A New Algorithm to Determine Adomian Polynomials for Nonlinear Polynomial Functions

Mithun Bairagi [a*]

[a]*Mangal Chandi High School, Khosalpur, 722206, Patrasayer, Bankura, West Bengal, India.*

Author's contribution

The sole author designed, analysed, interpreted and prepared the manuscript.

Original Research Article

Abstract

We present a new algorithm by which the Adomian polynomials can be determined for scalar-valued nonlinear polynomial functional in a Hilbert space. This algorithm calculates the Adomian polynomials without the complicated operations such as parametrization, expansion, regrouping, differentiation, etc. The algorithm involves only some matrix operations. Because of the simplicity in the mathematical operations, the new algorithm is faster and more efficient than the other algorithms previously reported in the literature. We also implement the algorithm in the MATHEMATICA code. The computing speed and efficiency of the new algorithm are compared with some other algorithms in the one-dimensional case.

*Corresponding author: E-mail: bairagirasulpur@gmail.com;

1 Introduction

The Adomian Decomposition Method (ADM) [1, 2, 3, 4, 5] has gained huge attention in different fields of science and engineering for solving nonlinear functional equations. In practice, many nonlinear problems do not admit exact solutions, and in most cases, we have to find approximate solutions by employing numerical or analytical approximation techniques. The ADM is a reliable technique for solving wide classes of nonlinear systems, including ordinary differential, partial differential, integro-differential, algebraic, differential-algebraic, non-integer-order differential, integral equations, and so on [6, 7, 8, 9, 10, 11, 12]). This technique can provide an analytical approximation to the exact solutions in the series form that converge very rapidly [13, 14, 15]. The Adomian decomposition method coupled with the Laplace transform, develops a powerful method called the Laplace Adomian decomposition method (LADM). LADM has also been used in numerous articles to find the numerical solution of fractional-order nonlinear differential equations, as can be seen in [16, 17, 18, 19, 20]. Following [6, 7, 21], let us recall the basic ideas of the Adomian Decomposition Method. We consider a nonlinear ODE in order p with independent variable x (real and scalar) and dependent variable u in the general form [6, 7]

$$\mathcal{F}u = g(x), \tag{1.1}$$

where $\mathcal{F}$ is the nonlinear operator from a Hilbert space H into H. In ADM, $\mathcal{F}$ is assumed to be decomposed into

$$Lu + Ru + Nu = g(x), \tag{1.2}$$

where L is the highest-order linear differential operator $L[.] = \frac{d^p}{dx^p}[.]$ which is assumed to be invertible, R is a linear differential operator containing the linear derivatives of less order than L, N is a nonlinear operator containing all other nonlinear terms, $g(x) \in H$ is a given analytic function. Here we should note that the choice of the operator L is not generally unique [22, 23, 24]. For example, in [23], A. Wazwaz has chosen the linear differential operator $L[.]$ as $L[.] = x^{-2} \frac{d}{dx} \left(x^2 \frac{d}{dx} \right)$ for the Lane-Emden equation. It is also notable that u is a scalar function of real variable x in Eq. (1.2). For a system of differential equations, u will be a vector-valued function. However, in this paper, our studies are restricted to single ODE where u is a scalar-valued function. The principle step of the decomposition method is to suppose a series solution defined by

$$u = \sum_{i=0}^{\infty} u_i, \tag{1.3}$$

and then the ADM scheme corresponding to the functional equation (1.2) converges rapidly to $u \in H$ which is the unique solution to the functional equation [6, 25]. Equation (1.3) decomposes the nonlinear term Nu into an infinite series

$$Nu = \sum_{i=0}^{\infty} A_i, \tag{1.4}$$

where A_i are the so-called Adomian polynomials which depend on the solution components $u_0, u_1, \ldots, u_i$. For a given nonlinear functional $Nu = F(u)$ ($F(u)$ is assumed to be an analytic function of variable u in Hilbert space H), the Adomian polynomials are determined by the following definitional formula introduced by G. Adomian [1, 2, 3, 26]:

$$A_M = \frac{1}{M!} \frac{d^M}{d\lambda^M} F \left(\sum_{k=0}^{\infty} u_k \lambda^k \right) \Bigg|_{\lambda=0}, \quad M = 0, 1, 2, \ldots, \tag{1.5}$$

where the analytic parameter λ is simply a grouping parameter. An important property of Adomian polynomial A_M is that it depends by construction only on the solution components $(u_0, u_1, \ldots, u_M)$ and does not depend on higher-order solution components u_k with $k > M$ [27, 28]. Therefore, the higher-order terms for $k > M$ do not contribute in summation in Eq. (1.5).

