Journal of Computational and Applied Mathematics 236 (2012) 2622–2631

Contents lists available at SciVerse ScienceDirect

Journal of Computational and Applied Mathematics

journal homepage: www.elsevier.com/locate/cam

An algorithm for the finite difference approximation of derivatives with arbitrary degree and order of accuracy

H.Z. Hassan [a,*], A.A. Mohamad [a,b], G.E. Atteia [c]

[a] Mechanical and Manufacturing Engineering, Schulich School of Engineering, University of Calgary, 2500 University Drive, NW, Calgary, Alberta, Canada T2N 1N4

[b] King Saud University, Riyadh, Saudi Arabia

[c] Geomatics Engineering, Schulich School of Engineering, University of Calgary, 2500 University Drive, NW, Calgary, Alberta, Canada T2N 1N4

ARTICLE INFO

Article history:
Received 24 May 2011
Received in revised form 8 September 2011

MSC:
65D25
65L12
68U99

Keywords:
Numerical differentiation
Derivative approximation
Finite difference
Higher degree
Order of accuracy

ABSTRACT

In this paper, we introduce an algorithm and a computer code for numerical differentiation of discrete functions. The algorithm presented is suitable for calculating derivatives of any degree with any arbitrary order of accuracy over all the known function sampling points. The algorithm introduced avoids the labour of preliminary differencing and is in fact more convenient than using the tabulated finite difference formulas, in particular when the derivatives are required with high approximation accuracy. Moreover, the given Matlab computer code can be implemented to solve boundary-value ordinary and partial differential equations with high numerical accuracy. The numerical technique is based on the undetermined coefficient method in conjunction with Taylor's expansion. To avoid the difficulty of solving a system of linear equations, an explicit closed form equation for the weighting coefficients is derived in terms of the elementary symmetric functions. This is done by using an explicit closed formula for the Vandermonde matrix inverse. Moreover, the code is designed to give a unified approximation order throughout the given domain. A numerical differentiation example is used to investigate the validity and feasibility of the algorithm and the code. It is found that the method and the code work properly for any degree of derivative and any order of accuracy.

1. Introduction

Numerical differentiation is an elementary and essential part in scientific modelling and numerical analysis. It is extensively used when we need to calculate the changing rate of data samples or discrete values that have unknown form of their original generating function. This is the case when solving the ordinary or partial differential equations numerically. Many methods have been introduced and discussed to determine the derivatives numerically. These methods can be classified into two approaches [1]. The first approach aims to develop formulas for calculating the derivatives numerically. This includes the Taylor expansion based finite difference method [2–6], the operator method [2,7], the interpolating polynomial method (e.g., Lagrangian, Newton–Gregory, Gauss, Bessel, Stirling, Hermite interpolating polynomials, etc.) [3–5,7], and the lozenge diagram method [8]. The operator, the polynomial interpolation, and the lozenge diagram methods are implicit and are generally based on difference tables constructed from the data sample. Furthermore, they are computationally expensive complicated methods due to the large number of storage memory places required to retain

* Corresponding author.
E-mail addresses: hzhahmed@ucalgary.ca, zoheir_hasan@yahoo.com (H.Z. Hassan), mohamad@ucalgary.ca (A.A. Mohamad), gemattei@ucalgary.ca (G.E. Atteia).

0377-0427/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2011.12.019

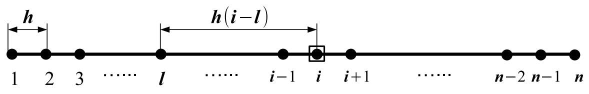

Fig. 1. The numerical stencil used for the approximation.

their difference tables. Moreover, these numerical differentiation formulas are in fact forms equivalent to the Taylor series based approximation [9]. The second approach does not give an explicit formula for the derivative; it just aims to evaluate it by using the function data. The automatic differentiation method [10–12], the regularization method [13–15], and the Richardson extrapolation method [4,16] fall in this category.

The Taylor series based finite difference approximation is used to numerically evaluate the derivative of a function at a grid reference point by using the data samples at the other neighbouring points within the domain. Depending on the degree of differentiation and order of accuracy required, a number of linear equations are generated using the undetermined coefficients method. These equations are to be solved to calculate the weighting coefficients for the differencing formula at the reference point. The procedure becomes more difficult as the degree of the derivative and accuracy required are increased due to the increase in the number of linear equations to be solved. Furthermore, the process should be repeated for each mesh point within the domain. In addition, if the required accuracy is changed for the same derivative, a new system of equations should be derived and solved for a new set of coefficients. Therefore, the implementation of the Taylor series based finite difference approximation is limited to lower degrees and orders. However, there exist some documented sets of pre-calculated finite difference weighting coefficients tabulated for high order and high degree derivatives; see [17]. In this reference, the formula introduced is restricted by the degree and order and the given table limits. Although Bickley [17] referred in his paper to the repeated application of the formula when higher degrees are required, this will be at the expense of the order of accuracy. Another attempt to tabulate weights for many degrees of derivatives and approximated to high order of accuracy was the focus of work done in [18] which was more extensive than the work done in [17].

