^{(n+1)}$$  \hspace{1cm} (4)

According the algorithm in [10] we present the following example

**Example 2.1** Suppose that we know from the beginning that the polynomial $p(x, y)$ that we are looking for, has total degree $n = 3$.

**Step 0.** In order to approximate $p(x, y)$ with a $3^{rd}$ order polynomial $p_n(x, y)$, we have to know its values at a set $S_{\Delta}^{(3)}$ of $N = 10$ equidistant points with
\[x_i = x_0 + i \cdot h_x = 0 + i \cdot 1 \Rightarrow x_i = i \text{ and } y_j = y_0 + j \cdot h_y = 0 + j \cdot 1 \Rightarrow y_j = j.\]

**Step 1:** Let the zero order matrix \(D_0\) of initial values is the following

\[
D_0 = \begin{pmatrix}
  d^{(0)}_{0,0} = 2 & d^{(0)}_{0,1} = 1 & d^{(0)}_{0,2} = -6 & d^{(0)}_{0,3} = -25 \\
  d^{(0)}_{1,0} = 3 & d^{(0)}_{1,1} = 1 & d^{(0)}_{1,2} = -13 \\
  d^{(0)}_{2,0} = 10 & d^{(0)}_{2,1} = 11 \\
  d^{(0)}_{3,0} = 29
\end{pmatrix}
\]

where \(D_0(i, j) := p(x_i, y_j) = p(i, j)\).

**Step 2:** For \(k=1\) to \(n=3\), by using the recursive formula (1) we have

For \(k=1\)

\[
D_1 = \begin{pmatrix}
  d^{(0)}_{0,0} = 2 & d^{(1)}_{0,1} = -1 & d^{(1)}_{0,2} = -7 & d^{(1)}_{0,3} = -19 \\
  d^{(1)}_{1,0} = 1 & d^{(1)}_{1,1} = -1 & d^{(1)}_{1,2} = -7 \\
  d^{(1)}_{2,0} = 7 & d^{(1)}_{2,1} = 3 \\
  d^{(1)}_{3,0} = 29
\end{pmatrix}
\]

For \(k=2\)

\[
D_2 = \begin{pmatrix}
  d^{(0)}_{0,0} = 2 & d^{(2)}_{0,1} = -1 & d^{(2)}_{0,2} = -6 & d^{(2)}_{0,3} = -12 \\
  d^{(2)}_{1,0} = 1 & d^{(2)}_{1,1} = -1 & d^{(2)}_{1,2} = -6 \\
  d^{(2)}_{2,0} = 6 & d^{(2)}_{2,1} = 4 \\
  d^{(2)}_{3,0} = 19
\end{pmatrix}
\]

For \(k=3\)

\[
D_3 = \begin{pmatrix}
  d^{(0)}_{0,0} = 2 & d^{(3)}_{0,1} = -1 & d^{(3)}_{0,2} = -6 & d^{(3)}_{0,3} = -6 \\
  d^{(3)}_{1,0} = 1 & d^{(3)}_{1,1} = -1 & d^{(3)}_{1,2} = -6 \\
  d^{(3)}_{2,0} = 6 & d^{(3)}_{2,1} = 4 \\
  d^{(3)}_{3,0} = 29
\end{pmatrix}
\]

**Step 3:** The Newton interpolating polynomial is

\[p_n(x, y) = X_d^T \cdot D \cdot Y_d = x^3 + 2x^2y - 3xy^2 - y^3 + 2\]

where \(D = D_3\) and

\[X_d = \begin{pmatrix}
  1 \\
  x \\
  x(x-1) \\
  \frac{x(x-1)(x-2)}{6}
\end{pmatrix} \quad \text{and} \quad Y_d = \begin{pmatrix}
  1 \\
  y \\
  y(y-1) \\
  \frac{y(y-1)(y-2)}{6}
\end{pmatrix} \]
According to [11] the total number of operations for all terms is

\[ F(n) = \begin{cases} 
\left( \frac{1}{6} n^3 + \frac{3}{4} n^2 + \frac{5}{6} n \right) a + \left( \frac{1}{12} n^3 + \frac{1}{8} n^2 - \frac{1}{12} n \right) b & n \text{ even} \\
\left( \frac{1}{6} n^3 + \frac{3}{4} n^2 + \frac{5}{6} n + \frac{1}{4} \right) a + \left( \frac{1}{12} n^3 + \frac{1}{8} n^2 - \frac{1}{12} n - \frac{1}{8} \right) b & n \text{ odd}
\end{cases} \]

where \( a \) (resp. \( b \)) is the computational cost of the terms 1 and 2 (resp. 3) of the recursive formula (1).