Main step of ADM is to determine the Adomian polynomials of the nonlinear term Nu. Using the definitional formula (1.5) it is difficult to calculate higher-order Adomian terms due to the complexity in calculations of higher-order derivatives. Later, many authors have developed several convenient algorithms for fast generation

of the one-variable and the multi-variable Adomian polynomials. Adomian and Rach [29] produced a recurrence rule that provides a systematic computational procedure to determine Adomian polynomials. Later, Rach in his paper [30] established simple symmetry rules (which is called Rach's rule) in Adomian and Rach's algorithms, by which Adomian's polynomials can be determined quickly to higher orders. Using the algorithm presented by Wazwaz in [27], we need to collect the terms from the expansion, which takes a large computational time for higher orders. Applying the algorithm in [31], we require to compute the derivative after substitution in a recurrence relation between the Adomian polynomials. Recently in [32], the authors modified the formula (1.5) to determine the Adomian polynomials for nonlinear polynomial functionals. In [21, 33], Duan has developed more efficient and fast recurrence algorithms for the rapid generation of the Adomian polynomials for one-variable (which is the one-dimensional case in our studies) and multi-variable cases. Duan's Corollary 1 algorithm [21] (called index recurrence algorithm) and Duan's Corollary 3 algorithm [33] do not involve the differentiation operator in determining the reduced polynomials in one dimension. We only require the operations of addition and multiplication, which make these algorithms faster and more efficient techniques.

In this work, we have presented a new algorithm for fastest computations of Adomian polynomials for scalar-valued nonlinear polynomial functional (with index as positive integers) in a Hilbert space H with the help of matrix formulations rather than recurrence processes. Our proposed algorithm does not require complex mathematical operations such as parametrization, expansion, regrouping, and differentiation. In this algorithm, the higher-order Adomian polynomials can be determined through few matrix operations, making it faster and more efficient than the other existing algorithms in the literature. We have generalized the new algorithm in two dimensions where the solution u depends on two-state variables such as t, x.

The paper is organized as follows: In Sec. 2 we present our algorithm to determine Adomian polynomials for nonlinear polynomial functional. In Sec. 3, we apply our algorithms to the polynomial functions, and the computation times are compared with some other popular algorithms previously reported in the literature. In Sec. 4, we discuss our results and make some conclusions on our works. We list the MATHEMATICA code for the new algorithms in Listing 3 for one-dimensional case and in Listing 6 for two-dimensional case in Appendix: A, B respectively. We have also listed the MATHEMATICA code for some other algorithms which are Duan's Corollary 1 algorithm [21] and Duan's Corollary 3 algorithm [33, 34] with the one-dimensional case in Listings 4, 5 in Appendix: A.

2 Description of Our Proposed Algorithm

In this section, we have described a new algorithm for calculating the Adomian polynomials. This algorithm is only applicable for scalar-valued nonlinear polynomial functional (with index as positive integers) in a Hilbert space H for the two-dimensional case. In order to increase the calculating efficiency in this algorithm, all the mathematical operations are performed in the matrix forms.

Let us now consider a nonlinear polynomial functional F depends on two different functions u and v in H. The functions u and v can be expanded into the following two-dimensional series

$$u = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} u_{ij} \quad \text{and} \quad v = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} v_{ij} . \tag{2.1}$$

To illustrate our algorithm, we take the nonlinearity F in the simple form

$$F = uv. \tag{2.2}$$

And this nonlinear function can be decomposed by a series

$$F = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} A_{ij}, \tag{2.3}$$

3

where A_{ij} are called Adomian polynomials of the components u_{ij}, v_{ij} ($i = 0, 1, \ldots, j = 0, 1, \ldots$). Now, we divide the algorithm into six main steps (labeled from Step-1 to Step-6), and to illustrate each step, we have used the nonlinear polynomial function (2.2).

Step-1 (Express the functions u and v in the matrix forms): In this step, the functions u and v are expressed in the matrix forms. For computations in computer, we truncate the infinite series (2.1) up to the finite terms $i = m, j = n$. We can increase the accuracy in our results by increasing the values of m, n as far as possible. The functions u, v in the Eq. (2.1) can be expressed by $(m + 1) \times (n + 1)$ matrices

$$U = \begin{pmatrix} u_{00} & u_{01} & \ldots & u_{0l} & \ldots & u_{0n} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ u_{k0} & u_{k1} & \ldots & u_{kl} & \ldots & u_{kn} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ u_{m0} & u_{m1} & \ldots & u_{ml} & \ldots & u_{mn} \end{pmatrix} \text{ and } V = \begin{pmatrix} v_{00} & v_{01} & \ldots & v_{0l} & \ldots & v_{0n} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ v_{k0} & v_{k1} & \ldots & v_{kl} & \ldots & v_{kn} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ v_{m0} & v_{m1} & \ldots & v_{ml} & \ldots & v_{mn} \end{pmatrix}. \quad (2.4)$$

