Future Generation Computer Systems 96 (2019) 101–110

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

An analytic computation-driven algorithm for Decentralized Multicore Systems

Yezhi Lin [a,b], Xinyuan Jin [c], Jiuqiang Chen [a,b], Ali Hassan Sodhro [e], Zhifang Pan [a,b,d,*]

[a] The First Affiliated Hospital of Wenzhou Medical University, Wenzhou 325000, PR China
[b] School of Information and Engineering, Wenzhou Medical University, Wenzhou 325035, PR China
[c] Oujiang College, Wenzhou University, Wenzhou 325035, PR China
[d] Information Technology Center, Wenzhou Medical University, Wenzhou 325035, PR China
[e] Computer and Information Science Department, Linkoping University, Linkoping 58183, Sweden

HIGHLIGHTS

- We present an innovative parallel algorithm to calculate approximations for initial or boundary value problems.
- A Python package AdomianPy is developed for initial or boundary value problems.
- Several examples are given to demonstrate the validity of our software package.
- It is a remarkable fact that the speed of executing our python package can be great improved by multi-core, even the super-linear speedup.

ARTICLE INFO

Article history:
Received 2 September 2018
Received in revised form 18 December 2018
Accepted 15 January 2019
Available online 4 February 2019

Keywords:
Parallel algorithm
Adomian–Rach double decomposition method
Adomian polynomials
Decentralized Multicore Systems

ABSTRACT

In the modern era, increasing numbers of cores per chip are applied for decentralized systems, but there is not any appropriate symbolic computation approach to construct multicore analytic approximation. Thus, it is essential to develop an efficient, simple and unified way for decentralized Adomian decomposition method to increase the potential speed of the multicore systems. In our paper, we present an innovative parallel algorithm of constructing analytic solutions for nonlinear differential system, which based on the Adomian–Rach double decomposition method and Rach's Adomian polynomials. Based on our algorithm, we further developed a user-friendly Python software package to construct analytic approximations of initial or boundary value problems. Finally, the scope of validity of our Python software package is illustrated by several different types of nonlinear examples. The obtained results demonstrate the effectiveness of our package by compared with exact solution and numeric method, the characteristics of each class of Adomian polynomials and the efficiency of parallel algorithm with multicore processors. We emphasis that the super-linear speedup may happens for the duration of constructing approximate solutions. So, it can be considered as a promising alternative algorithm of decentralized Adomian decomposition method for solving nonlinear problems in science and engineering.

1. Introduction

A large number of enigmas in engineering, biology, economics and other disciplines, e.g. flow and heat transfer problem, the simulations of the immune system, control and optimization theory, bound price problem, are often modeled using a system of nonlinear problems [1–5]. In particular, various kinds of decentralized systems, appeared in economics, medicine etc., are convenient and cost effective [6,7]. A wide range of analytic methods, like the Adomian decomposition method (ADM) [1,3], the perturbation-incremental method [8], the variational iteration method [9], the homotopy perturbation method [10], etc., is a reliable and efficient technique to handle such problems. It should be mentioned that the ADM is among the most simple and effective analytic methods to construct approximations of nonlinear differential equations, and have been modified and improved by Adomian and his collaborator, like the Adomian–Rach double decomposition [1,11], etc. These modifications, in most cases, undoubtedly have provided higher accuracy and faster convergence in nonlinear differential equations [1,12]. Furthermore, the ADM, which have been proved that it works efficiently for a large number of nonlinear problems including fractional differential equations [13], even stochastic system [1], is easy to be implemented by various programming languages, such as Maple [13], Mathematics [14], etc.

It is well known that symbolic computation, like ADM, consumes a large quantity of time and system resources in computing.

* Corresponding author.
E-mail address: panzhifang@wmu.edu.cn (Z. Pan).

https://doi.org/10.1016/j.future.2019.01.031
0167-739X/

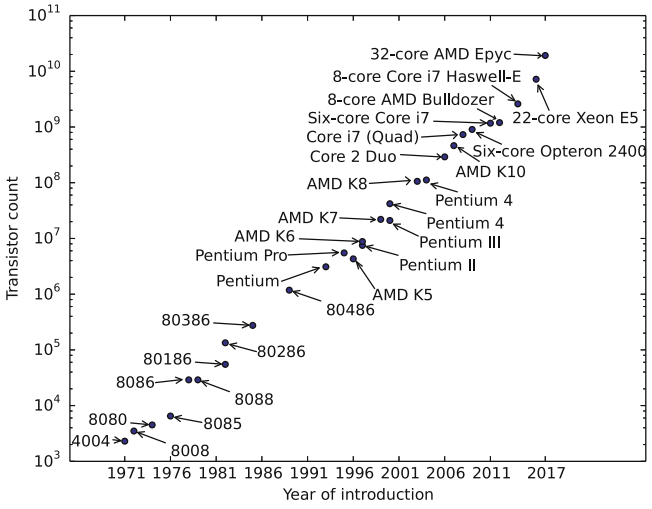

Fig. 1. Plot of transistor counts show that they double approximates every year follows Moore's law, and the datas are from the website Wikipedia.

In the past, we always sought the faster computation speed for obtaining the solution of the given problem from higher clock frequencies. Although it can seen from Fig. 1 that the number of transistors on integrated circuits becomes approximately twice every two year according to the Moore's law, heat dissipation has been a big problem for developing higher clock frequencies. Therefore, due to the balance among performance, power consumption and heat dissipation, chip designers have moved away from a focus on the high clock frequencies to multicore processors representing the future of computing. Nowadays researchers have no choice but to focus on parallel algorithms instead of traditional algorithms for accelerating the computational speed. More and more authors found the parallel algorithm can accelerate large-scale calculations [15]. Consequently, parallel computing for symbolic computation is widely utilized by engineers, experts and scientists on multicore computers [16].