The direct use of the finite difference method is computationally very expensive when higher degree derivatives with lesser errors are required. However, this method becomes more attractive if a closed explicit algebraic form of the coefficients is found. Gregory [19] has used the undetermined coefficients method to get the coefficients in his formula presented for calculating the derivatives. The method involved the solution of systems of linear equations, or equivalently, the inversion of a certain Vandermonde matrix. This work was extended in [20] to obtain explicit representations for the same coefficients in terms of Stirling numbers which are extensively tabulated. On the basis of Taylor series, Khan and Ohba [6,9,21,22] have presented the explicit forward, backward and central difference formulas for finite difference approximations with arbitrary orders for the first derivative, and the central difference approximations for higher derivatives. However, these explicit forward, backward and central difference formulas do not give the whole circumstances. The explicit approximation formulas for derivatives near the ends of an interval also need to be developed [1].

Previously mentioned techniques that were developed for finding explicit formula expressions for the derivatives of grid based functions have been of considerable complexity. Although these algorithms are correctly proven and tested, in real calculations they fail to provide accurate values for the weighting coefficients when more higher approximation orders are applied. The reason behind this is the resulting ill-conditioned system of linear equations with a large size Vandermonde matrix when solving for the coefficients. Therefore, it is hard to directly get the matrix inverse or even use its determinant in finding the coefficients. That is due to the fact that the determinant of the Vandermonde matrix goes to infinity as the matrix size is increased. One other significant remark regarding the previously published coefficient tables or closed formulas is related to the accuracy of the approximation. The accuracies of the forward and backward difference approximations are obviously less than those of the other points. This is because forward and backward formulas use data only on one side of a reference point and the formulas for the other points use data on both sides of reference points. In some special cases when an even degree of the derivative with an odd order of approximation accuracy are required, the point at the grid middle will be of higher order of accuracy than other points within the stencil. This issue may concern some researchers who aim to get a unified approximation order across the whole solution domain.

The present paper aims to introduce a dependable explicit formula that can be used to calculate the numerical approximations of arbitrary degree and order of derivative. The algorithm does not use the Vandermonde matrix or its determinant in finding the weighting coefficients. Furthermore, the explicit formula imposes a lesser calculation burden, and needs less computing time and storage, for estimating the derivatives than the other methods stated above. Moreover, the easy and user-friendly computer code is also handy for helping other researchers to use it directly in their calculations. The code is recommended for use by any researcher who is concerned with derivative approximation, especially in solving ODEs and PDEs. That is because it is just a programming function that can be called with input of the sampling data, the degree, and the order, and returns the corresponding derivatives at all the grid points to the main program. Finally, the proposed method gives a unified order of accuracy for the derivative at all the grid nodes.

2. Notation

Suppose that $x_1, x_2, \dots, x_n$ are n different real numbers and $x_1 < x_2 < \cdots < x_n$. Let f to be a continuous differentiable function in the interval $[x_1, x_n]$ and its values be given at these numbers. Then, for any one x_i the function value will be denoted as f_i. Moreover, for an equal tabular interval for the function f, the sampling period will be denoted by h where $h = x_{i+1} - x_i$ and $x_i = x_l + (i - l)h$ (for $h \geq 0$ and i, l will be denoted as an integer everywhere). Fig. 1 shows the numerical stencil which will be used in the algorithm derivation; the grid point i is considered the reference or base node. The derivative of the function f defined at the reference node i and at an arbitrary derivation degree m and an arbitrary order of accuracy O will be denoted by $f_i(m, O)$, where the required differentiation degree and order of accuracy are constrained by the relation $n \geq m + O$.

3. Derivation of the algorithm

By using the undetermined coefficients method, the mth-degree derivative at the base point i can be approximated up to the order of accuracy O by using a stencil with $n = m + O$ points as follows:

$$f_i(m, O) = \sum_{l=0}^{n} C_{i,l}(m, O) \cdot f_l, \quad i = 1, 2, 3, \ldots, n \tag{1}$$

where the n unknown factors, $C_{i,l}(m, O)$, are the weighting coefficients for the derivative. The values of the function f at all other nodes $l \neq i$ can be expressed as Taylor series in terms of the reference point i as given in Eq. (2):

$$\begin{aligned} f_l &= \sum_{k=0}^{\infty} \frac{(l-i)^k}{k!} h^k f_i^{(k)} \\ &= f_i + \frac{(l-i)}{1!} h f_i' + \frac{(l-i)^2}{2!} h^2 f_i'' + \frac{(l-i)^3}{3!} h^3 f_i^{(3)} + \cdots + \frac{(l-i)^n}{n!} h^n f_i^{(n)} + \cdots. \end{aligned} \tag{2}$$