Therefore, the complexity of Algorithm in triangular basis is \( O(n^3) \).

2.1 Numerical algorithmic implementation of interpolation

In this section we present the algorithmic implementation of the recursive formula (1) on a triangular basis for equidistant interpolation points. Specifically, we study the computation of the differences matrix. The algorithms given below have as input: the table of initial values and as output: the final table of differences. Based on known techniques that are used for the implementation of numerical algorithms in matrix computations [2, 3] we construct three programming approaches in MATLAB.

The first approach is a direct implementation of the recursive formula (1). In each step of \( k \), it executes all the loops for indices \( i \) and \( j \) by selecting the right type from the recursive formula (1). This programming approach is described in Appendix.

The second approach executes in each step of \( k \) the required iterations for each term of the recursive formula (1) separately. The advantage of this implementation is that it reduces the number of iterations. This programming approach is described in Appendix.

The third approach executes in each step of \( k \) the required iterations for each term separately by taking the advantage of the symmetry of the first and second term in the recursive formula (1). This programming approach is described in Appendix.

The number of computations \( F(n) \) in all the above implementations is the same, as given by the complexity in the previous section. Therefore, for the theoretical comparison of these implementations we define as measurement the number \( L(n) \) of the iterations used (\( n \) is the polynomial degree).

The above approaches execute \( L_1(n) \), \( L_1(n) \) and \( L_1(n) \) iterations, respectively, where

\[ L_1(n) = \frac{1}{2} n^3 + \frac{3}{2} n^2 + n, L_2(n) = \begin{cases} 
\frac{1}{4} n^3 + \frac{7}{8} n^2 + \frac{3}{4} n & n \text{ even} \\
\frac{1}{4} n^3 + \frac{7}{8} n^2 + \frac{3}{4} n + \frac{1}{8} & n \text{ odd}
\end{cases}, L_3(n) = \frac{1}{6} n^3 + \frac{1}{2} n^2 + \frac{1}{3} \]

Consequently, for the numbers \( L_1 \), \( L_2 \) and \( L_3 \) we have that \( \lim_{n \to +\infty} \frac{L_1(n)}{L_3(n)} = 3 \) and \( \lim_{n \to +\infty} \frac{L_2(n)}{L_3(n)} = \frac{3}{2} \). From the above, we conclude that if we consider as a comparison criterion the number of iterations \( (L(n)) \) then the third approach is the best choice.
3 Computation of the determinant

Let the two-variable polynomial matrix

\[
A(x, y) = \begin{bmatrix}
a_{11}(x, y) & \cdots & a_{1m}(x, y) \\
\vdots & \ddots & \vdots \\
a_{m1}(x, y) & \cdots & a_{mm}(x, y)
\end{bmatrix} \in \mathbb{R}[x, y]^{m \times m}
\] (5)

The computation of the determinant of the matrix \( A \) is described in the following algorithm

**Algorithm 3.1** Computation of the determinant with evaluation–interpolation technique.

1. **Step 1:** Calculate the upper bound of the total degree of the determinant with the following formula

\[
n = \min \left\{ \sum_{i=1}^{m} \left( \max_{1 \leq j \leq m} \{ \text{deg}[a_{i,j}(x, y)] \} \right), \sum_{j=1}^{m} \left( \max_{1 \leq i \leq m} \{ \text{deg}[a_{i,j}(x, y)] \} \right) \right\}
\]

2. **Step 2:** Evaluate the determinants at the specific set of points \( S^{(n)}_\Delta \). In this step we create the table with the initial values that are the determinants.

3. **Step 3:** Interpolate the values at the set \( S^{(n)}_\Delta \). The interpolating polynomial is the determinant of \( A(x, y) \) and is given by

\[
p(x, y) = X^T \cdot P \cdot Y
\]

The above algorithm is presented in [9] which is structured with the following characteristics: a) it uses random interpolation points for the evaluation and interpolation, b) it uses the interpolation method of the divided differences and c) it uses symbolic operations for the representation of the interpolating polynomial (determinant).

In this work we propose a modified algorithm which it uses the equidistant points and the interpolation method of the differences.

Consequently, the modified algorithm uses only numerical operations which can is implemented to any programming environment which supports numerical operations. The question is if the modified numerical algorithm is better instead to an algorithm which it uses symbolic operations. Therefore, we develop the numerical algorithm in the programming environment MATLAB and we compare the numerical algorithm with the built-in symbolic function of MATLAB \( \text{det()} \). This function uses symbolic operations to find the determinant of a polynomial matrix.