With the rapid development of numbers of cores per chip, there is a vast literature on symbolic computation or decentralized system [17–20], but very little of relative literature focuses on our goal: A general parallel algorithm of ADM on multicore architecture. It is well known that the Adomian–Rach decomposition method, to a certain extent, can provide higher precision than numeric method. Inspired by the parallel algorithm, we present a new parallel algorithm for decentralized system. Furthermore, we develop the package AdomianPy based on Python, which is open source and powerful programming language, and demonstrate its effectiveness by comparing approximations with exact solutions in nonlinear examples. It should be mentioned that a parallel implementation by multi-core clusters is developed for the ADM. Moreover, our parallel algorithm on high performance computing architectures has potential to accelerate decentralized systems beyond real time, like large power system.

The paper has been organized as follows. In next section, we give the description of our innovative parallel algorithm to construct decentralized analytic approximations of nonlinear differential system. In Section 3, our Python software AdomianPy is outlined. Subsequently, we demonstrate the validity of AdomianPy by several typical nonlinear differential systems, and illustrate the effectiveness of AdomianPy by comparing their exact solutions. Sections 4 and 5 are several conclusions as well as our findings and future work.

2. Key parallel algorithms

A great variety of natural enigmas of science and engineering are modeled by nonlinear differential systems, especially the initial or boundary value problems. It is well known that analytic solutions are of fundamental significance for these initial or boundary value problems. The ADM is simple and effective analytic method for construction analytic approximations of initial or boundary value problems. In this section, based on the Adomian–Rach double decomposition method [1,11] and Rach's the Adomian polynomials [21], a parallel algorithm, for simplicity, is just described with the following single nonlinear differential equation

$$Lu + Ru + f(u) = g, \tag{1}$$

to construct analytic approximate solutions, but it is easily generalized to a system of nonlinear problems [13,22].

We recast Eq. (1) as

$$Lu = g - Ru - f(u). \tag{2}$$

For unifying the initial value problem and boundary value problem, we define the inverse operator L^{-1} of $L = d^n/dt^n$ as a pure two-fold integration. For convenience of explanation, we adopt the $L^{-1} = \int\int(\cdot)dxdx$, and apply on both sides of Eq. (2), and then obtain the equivalent equation

$$u = -L^{-1}f(u) - L^{-1}Ru + \Phi_x + L^{-1}g. \tag{3}$$

The general solution u is decomposed by the ADM as

$$u = \sum_{s=0}^{\infty} \lambda^s u_s. \tag{4}$$

Similarly, the analytic nonlinear term $f(u)$ will be decomposed into a parameterized series:

$$f(u) = \sum_{s=0}^{\infty} \lambda^s A_s, \tag{5}$$

where the grouping parameter λ, can be a function as $\lambda = \lambda(x)$, but always be setting the convenient form $\lambda = 1$ for computation. The Rach's Adomian polynomials A_m can be induced by determining the parameterized partial sums $\Theta_s = \left[\!\left[\sum_{i=0}^{\infty} A_i\right]\!\right]_{i\geq s}$ and

$$\Theta_s = \left[\!\left[\left[\!\left[\sum_{i=0}^{\infty}\left[\!\left[\frac{1}{i!}\left(\sum_{v=0}^{\infty} u_v - u_0\right)^i \times \frac{\partial^i}{\partial u_0^i}N(u_0)\right]\!\right]_{v\geq p_i^1(s)}\right]\!\right]_{n\geq p_i^2(s)}\right]\!\right]_{\sum jv_j\geq p_i^3(s)},$$

in which the double brackets indicate the truncation operators, and $p_i^j(s)$ represents their respective cut-off frequencies [21]. Therefore, we will choose the different values of $p_i^j(s)$ to define Class I, II, III and IV Adomian polynomials:

$$\begin{aligned} A_0^{(i)} &\equiv \Theta_1^{(i)}, \\ A_s^{(i)} &\equiv \Theta_{s+1}^{(i)} - \Theta_s^{(i)}, \quad s \geq 1. \end{aligned} \tag{6}$$

Here the Alg. 1 is given to calculate the Adomian polynomials, and the $\Theta_s^{(i)}$, $i =$ will be obtained by the following formula respectively. In addition, we also note that a collection of works about Adomian polynomials have demonstrated the convergence of them, and presented the proofs [1,21].

Algorithm 1 The algorithm of Adomian polynomials

Require: s, *type* represents the Class I, II, III and IV;
if type=1 **then**
  Calculate $\Theta_{s+1}^{(\mathrm{I})}$;
else if type=2 **then**
  Calculate $\Theta_{s+1}^{(\mathrm{II})}$;
else if type=3 **then**
  Calculate $\Theta_{s+1}^{(\mathrm{III})}$;
else if type=4 **then**
  Calculate $\Theta_{s+1}^{(\mathrm{IV})}$;
end if
Access the previous obtained data $\Theta_s^{(type)}$, if necessary;
if s=1 **then**
  $A_0^{(type)} = \Theta_1^{(type)}$,
else
  $A_s^{(type)} = \Theta_{s+1}^{(type)} - \Theta_s^{(type)}$,
end if
return $A_s^{(type)}$