Substituting into Eq. (1) we get

$$\begin{aligned} f_i(m, O) = \sum_{l=1}^{l=n} C_{i,l}(m, O) \Bigg[& f_i + \frac{(l-i)}{1!} h f_i' + \frac{(l-i)^2}{2!} h^2 f_i'' + \frac{(l-i)^3}{3!} h^3 f_i^{(3)} + \cdots + \frac{(l-i)^{(n-1)}}{(n-1)!} h^{(n-1)} f_i^{(n-1)} \\ & + \frac{(l-i)^n}{n!} h^n f_i^{(n)} + \cdots \Bigg]. \end{aligned} \tag{3}$$

This can be rearranged to take the following form:

$$\begin{aligned} f_i(m, O) = {} & f_i \left[\sum_{l=1}^{l=n} C_{i,l}(m, O) \right] + f_i' \frac{h}{1!} \left[\sum_{l=1}^{l=n} C_{i,l}(m, O) \cdot (l-i) \right] + f_i'' \frac{h^2}{2!} \left[\sum_{l=1}^{l=n} C_{i,l}(m, O) \cdot (l-i)^2 \right] \\ & \vdots \\ & + f_i^{(n-1)} \frac{h^{(n-1)}}{(n-1)!} \left[\sum_{l=1}^{l=n} C_{i,l}(m, O) \cdot (l-i)^{(n-1)} \right] + E_i(m, O). \end{aligned} \tag{4}$$

The term $E_i(m, O)$ in the previous equation represents the remainder or error term in the approximation of $f_i(m, O)$. Eq. (4) can be rewritten in a more compact form as follows:

$$f_i(m, O) = \sum_{g=1}^{g=n} \left[f_i^{(g-1)} \frac{h^{(g-1)}}{(g-1)!} \sum_{l=1}^{l=n} \Big(C_{i,l}(m, O) \cdot (l-i)^{(g-1)} \Big) \right] + E_i(m, O). \tag{5}$$

Eq. (5) represents the exact value of the mth derivative of the function at i. An approximated value up to the order of accuracy of O can be obtained by neglecting the remainder term $E_i(m, O)$. Therefore, we can write

$$f_i(m, O) \approx \sum_{g=1}^{g=n} \left[f_i^{(g-1)} \frac{h^{(g-1)}}{(g-1)!} \sum_{l=1}^{l=n} \Big(C_{i,l}(m, O) \cdot (l-i)^{(g-1)} \Big) \right]. \tag{6}$$

We equate the coefficient of any degree derivative from the right hand side to the corresponding coefficient of the same degree derivative from the left hand side. Hence, a system of n linear equations in the unknown weighting coefficients can be represented by the following equation:

$$b_g \frac{(g-1)!}{h^{(g-1)}} = \sum_{l=1}^{l=n} \Big[C_{i,l}(m, O) \cdot (l-i)^{(g-1)} \Big], \quad g = 1, 2, 3, \ldots, n \tag{7}$$

where

$$b_g = \begin{cases} 0 & \text{if } g \neq m+1 \\ 1 & \text{if } g = m+1. \end{cases}$$

Therefore,

$$\sum_{l=1}^{l=n}\left[C_{i,l}(m,O)\cdot(l-i)^{(g-1)}\right] = \begin{cases} 0 & g = 1,2,3,\ldots,n \text{ and } g \neq m+1 \\ \dfrac{m!}{h^m} & g = m+1. \end{cases} \tag{8}$$

This equation can be represented in matrix form as follows:

$$\overbrace{\begin{bmatrix} (1-i) & (2-i) & \cdots & (l-i) & \cdots & (n-i) \\ (1-i)^1 & (2-i)^1 & \cdots & (l-i)^1 & \cdots & (n-i)^1 \\ (1-i)^2 & (2-i)^2 & \cdots & (l-i)^2 & \cdots & (n-i)^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (1-i)^{(g-1)} & (2-i)^{(g-1)} & \cdots & (l-i)^{(g-1)} & \cdots & (n-i)^{(g-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (1-i)^{(n-1)} & (2-i)^{(n-1)} & \cdots & (l-i)^{(n-1)} & \cdots & (n-i)^{(n-1)} \end{bmatrix}}^{\mathbf{A}_i(m,O)} \overbrace{\begin{bmatrix} C_{i,1}(m,O) \\ C_{i,2}(m,O) \\ C_{i,3}(m,O) \\ \vdots \\ C_{i,g}(m,O) \\ \vdots \\ C_{i,n}(m,O) \end{bmatrix}}^{\mathbf{C}_i(m,O)} = \frac{m!}{h^m} \overbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_g \\ \vdots \\ b_n \end{bmatrix}}^{\mathbf{b}_i(m,O)}. \tag{9}$$

Therefore,

$$\mathbf{A}_i(m,O)\mathbf{C}_i(m,O) = \frac{m!}{h^m}\mathbf{b}_i(m,O).$$

Solving for $\mathbf{C}_i(m,O)$, then

$$\mathbf{C}_i(m,O) = \frac{m!}{h^m}\mathbf{A}_i^{-1}(m,O)\mathbf{b}_i(m,O). \tag{10}$$