Step-2 (Extracting the submatrices from the matrices U and V): The Adomian polynomials corresponding to any matrix elements (let the matrix elements u_{kl}, v_{kl} located at row $k + 1$, column $l + 1$) in Eq. (2.4), depend on the other matrix elements whose row number (r) and column number (c) are less than or equal to $k + 1$ and $l + 1$ respectively, but do not depend on the matrix elements located at $r > k + 1$ and $c > l + 1$. In order to calculate the Adomian polynomials for the elements u_{kl} and v_{kl} in U and V, we extract the submatrices formed by the elements with rows $r \leq k + 1$ and columns $c \leq l + 1$ of the matrices U and V in Eq. (2.4). These submatrices are given by

$$U[0, 1, \ldots, k; 0, 1, \ldots, l] = \begin{pmatrix} u_{00} & u_{01} & \ldots & u_{0l} \\ \vdots & \vdots & \ldots & \vdots \\ u_{k0} & u_{k1} & \ldots & u_{kl} \end{pmatrix} \text{ and}$$

$$V[0, 1, \ldots, k; 0, 1, \ldots, l] = \begin{pmatrix} v_{00} & v_{01} & \ldots & v_{0l} \\ \vdots & \vdots & \ldots & \vdots \\ v_{k0} & v_{k1} & \ldots & v_{kl} \end{pmatrix}. \quad (2.5)$$

Step-3 (Flipping the submatrix): In this step, all the matrix elements of any one of the submatrices in Eq. (2.5) are flipped horizontally and then vertically or vice versa. Here we perform the flipping operation on the submatrix $V[0, 1, \ldots, k; 0, 1, \ldots, l]$. The flipping operation along horizontal axis can be shown in the following way

$$\begin{pmatrix} v_{00} & v_{01} & \ldots & v_{0l} \\ \vdots & \vdots & \ldots & \vdots \\ v_{k0} & v_{k1} & \ldots & v_{kl} \end{pmatrix} \xrightarrow{\text{flipping horizontally}} \begin{pmatrix} v_{0l} & v_{0l-1} & \ldots & v_{00} \\ \vdots & \vdots & \ldots & \vdots \\ v_{kl} & v_{kl-1} & \ldots & v_{k0} \end{pmatrix} = V[0, 1, \ldots, k; l, l-1, \ldots, 0]. \quad (2.6)$$

Then, the flipping operation along vertical axis is performed on the above flipped submatrix, which can be shown as

$$\text{flipping vertically} \downarrow \begin{pmatrix} v_{0l} & v_{0l-1} & \ldots & v_{00} \\ \vdots & \vdots & \ldots & \vdots \\ v_{kl} & v_{kl-1} & \ldots & v_{k0} \end{pmatrix} \longrightarrow \begin{pmatrix} v_{kl} & v_{kl-1} & \ldots & v_{k0} \\ \vdots & \vdots & \ldots & \vdots \\ v_{0l} & v_{0l-1} & \ldots & v_{00} \end{pmatrix}$$
$$= V[k, k-1, \ldots, 0; l, l-1, \ldots, 0]. \quad (2.7)$$

Step-4 (Element-wise matrices multiplication): In the element-wise multiplication (also known as the Hadamard product), each element i, j in the two matrices are multiplied together. We perform the

element-wise multiplication between two matrices $U[0, 1, \ldots, k; 0, 1, \ldots, l]$ and $V[k, k-1, \ldots, 0; l, l-1, \ldots, 0]$, given by

$$U[0, 1, \ldots, k; 0, 1, \ldots, l] \circ V[k, k-1, \ldots, 0; l, l-1, \ldots, 0] = W[0, 1, \ldots, k; 0, 1, \ldots, l] \qquad (2.8)$$

and in the matrix notation the above equation can be expressed by

$$\begin{pmatrix} u_{00} & u_{01} & \ldots & u_{0l} \\ \vdots & \vdots & \ldots & \vdots \\ u_{k0} & u_{k1} & \ldots & u_{kl} \end{pmatrix} \circ \begin{pmatrix} v_{kl} & v_{kl-1} & \ldots & v_{k0} \\ \vdots & \vdots & \ldots & \vdots \\ v_{0l} & v_{0l-1} & \ldots & v_{00} \end{pmatrix} = \begin{pmatrix} u_{00}v_{kl} & u_{01}v_{kl-1} & \ldots & u_{0l}v_{k0} \\ \vdots & \vdots & \ldots & \vdots \\ u_{k0}v_{0l} & u_{k1}v_{0l-1} & \ldots & u_{kl}v_{00} \end{pmatrix}. \qquad (2.9)$$

Here the symbol $\circ$ denotes the element-wise multiplication between two matrices.