2.1. Class I Adomian polynomials

When $p_I^1(s) = s, p_I^2(s) = s$ and $p_I^3(s) = \infty$, the partial sum can be induced by applying the truncation operator with the corresponding decomposition parameters as

$$\Theta_s^{(\mathrm{I})} = \sum_{k=0}^{\infty} \frac{1}{k!} \frac{\partial^{(k)} f(u_0)}{\partial \lambda^s} \left(\sum_{i=1}^{s-1} u_i \right)^k. \tag{7}$$

According Eq. (7), the Alg. 2 of Class I Adomian polynomials is presented. We remark that Class I Adomian polynomials [14,21] is available for some simple nonlinearities, like $f(u) = u^2$, and can be recasted as

$$A_s = f\left(\sum_{i=0}^{s} u_i \right) - f\left(\sum_{i=0}^{s-1} u_i \right). \tag{8}$$

Algorithm 2 Class I Adomian polynomials

Require: $f, s, u_i, i = 0, \cdots, s$;
$\Theta_s^{(\mathrm{I})} = N(\sum_{n=0}^{s-1} u_n)$.
return $\Theta_s^{(\mathrm{I})}$

2.2. Class II Adomian polynomials

When $p_{II}^1(s) = s, p_{II}^2(s) = \infty$ and $p_{II}^3(s) = \infty$, the partial sum can be induced by applying the truncation operator with the corresponding decomposition parameters as

$$\Theta_s^{(\mathrm{II})} = \sum_{i=0}^{s-1} \frac{1}{i!} \frac{\partial^{(i)} f(u_0)}{\partial \lambda^m} \left(\lambda \sum_{n=1}^{s-1} u_n \right)^i \Bigg|_{\lambda=1}. \tag{9}$$

We note that the Eq. (9) can be calculated by built-in Taylor expansions function in various symbolic computation packages or platforms, such as Taylor function in Maple, series function in Python package SymPy [23]. The Taylor expansions, most times, can help us to accelerate the computational speed and reduce the complexity of nonlinear problems. Here, we give the Alg. 3 to obtain the Θ_m.

From Eq. (6) and Alg. 3, we derive the Class II Adomian polynomials [21,24]. For the convenience of comparison, we list the

Algorithm 3 Class II Adomian polynomials

Require: $f, s, u_i, i = 0, \cdots, s$;
$\Theta_s^{(\mathrm{II})} = N(u_0 + \sum_{n=1}^{s-1} \lambda u_n)$.series(λ, 0, s).removeO()
$\Theta_s^{(\mathrm{II})} = \Theta_s$.subs(λ=1)
return $\Theta_s^{(\mathrm{II})}$

Adomian polynomials $A_i, i = 0, \ldots, 2$ as following.

$$A_0 = \Theta_1^{(\mathrm{II})} = f(u_0),$$

$$A_1 = \Theta_2^{(\mathrm{II})} - \Theta_1^{(\mathrm{II})} = \left(\sum_{i=1}^{2} u_i \right) f'(u_0),$$

$$A_2 = \Theta_3^{(\mathrm{II})} - \Theta_2^{(\mathrm{II})} = \left(\sum_{i=0}^{2} u_i \right)^2 \frac{f'(u_0)}{2!} + u_2 f'(u_0).$$

We remark that the Class II Adomian polynomials may be advantageous for computation on some nonlinear term.

2.3. Class III Adomian polynomials

When $p_{III}^1(s) = s - n + 1, p_{III}^2(s) = \infty$ and $p_{III}^3(s) = \infty$, the partial sum can be induced by applying the truncation operator with the corresponding decomposition parameters as

$$\Theta_s^{(\mathrm{III})} = \sum_{k=0}^{s-1} \frac{1}{k!} \frac{\partial^{(k)} f(u_0)}{\partial \lambda^s} \left(\lambda \sum_{n=1}^{s-k} u_n \right)^k \Bigg|_{\lambda=1}. \tag{10}$$

It is not easy that the Eq. (10) can be induced directly by Taylor expansion function. The Alg. 4 which we present as follows is a bit of complexity comparing with the Alg. 3.

Algorithm 4 Class III Adomian polynomials

Require: $f(u), s, u_i, i = 0, \cdots, m$
$\Theta_s^{(\mathrm{III})} = 0$
for $k = 1$ to s **do**
  temp = $f(u_0 + \sum_{n=1}^{s-k+1} \lambda u_n)$.series(λ, 0, k).removeO()
  $\Theta_s^{(\mathrm{III})} = \Theta_s^{(\mathrm{III})}$ + temp.coeff(λ, k-1)
end for
return $\Theta_s^{(\mathrm{III})}$

According to Eq. (6) and Alg. 3, we derive the Class III Adomian polynomials. For the convenience of comparison, Class III Adomian polynomials $A_i, i = 0, \ldots, 2$, are calculated as

$$A_0 = \Theta_1^{(\mathrm{III})} = f(u_0),$$

$$A_1 = \Theta_2^{(\mathrm{III})} - \Theta_1^{(\mathrm{III})} = u_1 f'(u_0),$$

$$A_2 = \Theta_3^{(\mathrm{III})} - \Theta_2^{(\mathrm{III})} = \frac{u_1^2}{2!} f''(u_0) + u_2 f'(u_0).$$

We note that the advantage of the Class III Adomian polynomials for computation, to be extent, is similar with Class II Adomian polynomials.