It is noted from Eq. (9) that the matrix $\mathbf{A}_i(m,O)$ is the transpose of the Vandermonde matrix $\mathbf{V}_i(m,O)$. Therefore, Eq. (10) can be written in the following form:

$$\mathbf{C}_i(m,O) = \frac{m!}{h^m}\left[\mathbf{V}_i^T(m,O)\right]^{-1}\mathbf{b}_i(m,O) = \frac{m!}{h^m}\left[\mathbf{V}_i^{-1}(m,O)\right]^T\mathbf{b}_i(m,O). \tag{11}$$

In order to get an explicit formulation for the coefficients $\mathbf{C}_i(m,O)$, the Vandermonde matrix inverse $\mathbf{V}_i^{-1}(m,O)$ should be expressed in a closed form. El-Mikkawy [23] has introduced an explicit closed form expression for the inverse matrix of the generalized Vandermonde matrix by using the elementary symmetric functions. After correcting this expression for some minor mistakes, the Vandermonde matrix inverse is given by

$$\mathbf{V}_i^{-1}(m,O) = \mathbf{m}_i(m,O) = \left[m_{l,r}\right], \quad m_{l,r} = (-1)^{n-l}\frac{\sigma_{n-l+1,r}^{(n)}}{\prod\limits_{\substack{p=1\\p\neq r}}^{p=n}(\upsilon_r-\upsilon_p)} \tag{12}$$

where the elementary symmetric functions $\sigma_{i,j}^{(n)}$ are defined as

$$\sigma_{i,j}^{(n)} = \begin{cases} \sum\limits_{\substack{r_1=1\\r_1\neq j}}^{r_1=n}\ \sum\limits_{\substack{r_2=r_1+1\\r_2\neq j}}^{r_2=n}\ \sum\limits_{\substack{r_3=r_2+1\\r_3\neq j}}^{r_3=n}\cdots\sum\limits_{\substack{r_{i-1}=r_{i-2}+1\\r_{i-1}\neq j}}^{r_{i-1}=n}\ \prod\limits_{h=1}^{h=i-1}\upsilon_{rh} & i\neq 1 \\ 1 & i=1. \end{cases} \tag{13}$$

The n distinct parameters $\upsilon_1, \upsilon_2, \upsilon_3, \ldots, \upsilon_n$, in Eq. (13) are defined at the reference node i by the following:

$$\upsilon_k = k - i, \quad k = 1,2,3,\ldots,n.$$

The transpose of the Vandermonde inverse matrix is therefore given by

$$\left[\mathbf{V}_i^{-1}(m,O)\right]^T = \boldsymbol{\omega}_i(m,O) = \left[\omega_{l,r}\right], \quad \omega_{l,r} = (-1)^{n-r}\frac{\sigma_{n-r+1,l}^{(n)}}{\prod\limits_{\substack{p=1\\p\neq l}}^{p=n}(\upsilon_l-\upsilon_p)}. \tag{14}$$

Substituting from this equation into Eq. (11), we get

$$\mathbf{C}_i(m,O) = \frac{m!}{h^m}\boldsymbol{\omega}_i(m,O)\mathbf{b}_i(m,O) = \frac{m!}{h^m}\left[\omega_{l,m+1}\right]. \tag{15}$$

Therefore,

$$C_{i,l}(m,O) = \frac{m!}{h^m}\omega_{l,m+1} = \frac{m!}{h^m}(-1)^{n-m-1}\frac{\sigma^{(n)}_{n-m,l}}{\prod\limits_{\substack{p=1\\p\neq l}}^{p=n}(\upsilon_l-\upsilon_p)}. \tag{16}$$

Now, the multiple-product term in the denominator of the right hand side of Eq. (16) can be expanded and simplified as follows:

$$\begin{aligned}\prod_{\substack{p=1\\p\neq l}}^{p=n}(\upsilon_l-\upsilon_p) &= \prod_{\substack{p=1\\p\neq l}}^{p=n}[(l-i)-(p-i)] = \prod_{\substack{p=1\\p\neq l}}^{p=n}(l-p)\\ &= (l-1)(l-2)\cdots(3)(2)(1)(-1)(-2)(-3)\cdots[-(n-l-1)][-(n-l)]\\ &= (-1)^{n-l}(l-1)!(n-l)! \end{aligned} \tag{17}$$

Substituting into Eq. (16) we get the final explicit closed form for the weighting coefficients in Eq. (1):

$$C_{i,l}(m,O) = (-1)^{l-m-1}\frac{m!}{h^m(l-1)!(n-l)!}\sigma^{(n)}_{n-m,l}. \tag{18}$$

The remainder term $E_i(m,O)$ in Eq. (4) can be expressed as follows:

$$\begin{aligned}E_i(m,O) &= \sum_{s=n}^{\infty} f_i^{(s)}\frac{h^s}{s!}\sum_{l=1}^{n}C_{i,l}(m,O)(l-i)^s\\ &= \sum_{s=n}^{\infty} f_i^{(s)}\frac{m!h^{s-m}}{s!}\sum_{l=1}^{n}\frac{(-1)^{l-m-1}(l-i)^s\sigma^{(n)}_{n-m,l}}{(l-1)!(n-l)!}.\end{aligned} \tag{19}$$