Step-5 (Summation over matrix elements): In this step, we take summation over all the elements of the matrix $W[0, 1, \ldots, k; 0, 1, \ldots, l]$ and this summation is

$$A_{kl} = \sum_{i=0}^{k} \sum_{j=0}^{l} W_{ij} = u_{00}v_{kl} + u_{01}v_{kl-1} + \ldots + u_{kl}v_{00}. \qquad (2.10)$$

Here A_{kl} is the Adomian polynomial for the two matrix elements u_{kl}, v_{kl}. In the Adomian polynomial A_{kl}, notably, the sum of the first index at subscripts of the components of u, v in each term in A_{kl} are same. Similarly, the sum of the second index of the components of u, v in each term in A_{kl} are also same (here for the first index, the sum is k and for the second index, the sum is l), which obey the important property of the Adomian polynomial given in [27].

Step-6 (Constructing Adomian matrix): Repeating the previous steps from Step-1 to Step-5, the Adomian polynomials corresponding to each matrices elements in Eq. (2.4) are determined. All the calculated Adomian polynomials are stored in a matrix and can be expressed by

$$A = \begin{pmatrix} A_{00} & A_{01} & \ldots & A_{0l} & \ldots & A_{0n} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ A_{k0} & A_{k1} & \ldots & A_{kl} & \ldots & A_{kn} \\ \vdots & \vdots & \ldots & \vdots & \ldots & \vdots \\ A_{m0} & A_{m1} & \ldots & A_{ml} & \ldots & A_{mn} \end{pmatrix}. \qquad (2.11)$$

We call the matrix A in (2.11) as Adomian matrix for the given polynomial nonlinearity (2.2).

We present the pseudo-code for the algorithms described in Step-1 to Step-6 in Listing 1 which compute the Adomian matrix of Eq. (2.2). Here, it is worthwhile to note how a few simple matrix operations in Step-1 to Step-6 generate the Adomian polynomials of Eq. (2.2). It is clear from Step-1 to Step-6 that only $4(m+1)(n+1) - (m+n+2)$ number of matrix operations ($2(m+1)(n+1) - (m+n+2)$ number of flippings, $(m+1)(n+1)$ number of element-wise matrices multiplications and $(m+1)(n+1)$ number of matrix summations) are required to compute the Adomian matrix of Eq. (2.2) with $i = m, j = n$ in Eq. (2.1). This simplicity in mathematical operations enhances the computing efficiency of this algorithm.

Listing 1: Computation of Adomian matrix A of Eq. (2.2) in pseudo-code.

```
1  input: Functions u and v of Eq. (2.2)
2  output: Adomian matrix A
3  function AdomianMatrix(u, v)
4     Express u in matrix form U:  U ← Matrix(∑_{i=0}^{m} ∑_{j=0}^{n} u_ij)
5     Express v in matrix form V:  V ← Matrix(∑_{i=0}^{m} ∑_{j=0}^{n} v_ij)
6     for k ← m to k ≥ 0 do
```

```
7        for l ← n to l ≥ 0 do
8            U[0, 1, ..., k; 0, 1, ..., l] ← the submatrix of U for the elements U_{kl}
9            V[0, 1, ..., k; 0, 1, ..., l] ← the submatrix of V for the elements V_{kl}
10           V[k, k − 1, ..., 0; l, l − 1, ..., 0] ← V[0, 1, ..., k; 0, 1, ..., l] are flipped horizontally and
                 then vertically
11           Element−wise multiplication: W[0, 1, ..., k; 0, 1, ..., l] ←
                 U[0, 1, ..., k; 0, 1, ..., l] ∘ V[k, k − 1, ..., 0; l, l − 1, ..., 0]
12           A_{kl} ← ∑_{i=0}^{k} ∑_{j=0}^{l} W_{ij}
13       end for
14   end for
15   return A
16 end function
```

Listing 2: Computation of Adomian matrix A of Eq. (2.12) in pseudo-code.

```
1  input: Functions u^{(1)}, u^{(2)}, u^{(3)}, ..., u^{(P−2)}, u^{(P−1)}, u^{(P)} of Eq. (2.12)
2  output: Adomian matrix A
3  function AdomianMatrix2(u^{(1)}, u^{(2)}, u^{(3)}, ..., u^{(P−2)}, u^{(P−1)}, u^{(P)})
4      Express u^{(1)}, u^{(2)}, u^{(3)}, ..., u^{(P−2)}, u^{(P−1)}, u^{(P)} in matrix forms: U^{(P)} ← Matrix(
             ∑_{i=0}^{m} ∑_{j=0}^{n} u_{ij}^{(P)})
5      A ← U^{(P)}
6      for k ← P to k ≥ 2 do
7          A ← AdomianMatrix(U^{(k−1)}, A)
8      end for
9      return A
10 end function
```