2.4. Class IV Adomian polynomials

Adomian and his collaborator extensively investigated the Class IV Adomian polynomials alias the classic Adomian polynomials [1, 14,21,25]. The algorithm of the classic Adomian polynomials can be given in the formula

$$\Theta_s^{(\mathrm{IV})} = \sum_{k=0}^{s-1} \frac{1}{k!} \frac{\partial^{(k)} f(u_0)}{\partial \lambda^s} \lambda^k \Bigg|_{\lambda=1}. \tag{11}$$

We note that the Eq. (11) can be directly calculated by series function in Python package Sympy. Therefore, we present the Alg. 5 to derive the Θ_m.

Algorithm 5 Class IV Adomian polynomials

Require: $f(u), s, u_i, i = 0, \cdots, s$;
1: $\Theta_s^{(\mathrm{IV})} = f(\sum_{n=0}^{s-1} \lambda^s u_n)$.series($\lambda$, 0, s).removeO()
2: $\Theta_s^{(\mathrm{IV})} = \Theta_s$.subs($\lambda$=1)
3: **return** $\Theta_s^{(\mathrm{IV})}$

Using Eqs. (6) and (11) , we derive the Class IV Adomian polynomials

$$A_s = \Theta_{s+1}^{(\mathrm{IV})} - \Theta_s^{(\mathrm{IV})} = \sum_{k=0}^{s} \frac{1}{k!} \frac{\partial^{(k)} f(\sum_{n=0}^{s} \lambda^s u_n)}{\partial \lambda^s}\bigg|_{\lambda=0}. \tag{12}$$

We list the Class IV Adomian polynomials $A_i, i = 0, \ldots, 2$, as follows:

$$A_0 = \Theta_1^{(\mathrm{IV})} = f(u_0),$$
$$A_1 = \Theta_2^{(\mathrm{IV})} - \Theta_1^{(\mathrm{IV})} = u_1 f'(u_0),$$
$$A_2 = \Theta_3^{(\mathrm{IV})} - \Theta_2^{(\mathrm{IV})} = u_2 f'(u_0) + \frac{u_1^2}{2!} f''(u_0).$$

We remark that because of deriving Class IV Adomian polynomials from the Taylor serious, the algorithm is suitable for programming. To sum up, the algorithms of the Class II, III and IV Adomian polynomials are easy to be programmed in any algebra system, such as Maple, Mathematics, etc., even Python. The strengths and weaknesses of four known class of Adomian polynomials will be further discussed by some example in next section.

By these algorithm of four classes Adomian polynomials, we obtain the special series for nonlinear term $f(u)$. Then we continue to decompose the Φ_x by Adomian–Rach decomposition method given as

$$\Phi_x = \sum_{s=0}^{\infty} \Phi_{x,s}. \tag{13}$$

Furthermore, we substitute Eqs. (4), (5) and (13) into (3), and get the following result

$$\sum_{s=0}^{\infty} \lambda^s u_s = L^{-1} g + \sum_{s=0}^{\infty} \Phi_{x,s} - L^{-1} R \sum_{s=0}^{\infty} \lambda^s u_s - L^{-1} \sum_{s=0}^{\infty} \lambda^s A_s. \tag{14}$$

Letting $\lambda = 1$ is simplified to Eq. (14). Therefore, we easily determine the matching coefficients $\Phi_{x,s} = c_{0,s} + x c_{1,s}$ from the initial or boundary conditions. For the initial conditions

$$g_{j+1} = u \to \frac{\partial^j u}{\partial x^j}\bigg|_{x=b_{j+1}} - p_{j+1}, \quad j = 0, 1, \tag{15}$$

or the exact Dirichlet boundary conditions

$$g_j = u \to u|_{x=b_j} - p_j, \quad j = 1, 2, \tag{16}$$

upon substitution Eq. (15) or (16) and solving for the matching coefficients $c_{0,s}$ and $c_{1,s}$, we have

$$g_j(\xi_1) = 0, \quad j = 1, 2,$$
$$g_j(\xi_{s+2}) = 0, \quad j = 1, 2.$$

According to the initial or boundary formulae, the Alg. 6 is given to unify the initial conditions and boundary conditions.

Algorithm 6 Handling initial or boundary conditions

Require: *IBCs* (the initial or boundary conditions), *func* (the dependent variables);
1: $boundary = dict()$
2: **for** *ics* in *IBCs* **do**
3: **if** Shaped like $u|_{x=b} = p$ **then**
4: We convert the mathematical expression $u|_{x=b} = p$ to the Python function $u- > sub(u, x, b) - p$
5: **else if** Shaped like $\frac{\partial^j u}{\partial x^j}\Big|_{x=b} = p$ **then**
6: We convert the mathematical expression $\frac{\partial^j u}{\partial x^j}\Big|_{x=b} = p$ to the Python function $u- > Derivative(u, \underbrace{x, x, \cdots, x}_{j}).subs(x, b) - p$
7: **end if**
8: **end for**
9: Inserts the specified Python function in the boundary collection
10: **return** *boundary*

The components u_s are given from the our modified recursion scheme

$$u_0 = c_{0,0} + x c_{1,0} + L^{-1} g,$$
$$u_{s+1} = c_{0,s+1} + x c_{1,s+1} - L^{-1} R u_s - L^{-1} A_s. \tag{17}$$

The inverse operator L^{-1} usually, but not always, cost too much time and computer resource to obtain the solution u_{s+1}. For L^{-1} is linear, that is to say, $L^{-1}(Ru_s) + L^{-1}(A_s) = L^{-1}(Ru_s + A_s)$. Moreover, $Ru_s + A_s$ always can be expressed as $\sum_{i=1}^{M} a_i f_i(x)$. Therefore, we have

$$L^{-1}(Ru_s) + L^{-1}(A_s) = L^{-1}\left(\sum_{i=1}^{M} a_i f_i(x)\right) = \sum_{i=1}^{M} L^{-1}\left(a_i f_i(x)\right). \tag{18}$$

In this situation, as the computation usually costs much CPU resource, the Alg. 7 with Python package multiprocessing is applied to accelerate the computation speed. In addition, as reducing the time overhead to call processes, we used one process in our algorithm to do the computation for small-scale expressions.