4. Listing and description of the code

In this section, we will be discussing the Matlab computer program developed to numerically calculate the arbitrary degree derivative of any continuous function f, approximated with any arbitrary order of accuracy. The code is shown as Listing 1. The Matlab function Sig(v, n), line 94 in the listing, calculates the matrix $\sigma^{(n)}_{i,j}$ as given by Eq. (13). Due to symmetry, the elements of the jth column of $\sigma^{(n)}_{i,j}$ may be obtained by using

$$\sigma^{(n)}_{i,j} = \sigma^{(n)}_{i,1}|_{\upsilon_j\to\upsilon_1}. \tag{20}$$

The notation $\upsilon_j \to \upsilon_1$ in Eq. (20) means that, for specific i and j, $\sigma^{(n)}_{i,j}$ may be obtained from the algebraic expression for $\sigma^{(n)}_{i,1}$ by replacing each υ_j by υ_1 in the expression for $\sigma^{(n)}_{i,1}$ (see El-Mikkawy [23]). Then, this function is called by another Matlab function called Coeff(m, O, h), line 80 in the code, which calculates the weighting coefficients matrix for all the pre-specified stencil $n = m + O$ nodes as given by Eq. (18). Now, the weighting coefficients are available for the third function f_derivative(m, O, f, h), line 22, for performing the remaining calculations for the required differentiation across the whole domain with N nodes and the specified degree m, order O and data step size h. First, if the approximation order is different, this function adjusts the numerical stencil for a unified order of accuracy. This situation usually arises when m is even and O is an odd number. In other words, we have an odd number n of stencil nodes and the middle node in the stencil is approximated with a central difference approximation. Therefore, it attains a one higher order of accuracy than other nodes in the stencil. In order to obtain a unified approximation order over all the stencil nodes, another node is added to the stencil to eliminate the central difference approximation as shown in the code, lines 31–48. The objective of the remaining code lines 52–78 is to generate the derivative over the real domain of N nodes by using the pre-calculated weighting coefficients over the numerical stencil. Finally, the calculated differentiation column vector is passed to the main program.

Listing 1. The Matlab computer program.

```

% Subject: An algorithm for the finite difference approximation of
%          derivatives with arbitrary degree and accuracy order.
% Date: May 10, 2011
% By: H. Z. Hassan

% This is the main program 'main.m'
% Generating tabular data for tne numerical experimentation
% the used function is f = exp(x) with N = 50 node and a fixed
% period h = 0.1 starting at the value x1 = 0.0

clear all; clc; format long;
N = 50;
h = 0.1;
x1 = 0.0;
x = [x1:h:x1+(N−1)*h];
f = exp(x);
f = f(:);

% Evaluating the function derivatives of degree m and order O
m = 3;
O = 8;
Derivative = f_derivative(m,O,f,h);

% Exact error calculation
Exact_Error =  Derivative − f;
% ==============================================================
function Diff_fun = f_derivative(m,O,f,h)

n = m + O;
C = Coeff(m,O,h);
Bickley_scale = h^m * factorial(n−1)/factorial(m);
Bickley_coeff = C * Bickley_scale;
Bickley_coeff = int64(Bickley_coeff);

% Modifying the numerical stencil for a unified accuracy
if (mod(m,2)==0 && mod(O,2)~=0)
    np = n +1;
    j = np/2;
    for i = 1:j−1
        coef_u(i,:) = [C(i,:) 0];
    end

    coef_m(1,:) = [0 C(j−1,:)];
    coef_m(2,:) = [C(j+1,:) 0];
    for i = j+2:np
        coef_l(i−j−1,:) = [0 C(i−1,:)];
    end

    C = [ coef_u;coef_m;coef_l];

else
    np = n;
end

% Generating the derivative over the real domain of N nodes

[N b]  = size(f);

if N<np
    error('The size of the given data should be greater than or equal to %d . \n\t Try using larger set of data or ↩
        decrease the accuracy order.', np);
end

if mod(np,2)==0
    j = np/2;
else
    j = (np+1)/2;
end

U = [C(1:j−1,:) zeros(j−1,N−np)];
L = [zeros(np−j,N−np) C(j+1:np,:) ];
k = C(j,:);
M = zeros(N−np+1,N);

for i =1:N−np+1
      for j = 1:N
          if i==j
```

```
                M(i,j:j+np−1) = k;
            end
        end
end

coef = [U;M;L];
Diff_fun = coef*f;
% ============================================================
function C = Coeff(m,O,h)
% Weighting coefficients calculation

n = m + O;
k = [1:n];
k = k(:);
for i =1:n
    v = k − i;
    sigma = Sig(v,n);
   for l=1:n
       C(i,l) = ((−1)^(l−m−1)*factorial(m)/(factorial(l−1)*factorial(n−l)* h^m ) )* sigma(n−m,l);
    end
end
% ============================================================
function S = Sig(v,n)
S = ones(n,n);
s = S;
for k = 1:n
   vv = v;
   vv(k) = v(1);
   for i =2:n
       s(i,i)=s(i−1,i−1)*vv(i);
        if i >2
           for j=i−1:−1:2
               s(j,i)=s(j−1,i−1)*vv(i)+s(j,i−1);
           end
        end
   end
   S(:,k) = s(:,n);
end
return
```