2.1 F in general form

Let us now consider the nonlinear polynomial functional F in the following general form

$$F = u^{(1)} u^{(2)} u^{(3)} \ldots u^{(P-2)} u^{(P-1)} u^{(P)}, \tag{2.12}$$

where F depends on P number of two-dimensional functions $u^{(1)}, u^{(2)}, u^{(3)}, \ldots, u^{(P)}$. For $P = 2$ and $u^{(1)} = u, u^{(2)} = v$, Eq. (2.12) is reduced to Eq. (2.2). The algorithms presented in the Step-1 to Step-6 also work for Eq. (2.12) in the following way. Let $U^{(1)}, U^{(2)}, U^{(3)}, \ldots, U^{(P)}$ are the matrix forms of the two-dimensional functions $u^{(1)}, u^{(2)}, u^{(3)}, \ldots, u^{(P)}$ respectively. In order to determine the Adomian matrix of Eq. (2.12), at first, we will start to determine the Adomian matrix for the first two matrices $U^{(1)}, U^{(2)}$ or for the last two matrices $U^{(P-1)}, U^{(P)}$ using the algorithms presented in the Step-1 to Step-6. Let $A^{(P-1)(P)}$ is the Adomian matrix of the last two matrices $U^{(P-1)}$ and $U^{(P)}$. Next, we determine the Adomian matrix of the two matrices $A^{(P-1)(P)}$ and the previous one $U^{(P-2)}$. This process is continued up to first matrices $U^{(1)}$. After completing this process, finally, we will get the Adomian matrix of F given in Eq. (2.12). We present this process in pseudo-code in Listing 2 which determines the Adomian matrix of Eq. (2.12).

Now, we consider the nonlinear polynomial functional F in the more general and complicated form (a sum raised to a power)

$$F = \left(u^{(1)} + u^{(2)} + u^{(3)} + \ldots + u^{(P-2)} + u^{(P-1)} + u^{(P)}\right)^{\mathcal{N}} \tag{2.13}$$

where the power index $\mathcal{N}$ is a positive integer number. In this case, at first, we expand Eq. (2.13) in sum of product terms. Then we can easily determine the Adomian matrix of each term of the expansion using the above

algorithms for Eq. (2.12). Finally, simply adding all the Adomian matrices of each term, we get the Adomian matrix of Eq. (2.13).

In a one-dimensional case, the series (2.1) have only one index (say i). Therefore, all the matrices are one dimension, and in this case, in Step-3, we have to perform only a horizontal flipping operation. Besides this, all the algorithms described from Step-1 to Step-6 are identical in a one-dimensional case. In the following, we call the new algorithm presented by us the Adomian matrix algorithm.

3   Software Implementation and Comparisons with Other Algorithms

We have implemented the algorithm described in Sec. 2 (called Adomian matrix algorithm) into MATHEMATICA code in Listings 3 (one-dimensional case), 6 (two-dimensional case) of Appendix: A, B respectively. These MATHEMATICA programs can determine one-dimensional (using Listing 3) and two-dimensional (using Listing 6) Adomian polynomials of the following polynomial functional

$$F = u^{\mathcal{N}}, \tag{3.1}$$

where the power index $\mathcal{N}$ is an positive integer number that represents the order of nonlinearity. To determine the Adomian polynomials of Eq. (3.1), we have to input the power index $\mathcal{N}$ and the order of the Adomian matrix in the function arguments (detailed descriptions of these function arguments are given in the Appendix) of the MATHEMATICA functions, and these functions print the Adomian polynomials in the output cell of the MATHEMATICA notebook.

MATHEMATICA codes for some other algorithms such as Duan's Corollary 1 algorithm [21], Duan's Corollary 3 algorithm [33] for one-dimensional case are also presented in Listings 4, 5 of Appendix: B. The MATHEMATICA programs in Listings 4 and Listings 5 are taken from Appendix: A.1 in [21] and from Appendix: A in [34] respectively. Here to make the programs more faster we have modified the programs (given in [21], [34]) which work only with the polynomial functional (3.1) and evaluate the differentiation of (3.1) using the factorial formula $\frac{d^i F}{du^i} = \frac{\mathcal{N}!}{(\mathcal{N}-i)!} u^{\mathcal{N}-i}$.