Algorithm 7 The parallel algorithm of the inverse operator

Require: $coreNumber, L^{-1}, Ru_s + A_s$
1: **if** For large-scale expressions **then**
2: p = Pool(coreNumber)
3: $Ru_s + A_s = \sum_{i=1}^{M} a_i f_i(x)$
4: temp = $[a_1 f_2(x), a_2 f_2(x), \cdots, a_M f_M(x)]$
5: p.map(L^{-1}, temp)
6: p.close
7: **else**
8: $L^{-1}(Ru_s + A_s)$
9: **end if**

According to our modified recursion scheme described as the Alg. 8, we easily determine the analytic approximate solution

$$\xi_r = \sum_{s=0}^{r-1} u_s(x). \tag{19}$$

Algorithm 8 The succinct description of the algorithm for Adomian decomposition method.

Require: *boundary*, n (the necessary highest order of approximations)

1: **for** *eq_no*, L' in *enumerate*(L) **do**
2: $u_0^{eq\_no} = \Phi_{x,s}^{eq\_no} + L'^{-1} g_{eq\_no}$
3: **end for**
4: **for** i in *range*(n) **do**
5: According the previous stated each class of Adomian polynomials, the Adomian polynomials $A_j^i, j = 1, \cdots, k$ (the number of equations), are obtained.
6: **for** *eq_no*, L' in *enumerate*(L) **do**
7: We obtain the $u_i^{eq\_no}$ by the formula $u_i^{eq\_no} = -L'^{-1} R_{eq\_no} - L'^{-1} A_{eq\_no}^i$, and also apply the parallel algorithm for it to accelerate the computation speed.
8: Solve for the $\Phi_x^{eq\_no}$ from initial conditions or boundary conditions
9: $u_i^{eq\_no} = u_i^{eq\_no} + \Phi_x^{eq\_no}$
10: **end for**
11: **end for**
12: For each of differential equations, we can obtain the analytic approximation $\xi_r = \sum_{s=0}^{r-1} u_s(x)$
13: **return** ξ

3. The application of the package AdomianPy

This section will give the application of the package AdomianPy, which is the implementation of our parallel algorithm described in the above section. We will give some examples to illustrate the effectiveness of our package AdomianPy, and demonstrate the score of that including a system of ordinal or partial differential equation, initial value problems and boundary value problems. what is more, by comparing other numeric method, symbolic computation can provide higher precision. Finally, the interface and usage of AdomianPy are described in Appendix.

Example 1. Consider the single nonlinear partial differential equation [14]

$$\frac{\partial \phi}{\partial t} - \phi \frac{\partial \phi}{\partial x} = 0, \quad \phi(x, 0) = x, \tag{20}$$

with the exact solution

$$\phi^*(t, x) = \frac{x}{1 - t}. \tag{21}$$

Our package plots the curves of 7-term approximations $\xi_7^{(j)}, j =$ I, II, III and IV which derived from the four classes Adomian polynomials respectively at $x = 1$ in Fig. 2. They have the same characteristics at x equals other values.

As we can see from Fig. 2, the curves of all 7-term truncated series and its exact solution agree well at $-1 < t < 1$, and the 7-term approximation $\xi_7^{(I)}$ has the largest effective region. It mean that our package AdomianPy is valid to construct analytic approximate solutions for nonlinear problems. In addition, we remark that the interval of convergence $\Omega^{(IV)} < \Omega^{(III)} < \Omega^{(II)} < \Omega^{(I)}$ for four known class of Adomian polynomials, and also observe that the computation time of selecting the Class IV Adomian polynomials is the least than others.

Example 2. Consider the single pendulum equation without utilizing linearization [14]

$$\frac{d^2\omega}{dt^2} + \sin(\omega) = 0, \quad \omega'(0) = 1, \omega(0) = 0, \tag{22}$$

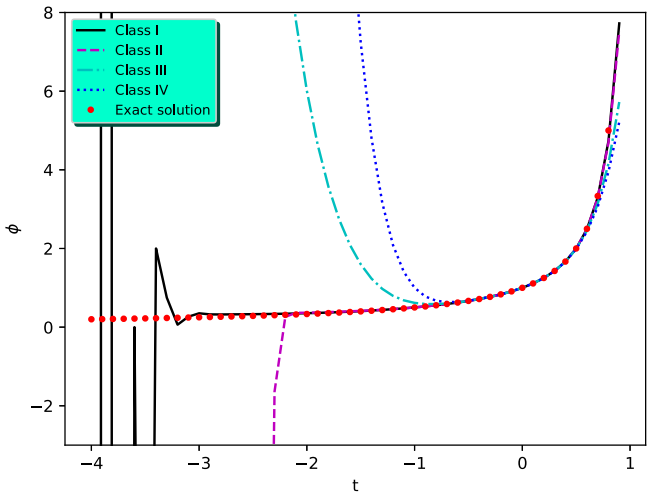

Fig. 2. The 4th-order truncated series $\xi_7^{(I)}$ (black solid line), $\xi_7^{(II)}$ (magenta dashed line), $\xi_7^{(III)}$ (cyan dash–dot line), $\xi_7^{(IV)}$ (blue dotted line), and the exact solution $z^*(x, y)$ (red point marker).