5. Comparison with other existing methods and formulas

In Section 3, we have presented the mathematical derivation of our new method introduced for the numerical calculation of the derivatives with any desired order of accuracy and any required degree. On the basis of this method, the derivative $f_i(m, O)$ is constructed through the undetermined coefficient method as expressed by Eq. (1) with the weighting coefficients $C_{i,l}(m, O)$ given by the formula in Eq. (18) in terms of the elementary symmetric functions $\sigma_{i,j}^{(n)}$ defined by Eq. (13). Generally, the algebraic derivation of the formula provided is more straightforward and transparent as well as being very simple and without any mathematical complications. This makes the problem under consideration easy and clear for other researchers who are novices and only need to apply the method in their researches, as well as those who are expert. The computer code is handy and is provided in the form of a programming function which can simply be called and executed within the main application by using the required derivative degree m, the order of accuracy needed O, the function values f, and the regular spacing size h as inputs. Consequently, it is very suitable for being implemented by others, especially when solving ordinary or partial differential equations, because there is no need to construct finite difference schemes that require more effort, in particular when high orders of accuracy and high differentiation degrees are needed.

It is evident that the method presented is explicit and the proven formula is given in a closed form. As a consequence, it is computationally less expensive compared with those of other implicit methods such as the operator method [2,7], the interpolating polynomial method [3–5,7], and the lozenge diagram method [8]. Moreover, it is not restricted by the degree and order unlike the tabulated data of [17,18]. From the comparison between our method and the method discussed in [19], we can say that our theory is superior and advanced. That is because the latter method requires the solution of a linear system of equations to determine the coefficients. This imposes more computational burden as well as leading to large numerical errors and less accuracy of the calculated values of the coefficients, especially in the case of high degree derivatives and high orders of accuracy. Although Spitzbart and Macon [20] have eliminated the problems discussed in relation to the study of Gregory [19], their proposed algorithm has a disadvantage compared with our method presented here. Their model developed for calculating the coefficients constitutes three summation formulas having to be evaluated in sequence. These formulas require evaluation of the Stirling numbers of the first kind, a process that involves the construction of tables. As a result, the Spitzbart and Macon method requires more memory and is computationally expensive.

In fact, the algorithm presented in this paper has an extensive and comprehensive expression, and is not restricted by either the derivative degree or the finite difference scheme type, unlike the studies done in [6,9,21,22], which have more limitations. For example, Khan and Ohba [21] have presented the explicit forward, backward and central difference formulas for finite difference approximations with arbitrary orders but they apply only for the first-degree and second-degree derivatives, except for a formula for any order and higher degree which is limited to the central finite difference approximation type and therefore is not applicable at points near the stencil right and left ends. That is because the central finite difference scheme uses the function values from both sides of the base point. Moreover, the formula in [9] applies for the first-degree derivative only. In another work [6], the restriction was related to the nodes near the ends because the formula was based on the central finite difference approximation.

In his study, Li [1] has introduced general explicit finite difference formulas with arbitrary order of accuracy for approximating the first-degree and higher derivatives. His technique was named later as Li's method (see Hickernell and Yang [24]) or Li's theorem (see Bai et al. [25]). Since our new formula for the weighing coefficients has a similar construction to the formulas given in [1], it is important to clarify the following general major differences:

1. In Li's method, the author has based the mathematical proof of the weighting coefficients formulas on solving the linear algebraic system of equations by using Cramer's rule. Through the implementation of the generalized Vandermonde determinant and the use of the basic properties of that determinant, Li has obtained the formulas for the weighting coefficients. However, in our methodology presented here and as seen in Section 3, we have solved the linear algebraic system of equations through the matrix inverse method. This is done by implementation of the explicit closed formula for the inverse of the Vandermonde matrix given in [23].
2. The formulas for weighting coefficients in Li's method are given in terms of the root coefficient, as named by Li in his code [26], which is given by a set of different expressions in correspondence with the derivative degree required to be calculated; see Eq. (2-1) in [1]. In comparison, our formula for the weighting coefficients, Eq. (18), is proven to be dependent on the elementary symmetric function which is defined by Eq. (13) and is different from the root coefficient in Li's method.