We have compared the Adomian matrix algorithm with other algorithms by employing the MATHEMATICA programs given in Listings 3, 4, 5, 6 and using the polynomial functional (3.1). In Table 1, we have shown the comparisons between the computing speeds (measured in seconds) of the Adomian matrix algorithm (3rd column) and two different other algorithms (4th and 5th columns) for the one-dimensional case using the MATHEMATICA programs given in Listings 3, 4, 5 in Appendix: A. We measure the computing times by MATHEMATICA 9.0 on the laptop with Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz and 8 GB RAM, using the MATHEMATICA command `Timing[]` with suppressing output (i.e., the results are retained in memory). Table 1 displays that the Adomian matrix algorithm is faster and more efficient than the other two algorithms: Duan's Corollary 1 algorithm [21] and Duan's Corollary 3 algorithms [33]. For example, we observe that in calculating the first 50 Adomian polynomials, the Adomian matrix algorithm is almost 10^4 times faster for $\mathcal{N} = 3$ and almost 10^3 times faster for $\mathcal{N} = 10$ in comparison to the other two algorithms. Moreover, in calculating the first 100 Adomian polynomials, the Adomian matrix algorithm spends the time $\sim 10^{-2}$ s, but, notably, the other two algorithms are unable to give results within an elapsed time of 600 s.

We have also checked the computation efficiency of the Adomian matrix algorithm in the two-dimensional cases using the MATHEMATICA code in Listing 6. For example, the Adomian polynomials of Eq. (3.1) in the order of 40×40 are generated within 2.6 s for $\mathcal{N} = 3$ and within 19.5 s for $\mathcal{N} = 10$.

Table 1. Comparisons of computing times (unit: seconds) of the Adomian matrix algorithm with some other algorithms using different values of $\mathcal{N}$ in (3.1) and the different number (n) of Adomian polynomials in one dimension. In some table cells, $\times$ symbols indicate the algorithm in the corresponding column is unable to compute Adomian polynomials after spending almost 600s

Nonlinearity ($\mathcal{N}$)	Number of Adomian polynomials (n)	Adomian matrix algorithm	Duan's Corollary 1 algorithm [21]	Duan's Corollary 3 algorithm [33]
3	10	0.00047	0.0020	0.0025
	30	0.002	0.83	0.76
	50	0.0047	62	46
	100	0.017	$\times$	$\times$
5	10	0.00078	0.0026	0.0025
	30	0.0039	0.87	0.68
	50	0.0092	62.5	46.4
	100	0.037	$\times$	$\times$
10	10	0.0033	0.0037	0.0029
	30	0.012	0.96	0.65
	50	0.026	62.7	46.7
	100	0.095	$\times$	$\times$

4 Conclusion

We have presented a new algorithm (called the Adomian matrix algorithm) to determine the Adomian polynomials for scalar-valued nonlinear polynomial functional (with index as positive integers) in a Hilbert space H. The computations in the Adomian matrix algorithm do not need complicated mathematical operations such as parametrization, expansion, regrouping, differentiation, and so on. It is clear from Step-1 to Step-6 in Sec. 2 that the Adomian polynomials are determined entirely by some simple matrix operations. Because of the simplicity in mathematical operations, the algorithm is more efficient for the fast generation of the Adomian polynomials. We have designed two MATHEMATICA programs (one-dimensional case in Listing 3 and two-dimensional case in Listing 6) based on the Adomian matrix algorithm, and compared its efficiency in computations for the one-dimensional cases with other two popular and powerful algorithms, which are Duan's Corollary 1 algorithm [21] and Duan's Corollary 3 algorithms [33]. We have observed that the computation efficiency of the Adomian matrix algorithm is better than the other two algorithms. For example, in calculating the first 50 Adomian polynomials in one dimension with the nonlinearity index N=3 in Eq. (3.1), the Adomian matrix algorithm is almost 10^4 times faster than the other two algorithms. For N=10, we are able to find the first 100 Adomian polynomials using this new algorithm in just 10^{-2} s, whereas for N=3 and n=100, the other two algorithms fail to produce any results until 600 s have passed. Therefore, we can conclude that the Adomian matrix algorithm can be used to determine a large number of Adomian polynomials of nonlinear polynomial functionals that make the solutions more accurate.

Competing Interests

Author has declared that no competing interests exist.

References

[1] Adomian G. Stochastic System. Academic Press, New York; 1983.

[2] Adomian G. Nonlinear Stochastic Operator Equations. Academic Press, Orlando; 1986.

[3] Adomian G. Nonlinear Stochastic Systems Theory and Applications to Physics. Kluwer Academic, Dordrecht; 1989.

[4] Rach R, Adomian G, Meyers RE. A modified decomposition. Comput. Math. Appl. 1992;23:17–23.

[5] Adomian G. Solving Frontier Problems of Physics: The Decomposition Method. Kluwer Academic, Dordrecht; 1994.

[6] Mavoungou T, Cherruault Y. Convergence of Adomian's Method and Applications to Non-linear Partial Differential Equations. Kybernetes. 1992;21:13-25.