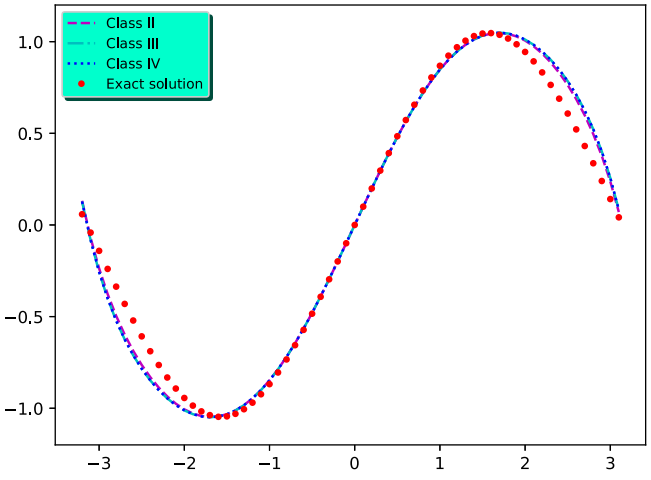

Fig. 3. The 4-term approximation $\xi_4^{(II)}$ (magenta dashed line), $\xi_4^{(III)}$ (cyan dash–dot line), $\xi_4^{(IV)}$ (blue dotted line), and the unique solution $x^*(t)$ (red point marker).

which admits the unique solution

$$\omega^*(t) = 2 \arcsin\left(\frac{1}{2} \mathrm{sn}\left(t, \frac{1}{4}\right)\right). \tag{23}$$

To show the validity of our algorithm, it takes our program about a few seconds to deliver ($n = 4$) Fig. 3, and also can be seen that the curves of all 4*th*-order truncated series and exact solution agree very well on the interval $[-1, 1]$ as shown in Fig. 3.

In addition, to show the improvement of multi-core, we will compare the elapsed times, which are consuming by calculating the 9-term approximate solutions $\xi_9^{(IV)}$ with multiprocessing technology, are recorded by running PyPy compiled Python program in Arch Linux using a laptop with a 4-core CPU processor (I7-4700MQ) as shown in Fig. 4. It can be seen that the speed of execution our program has great been improved within 4 processes by using Python Package Multiprocessing from Fig. 4.

Example 3. Consider a linear singular differetial system with initial conditions [26]

$$\begin{aligned} z_1' &= -1002 z_1 + 1000 z_2^2, \quad z_1(0) = 1, \\ z_2' &= z_1 - z_2 - z_2^2, \quad z_2(0) = 1. \end{aligned} \tag{24}$$

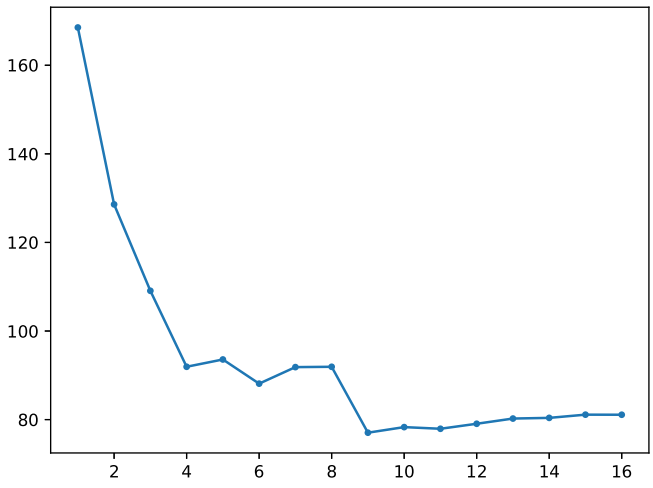

Fig. 4. The relation between elapsed times (seconds) and the number of processing.

For chasing higher accuracy, we compute 200-th truncated series by selecting Class IV Adomian polynomials, then output the according figure comparing the following exact solution

$$z_1^*(t) = e^{-2t}, \\ z_2^*(t) = e^{-t}, \tag{25}$$

to show the validity of our Python package AdomianPy, as display in Fig. 5. We note that the interval of convergence can be enlarge by computing higher order truncated series for this example.

Example 4. Consider a multi-order and product nonlinear equation with multi-point boundary condition [27]

$$x^{(4)}(t) + x(t)x'(t) - 4t^7 - 24 = 0, \tag{26}$$

with four-different-point boundary conditions,

$$x^{(3)}\left(\frac{1}{4}\right) = 6, \quad x''\left(\frac{1}{2}\right) = 3, \quad x(1) = 1, \quad x(0) = 0. \tag{27}$$

Its exact solution is

$$x^*(t) = t^4. \tag{28}$$

By selecting $type = 2, 3, 4$ respectively, the graphs of the $7th$-order approximations and its exact solution can be outputted (by $n = 7$) by AdomianPy as display in Fig. 6. We remark that the