In Table 1 we show the comparison in execution time between the numerical algorithm and the built-in symbolic function of the MATLAB \( \text{det()} \). The performance tests have been implemented in a PC with the following specifications: Intel Core Quad CPU (Q9400) at 2600 GHz with 3.5 Gb RAM, and
Table 1: Execution times (sec) of the computation of the determinant (symbolic/numerical)

<table>
<thead>
<tr>
<th>( m \times d )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.00169</td>
<td>0.00413</td>
<td>0.00701</td>
<td>0.00802</td>
<td>0.00944</td>
<td>0.01291</td>
</tr>
<tr>
<td>3</td>
<td>0.00401</td>
<td>0.00849</td>
<td>0.02471</td>
<td>0.05285</td>
<td>0.09142</td>
<td>0.18062</td>
</tr>
<tr>
<td>4</td>
<td>0.00574</td>
<td>0.01082</td>
<td>0.01695</td>
<td>0.02660</td>
<td>0.03934</td>
<td>0.05869</td>
</tr>
<tr>
<td>5</td>
<td>0.00446</td>
<td>0.02341</td>
<td>0.07541</td>
<td>0.17994</td>
<td>0.38370</td>
<td>0.71765</td>
</tr>
<tr>
<td>6</td>
<td>0.01347</td>
<td>0.02842</td>
<td>0.09989</td>
<td>0.27307</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.00823</td>
<td>0.04896</td>
<td>0.19908</td>
<td>0.54920</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.72510</td>
<td>0.25611</td>
<td>0.75406</td>
<td>1.76506</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0.01359</td>
<td>0.11772</td>
<td>0.44021</td>
<td>1.17330</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.28411</td>
<td>1.79723</td>
<td>6.38249</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3.00744</td>
<td>13.7737</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0.03993</td>
<td>0.34853</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>15.4937</td>
<td>122.274</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0.05579</td>
<td>0.54888</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>294.577</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0.08633</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>2891.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0.12801</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>33493.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.16895</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the release of MATLAB is R2009b.

The dataset of this comparison are polynomial matrices with dimensions \( m \times m \) and the upper bound of the degree for each element is \( d \). Each polynomial entry has random coefficients from the set \( \{-1, 0, 1\} \). The upper bound of the total degree of the determinant is \( n = m \cdot d \).

The performance tests that we have implemented gives rise to the following conclusions: (1) For small matrices \( m \leq 4 \) the symbolic function is better from numerical algorithm, (2) for matrices where \( m \geq 6 \) the numerical algorithm is better from symbolic function, (3) the execution time is increased very quickly when the size of the matrix is increased in symbolic function and (4) the execution time is increased proportional to the size of the matrix in numerical algorithm.

Generally, from the Table 1 we conclude that the numerical algorithm instead to symbolic function has better execution times while the dimensions of the polynomial matrix are increased. When \( n \geq 6 \) the difference between the execution times is very big.
Unfortunately, the numerical computation of the determinant has the following disadvantages: (1) In initial step the estimation of the upper bound of the degree of the polynomial must be the best since the number of the interpolation points is depended from the degree and (2) in the evaluation step we compute determinants which must be less than $10^{16}$ since we have limitations for the numerical precision. In a floating operation system the double precision has in mantissa 53 binary digits or approximately 16 decimal digits.

Consequently, we can extend the numerical precision of the numerical algorithm by using a programming software which support extended double precision in case where the coefficients of the polynomial entries is float numbers or support very long integer in case where the coefficients of the polynomial entries is integer numbers.

4 Conclusions

A new numerical algorithm for the computation of the determinant of a bivariate polynomial matrix are presented. The advantages of this algorithm are: a) the exclusive use of numerical operations which gives the opportunity to develop this numerical algorithm in any programming environment and b) the reduction of execution time instead to function of MATLAB $\text{det()}$ which uses symbolic operations.

The proposed numerical algorithm can found applications in control theory for the computation of: a) the transfer function of two-variable systems, b) the solution of polynomial matrix Diophantine equations, and c) the generalized inverse of two-variable polynomial matrices. Further work, has to be done in the implementation in a parallel programming environment and to find new estimations for the upper bound of the degree for the reduction of the evaluations and interpolation points.

Acknowledgements. The authors wish to acknowledge financial support provided by the Research Committee of the Technological Education Institute of Central Macedonia, under grant SAT/IC/24062015-116/7.

References


Received: June 14, 2015; Published: August 1, 2015