[7] N. Ngarhasta, et al. New numerical study of Adomian method applied to a diffusion model. Kybernetes. 2002;31:61–75.

[8] Wazwaz AM. Exact solutions to nonlinear diffusion equations obtained by the decomposition method. Appl. Math. Comput. 2001;123:109–122.

[9] Wazwaz AM. Partial Differential Equations and Solitary Waves Theory. Higher Education. Beijing; 2009.

[10] Yee E. Application of the decomposition method to the solution of the reaction–convection–diffusion equation. Appl. Math. Comput. 1993;56:1–27.

[11] Biazar J. Solution of systems of integral-differential equations by Adomian decomposition method. Appl. Math. Comput. 2005;168:1232–1238.

[12] Duan JS, An JY, Xu MY. Solution of system of fractional differential equations by Adomian decomposition method. Appl. Math. J. Chin. Univ. B 2007;22:7–12.

[13] Cherruault Y. Convergence of Adomian's method. Kybernetes. 1989;18:31–38.

[14] Abbaoui K, Cherruault Y. Convergence of Adomian's method applied to differential equations. Comput. Math. Appl. 1994;28:103–109.

[15] Abbaoui K, Cherruault Y. New ideas for proving convergence of decomposition methods. Comput. Math. Appl. 1995;29:103–108.

[16] Haq F, et al. Numerical Solution of Fractional Order Epidemic Model of a Vector Born Disease by Laplace Adomian Decomposition Method. Punjab University Journal of Mathematics. 2017;49:13.

[17] Ali A, Humaira, Laila, Shah K. Analytical solution of General Fisher's Equation by using Laplace Adomian decomposition method. J. Pure Appl. Math. 2018;2:01.

[18] Shah K, Bushnaq S. Numerical treatment of fractional endemic disease model via Laplace Adomian decomposition method. Journal of Science and Arts. 2017;39:257.

[19] Haq F, Shah K, Ur Rahman G, Shahzad M. Numerical solution of fractional order smoking model via Laplace Adomian decomposition method. Alexandria Engineering Journal. 2018;57:1061.

[20] Ali A, Shah K, Ali Khan R. Numerical treatment for traveling wave solutions of fractional Whitham-Broer-Kaup equations. Alexandria Engineering Journal. 2018;57:1991.

[21] Duan JS. An efficient algorithm for the multivariable Adomian polynomials. Appl. Math. Comput. 2010;217:2456–2467.

[22] Adomian G, Rach R, Shawagfeh NT. On the analytic solution of the Lane–Emden equation. Found. Phys. Lett. 1995;8:161–181.

[23] Wazwaz AM. A new method for solving singular initial value problems in the second order ordinary differential equations. Appl. Math. Comput. 2002;128:45–57.

[24] Wazwaz AM. The modified decomposition method for analytic treatment of differential equations. Appl. Math. Comput. 2006;173:165–176.

[25] Bougoffa L. An efficient method for solving nonlocal initial-boundary value problems for linear and nonlinear first-order hyperbolic partial differential equations. J. Appl. Math. Comput. 2013;43:31–54.

[26] Duan JS. New recurrence algorithms for the nonclassic Adomian polynomials. Computers and Mathematics with Applications. 2011;62:2961.

[27] Wazwaz AM. A new algorithm for calculating Adomian polynomials for nonlinear operators. Appl. Math. Comput. 2000;111:53–69.

[28] Azreg-Aï nou M. A developed new algorithm for evaluating Adomian polynomials. Comput. Model. Eng. Sci. 2009;42:1–18.

[29] Adomian G, Rach R. Inversion of nonlinear stochastic operators. J. Math. Anal. Appl. 1983;91:39–46.

[30] Rach R. A convenient computational form for the Adomian polynomials. J. Math. Anal. Appl. 1984;102:415–419.

[31] Biazar J, Shafiof SM. A Simple Algorithm for Calculating Adomian Polynomials. Int. J. Contemp. Math. Sciences. 2007;2:975-982.

[32] Agom EU, Ogunfiditimi FO. Modified Adomian Polynomial for Nonlinear Functional with Integer Exponent. IOSR-JM, 2015;11:40-45.

[33] Duan JS. Convenient analytic recurrence algorithms for the Adomian polynomials. Appl. Math. Comput. 2011;217:6337–6348.

[34] Duan JS, Rach R, Wazwaz AM. A reliable algorithm for positive solutions of nonlinear boundary value problems by the multistage Adomian decomposition method. Open Engineering. 2014;5:59–74.

Appendix A   Mathematica programs for one-dimensional case

The following three MATHEMATICA programs can determine one-dimensional Adomian polynomials of the nonlinear function (3.1). The function arguments N_ and n_ represent the nonlinear power index $\mathcal{N}$ in Eq. (3.1) and the number of first Adomian polynomials, respectively.