It should be mentioned that Li's theorem has the advantage of applicability to equally and to unequally spaced data, as compared with our method presented here which applies only for equally spaced data. However, we stress that our new method is far better than Li's method when equally spaced data are under consideration. The reasons behind that are discussed in the following points:

1. Although the algorithm developed in [1] can be used to evaluate the derivatives numerically with any arbitrary order of accuracy, it is restricted by the required derivative degree. In other words, it can only be used with some predefined derivative degrees and not with any arbitrary degree. The reason is that Li did not express the root coefficient in his formulas in terms of a single closed form function. Instead, he defined this coefficient in terms of multi-expressions corresponding to the derivative degree; see Eq. (2-1) in [1]. As a consequence, Li's method is not a closed form method. If the required derivative degree is changed, the corresponding expression for the root coefficient must be changed as well. Therefore, this makes the method less convenient. Additionally, more efforts are exerted in writing a programming code for Li's method. That is because, for each required degree derivative approximation, the corresponding expression for the root coefficient has to be added to the code. Consequently, the programming code required to evaluate up to the eighth-degree differentiation and based on Li's method will be very long and becomes even longer if evaluations of other degrees are required; see [26]. However, the method introduced in our present study is superior and can be adequately employed to evaluate the derivative with any arbitrary order and also any arbitrary degree. That is because the formula presented is in closed form as regards the order as well as the degree. Roughly speaking, our algorithm presented here is much more convenient, more dependable, and simpler than Li's theorem.
2. It is well known that the computational burden is increased by increasing the running and execution times, the amount of storage memory used, and the size of the written programming code. These factors are considered judging criteria for using one method rather than another. Therefore, we can deduce that Li's model is computationally expensive compared with our model presented in this study. This can be shown by comparing the piece of our code required to calculate $\sigma_{i,j}^{(n)}$ for any arbitrary degree (see lines 94–110 in Listing 1) with that required to calculate the root coefficients of Li's method up to only the eighth-degree differentiation (see [26]). Moreover, the necessity of using a large number of nested loopings when evaluating the root coefficients in Li's formulas results in a longer running time. In the case of high degree derivatives, this number of nested looping blocks and the associated code execution time become large. For example, to approximate the mth-degree derivative of some data by using Li's technique, we need a total of $(m+1)$ nested looping structures in the programming code; see [26].

6. Numerical experimentation and code validation

In order to prove the validity of our algorithm and computer code, the calculated weighting coefficients obtained when using our method are compared with the coefficients derived in [17]. According to Bickley's formula, his tabulated coefficients are scaled by a factor of $\lambda = h^m(n-1)!/m!$. Therefore, we scale our calculated weighting coefficients using the same scale, λ, in order to make a comparison with his results, as shown in the listing code lines 26–28. Our calculated scaled coefficients based on a fourth-degree derivative with an approximation of the fifth order of accuracy, $m = 4$, $O = 5$, as an example, are given in Table 1. These coefficients are the same coefficients as were tabulated in [17] and this result verifies the validity of the algorithm and computer program presented. To examine our technique for finding the discrete function

Table 1
Scaled weighting coefficients, $\tilde{C}_{i,l} = \lambda * C_{i,l}$.

Node i	$\tilde{C}_{i,1}$	$\tilde{C}_{i,2}$	$\tilde{C}_{i,3}$	$\tilde{C}_{i,4}$	$\tilde{C}_{i,5}$	$\tilde{C}_{i,6}$	$\tilde{C}_{i,7}$	$\tilde{C}_{i,8}$	$\tilde{C}_{i,9}$
1	22 449	−147 392	428 092	−720 384	769 510	−534 464	235 452	−60 032	6 769
2	6 769	−38 472	96 292	−140 504	132 510	− 83 384	34 132	−8 232	889
3	889	−1 232	−6 468	21 616	−28 490	20 496	−8 708	2 128	− 231
4	− 231	2 968	−9 548	12 936	−7 490	616	1 092	− 392	49
5	49	− 672	4 732	−13 664	19 110	−13 664	4 732	− 672	49
6	49	− 392	1 092	616	−7 490	12 936	−9 548	2 968	− 231
7	− 231	2 128	−8 708	20 496	−28 490	21 616	−6 468	−1 232	889
8	889	−8 232	34 132	−83 384	132 510	−140 504	96 292	−38 472	6 769
9	6 769	−60 032	235 452	−534 464	769 510	−720 384	428 092	−147 392	22 449

Table 2
Comparison between the calculated and the exact derivatives for $f = \exp(x)$.

Derivative	Exact value	Numerical value	Approximation error
$f_{x=0.0}(1, 5)$	1.00000000000000000000	1.00000206919171397146	0.20692E−005
$f_{x=1.9}(1, 14)$	6.68589444227927032216	6.68589444227927209852	0.17764E−014
$f_{x=4.4}(2, 7)$	81.4508686649681408198	81.4508686493600890799	−0.15608E−007
$f_{x=4.4}(2, 12)$	81.4508686649681408198	81.4508686649690503145	0.90949E−012
$f_{x=0.3}(3, 2)$	1.34985880757600318347	1.35323683113972249272	0.33780E−002
$f_{x=2.4}(3, 10)$	11.0231763806416047657	11.0231763806554603491	0.13856E−010
$f_{x=2.9}(5, 7)$	18.1741453694430674659	18.1741453275608364493	−0.41882E−007
$f_{x=1.4}(8, 4)$	4.05519996684467542991	4.05518272519111633301	−0.17242E−004

derivatives within a given data sample, the exponential function is used to do this test due to its simplicity in analytical differentiation. The data sample f to be used by the program is generated at a total of $N = 50$ equispaced nodes beginning at $x_1 = 0$ and with the step size $h = 0.1$; see the code listing lines 12–17. The tabulated data sample f is introduced into the Matlab code and the derivatives of different degrees and orders of accuracy are calculated. Table 2 shows a sample of the code outputs with a floating point precision of 20 decimal numbers. The calculated arbitrary degree and order derivatives of f at randomly chosen locations in the solution domain are compared with the exact analytical derivatives. As seen from Table 2, the numerical errors from the approximation are of the same selected order of accuracy, with error $\approx h^O$. Moreover, it is important to use high precision of calculations when calculating high degree and high order derivatives. That is because the output is highly sensitive to the calculation precision used.