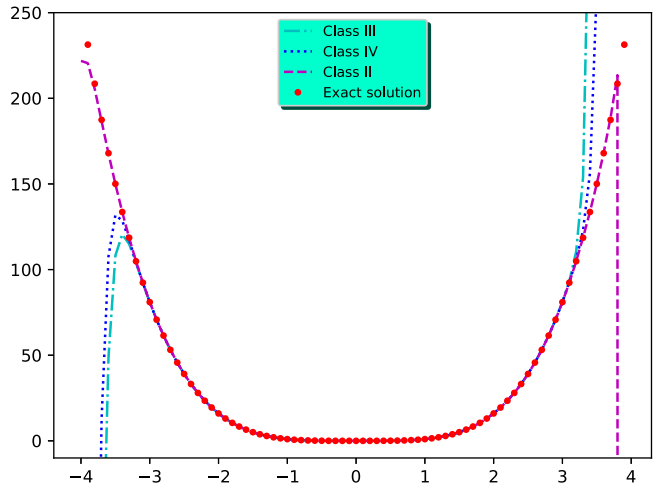

Fig. 6. The 7-term approximate solutions $\xi_7^{(II)}$ (magenta dashed line), $\xi_7^{(III)}$ (cyan dash–dot line), $\xi_7^{(IV)}$ (blue dotted line), and the exact solution $x^*(t)$ (red point marker).

curves of the 7-term truncated series $\xi_7^{(II)}$, $\xi_7^{(III)}$ and $\xi_7^{(IV)}$ as well as the exact solution agree very well because of error of solution is less than 10^{-7} from Fig. 7. It mean that our package is effective to construct approximations of nonlinear equation. We observe that the approximate solution $\xi_7^{(II)}$ converges in a larger convergence region, but consumes more time than the approximate solution $\xi_7^{(III)}$ and $\xi_7^{(IV)}$.

In addition, to show the validity of our parallel algorithm, we will compare the elapsed times, which are consuming by calculating the 9-term approximate solutions $\xi_9^{(IV)}$ with multiprocessing technology, are recorded by running PyPy compiled Python program in Arch Linux using a laptop with a 4-core CPU processor (I7-4700MQ) as display in Fig. 8. We lay emphasis on the superlinear speedup as we use 2 processes to compute the approximate solutions, and also note the consuming time to calculate the approximate solutions may grows as more than 4 processes is used to run our program.

Example 5. Consider a problem of deflections of an elastic loaded string with Dirichlet boundary conditions [28]

$$x'' = -\left(1 + a^2(x')^2\right), \quad x(0) = 0, \quad x(1) = 0, \quad t \in [0, 1]. \tag{29}$$

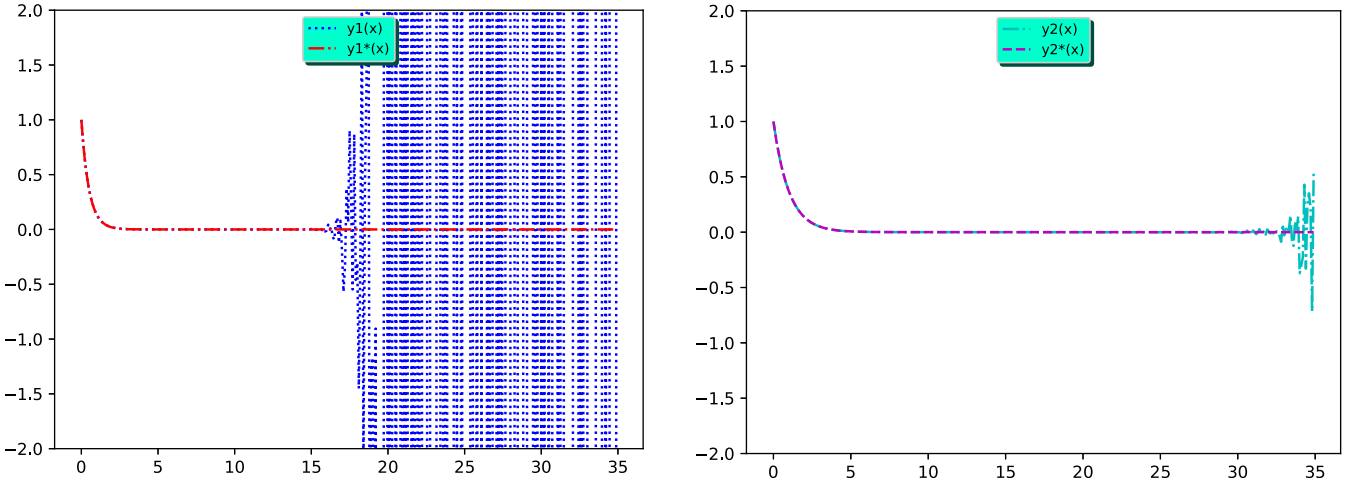

Fig. 5. The approximate solutions of Eq. (24) and the exact solution of Eq. (25) for $0 \le t \le 35$ and $-2 \le z_1, z_2 \le 2$.

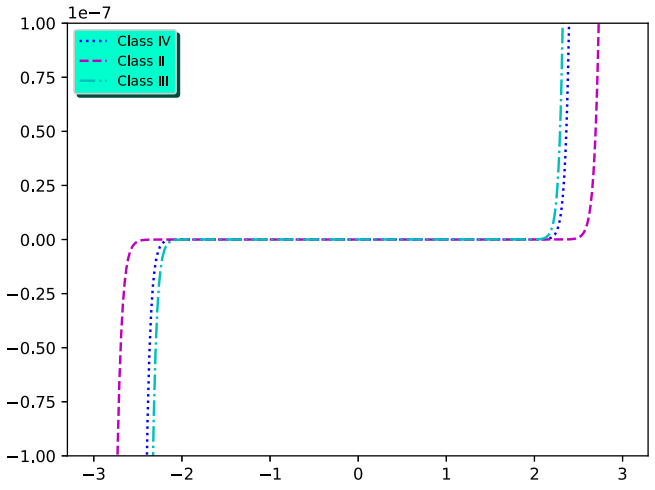

Fig. 7. The error $\xi_7^{(\mathrm{II})} - x^*(t)$ (magenta dashed line), $\xi_7^{(\mathrm{III})} - x^*(t)$ (cyan dash–dot line), $\xi_7^{(\mathrm{IV})} - x^*(t)$ (blue dotted line).