Listing 3: Program based on the Adomian matrix algorithm.

```
AdomMatAlgo1D[N_, n_] := Module[{h, j, k},
  u =.;
  mat = Table[Subscript[u, h], {h, 0, n - 1}];
  temmat = Table[Subscript[u, h], {h, 0, n - 1}];
  For[j = 1, j <= N - 1, j++,
    For[k = n, k >= 1, k--,
      mat[[k]] = Total[temmat[[;; k]]*Reverse[mat[[;; k]]]];
    ];
  ];
  mat
]
```

Listing 4: Program based on the Duan's Corollary 1 algorithm [21].

```
DuanIndexAlgoAdom[N_, n_] := Module[{Apoly, Zpoly, dirClt},
Subscript[Apoly, 0] = Subscript[u, 0]^N;
Zpoly = Table[0, {i, 1, n - 1}, {j, 1, i}];
Do[Zpoly[[suInd, 1]] = Subscript[u, suInd], {suInd, 1, n - 1}];
For[i = 2, i <= n - 1, i++,
  For[j = 2, j <= i, j++,
    Zpoly[[i, j]] = Expand[Subscript[u, 1]*Zpoly[[i - 1, j - 1]]];
    If[Head[Zpoly[[i, j]]] === Plus,
      Zpoly[[i, j]] = Map[#/Exponent[#, Subscript[u, 1]] &, Zpoly[[i, j]]],
      Zpoly[[i, j]] = Map[#/Exponent[#, Subscript[u, 1]] &, Zpoly[[i, j]],
        {0}]]];
  For[j = 2, j <= Floor[i/2], j++,
    Zpoly[[i, j]] = Zpoly[[i, j]] + (Zpoly[[i - j, j]] /.
    Subscript[u, sub_] -> Subscript[u, sub + 1])]];
dirClt = Table[Factorial[N]/Factorial[N - j]*(Subscript[u, 0]^(N - j)), {j,
    1, n - 1}];
Do[Subscript[Apoly, suInd] = Take[dirClt, suInd].Zpoly[[suInd]], {suInd, 1,
    n - 1}];
Table[Subscript[Apoly, suInd], {suInd, 0, n - 1}]]
```

Listing 5: Program based on the Duan's Corollary 3 algorithm [33, 34].

```
DuanCoro3AlgoAdm[N_, n_] := Module[{cPoly, i, k, j, derClt},
Table[cPoly[i, k], {i, 1, n - 1}, {k, 1, i}];
derClt = Table[Factorial[N]/Factorial[N - k]*(Subscript[u, 0]^(N - k)),
            {k, 1, n - 1}];
Apoly[0] = Subscript[u, 0]^N;
For[i = 1, i <= n - 1, i++,
```

```
  cPoly [ i ,  1] = Subscript [ u,  i ];
  For [ k = 2,  k <= i ,  k++,
     cPoly [ i ,  k] = Expand [1 / i *Sum [( j  + 1)*Subscript [ u,  j  + 1]*cPoly [ i  − 1 −
        j ,  k  − 1],
                        { j ,  0 , i  − k }]]];
  Apoly [ i ] = Take [ derClt ,  i ] . Table [ cPoly [ i ,  k] , {k,  1,  i }]];
Table [ Apoly [ i ] ,  { i ,  0,  n − 1}]]
```

Appendix B   Mathematica programs for two-dimensional case

The following MATHEMATICA program can determine two-dimensional Adomian polynomials of the nonlinear function (3.1). The function arguments N_, m_ and n_ represent the nonlinear power index $\mathcal{N}$ in Eq. (3.1), the number of rows and number of columns in the Adomian matrix, respectively.

Listing 6: Program based on the Adomian matrix algorithm.

```
AdomMatAlgo2D [ N_ ,  m_ ,  n_ ]  := Module [{ g,  h,  j ,  k,  l },
  u  =.;
  mat = Table [ Subscript [ u,  g,  h] ,  { g,  0,  m − 1} ,  {h,  0,  n − 1}];
  temmat = Table [ Subscript [ u,  g,  h] ,  { g,  0,  m − 1} ,  {h,  0,  n − 1}];
  For [ j  = 1,  j  <= N − 1,  j++,
    For [ k = m,  k >= 1,  k−−,
      For [ l  = n,  l  >= 1,  l −−,
        mat [[ k,  l ]] = Total [ temmat [[ ; ;  k,  ; ;  l ]] * Reverse [ Reverse [ mat [[ ; ;
           k,  ; ;  l ]] ,  1] ,  2] ,  2];
        ];
      ];
    ];
  mat
]
```