7. Conclusions

In this present work, we introduced an algorithm and a computer program for calculating numerically the derivatives of discrete functions with arbitrary degree and order of accuracy using a closed explicit formula. The derivation of the algorithm is based on the undetermined coefficients method and the closed form of the Vandermonde matrix inverse as a function of the elementary symmetric functions. The numerical testing showed good results when various degree differentiations with various approximation orders were calculated.

References

[1] J. Li, General explicit difference formulas for numerical differentiation, J. Comput. Appl. Math. 183 (1) (2005) 29–52.
[2] C. Jordan, Calculus of Finite Differences, second ed., Chelsea, New York, 1950.
[3] L. Collatz, The Numerical Treatment of Differential Equations, second ed., Springer, Berlin, 1966.
[4] S.C. Chapra, R.P. Canale, Numerical Methods for Engineers, sixth ed., McGraw-Hill, 2010.
[5] J.D. Hoffman, Numerical Methods for Engineers and Scientists, second ed., Marcel Dekker, New York, 2001.
[6] I.R. Khan, R. Ohba, Taylor series based finite difference approximations of higher-degree derivatives, J. Comput. Appl. Math. 154 (2003) 115–124.
[7] G. Dahlquist, A. Bjorck, Numerical Methods, Prentice Hall, Englewood Cliffs, 1974.
[8] C.F. Gerald, P.O. Wheatley, Applied Numerical Analysis, fifth ed., Addison-Wesley, 1994.
[9] I.R. Khan, R. Ohba, New finite difference formulas for numerical differentiation, J. Comput. Appl. Math. 126 (2000) 269–276.
[10] L.B. Rail, Automatic Differentiation: Techniques and Applications, in: Lect. Notes Comput. Sci., vol. 120, 1981.
[11] H.B. Christian, H.M. Bücker, P. Hovland, U. Naumann, J. Utke, Advances in Automatic Differentiation, Springer-Verlag, Berlin, Heidelberg, 2008.
[12] A. Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, second ed., SAIM, Philadelphia, PA, 2008.
[13] S. Lu, S.V. Pereverzev, Numerical differentiation from a viewpoint of regularization theory, Math. Comp. 75 (256) (2006) 1853–1870.
[14] J. Cullum, Numerical differentiation and regularization, SIAM J. Numer. Anal. 8 (2) (1971) 254–265.
[15] A.G. Ramm, A.B. Smirnova, On stable numerical differentiation, Math. Comp. 70 (235) (2001) 1131–1154.
[16] A. Sidi, Extrapolation methods and derivatives of limits of sequences, Math. Comp. 69 (229) (1999) 305–324.
[17] W.G. Bickley, Formulae for numerical differentiation, Gaz. Math. 25 (263) (1941) 19–27.
[18] H.B. Keller, V. Pereyra, Symbolic generation of finite difference formulas, Math. Comp. 32 (144) (1978) 955–971.
[19] R.T. Gregory, A method for deriving numerical differentiation formulas, Amer. Math. Monthly 64 (2) (1957) 79–82.
[20] A. Spitzbart, N. Macon, Numerical differentiation formulas, Amer. Math. Monthly 64 (10) (1957) 721–723.
[21] I.R. Khan, R. Ohba, N. Hozumi, Mathematical proof of closed form expressions for finite difference approximations based on Taylor series, J. Comput. Appl. Math. 150 (2003) 303–309.

[22] I.R. Khan, R. Ohba, Closed-form expressions for the finite difference approximations of first and higher derivatives based on Taylor series, J. Comput. Appl. Math. 107 (1999) 179–193.
[23] M.E.A. El-Mikkawy, Explicit inverse of a generalized Vandermonde matrix, Appl. Math. Comput. 146 (2003) 643–651.
[24] F.J. Hickernell, S. Yang, Simplified analytical expressions for numerical differentiation via cycle index, J. Comput. Appl. Math. 224 (2009) 433–443.
[25] H. Bai, A. Xu, F. Cui, Representation for the Lagrangian numerical differentiation formula involving elementary symmetric functions, J. Comput. Appl. Math. 231 (2009) 907–913.
[26] Dr. Jianping Li's Home Page, Subroutines, numerical differentiation. http://www.lasg.ac.cn/staff/ljp/subroutine/differentiation_uneven.f (last accessed 07.09.11).