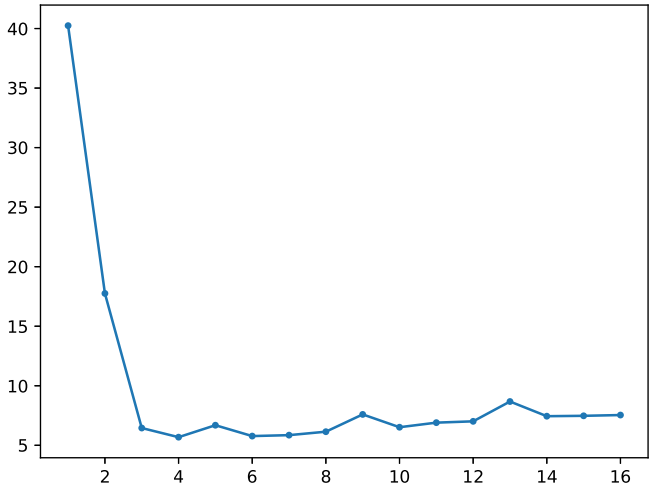

Fig. 8. The relation between elapsed times (seconds) and the number of processing.

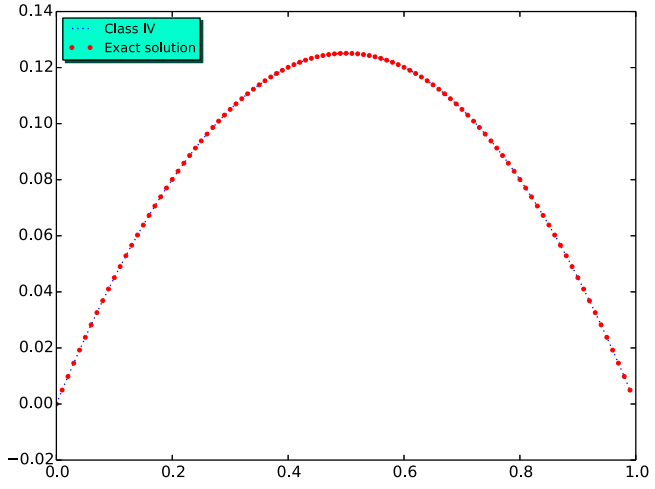

Fig. 9. The 11-term approximate solutions $\xi_{11}^{(\mathrm{IV})}$ (blue dotted line), and the exact solution $x^*(t)$ (red point marker).

For physical parameter $a = 1/7$, Cuomo and Marasco proposed a numerical approach to obtain the solution of Eq. (29) in [28]. By using our package AdomianPy for the Class IV Adomian polynomials, we experimentally calculate the truncated solution ξ_{11} comparing exact solution

$$x^*(t) = \frac{1}{a^2} \ln\left(\frac{\cos a(t - \frac{1}{2})}{\cos \frac{a}{2}}\right), \tag{30}$$

as shown in Fig. 9, and the absolute error comparing with that of Cuomo's numerical approach by using the finite difference method as shown in Fig. 10. It can be seen from Fig. 10 that symbolic method can obtain higher precision than numeric method for nonlinear problem. It can be obtain higher precision if someone overcome the limit of float point types in Python platforms which is approximately 15 decimal digits.

Example 6. Consider a 12th-order ordinary differential equation [13]

$$x^{(12)}(t) = -132\sin(t) - 23t\cos(t) + x(t), \tag{31}$$

with two-point boundary conditions

$$x^{(k)}(-1) = a_k, \quad x^{(k)}(1) = b_k, \quad k = 0, \ldots, 5, \tag{32}$$

where

$$\begin{aligned} a_0 &= b_0 = 0, \\ a_1 &= b_1 = 2\sin(1), \\ a_2 &= -b_2 = 4\cos(1) + 2\sin(1), \\ a_3 &= b_3 = 6\cos(1) - 6\sin(1), \\ a_4 &= -b_4 = -8\cos(1) - 12\sin(1), \\ a_5 &= b_5 = -20\cos(1) + 10\sin(1). \end{aligned} \tag{33}$$

For the 12th-order boundary value problem, by selecting the Class IV Adomian Polynomials, we calculate the truncated series solution which can be displayed as shown in Fig. 11. As shown in Fig. 11, the curves of the 10-term approximate solution $\xi_{10}^{(\mathrm{IV})}$ and its exact solution agrees very well to show the effectiveness of our package AdomianPy.

In short, the effectiveness of our package AdomianPy are demonstrated by these examples. Applying Python package AdomianPy, one can solve initial value problems by deriving the analytic approximate solutions automatically, but also boundary value problems. It is a remarkable fact that the speed of executing our python package can be great improved by multi-core, even the super-linear speedup such as Example 4. Besides, from these example, we know that selecting different Class of Adomian polynomials may be important for given nonlinear problems, and the characteristic of four known Class of Adomian polynomials will be discussed in next section.

4. Discussions

In this paper, combined the Adomian–Rach double decomposition method with the Rach's Adomian polynomials, a parallel algorithm is given to construct truncated series for initial or boundary value problems. Some discussions about the four known class of Adomian polynomials and speedup are given as following. On the one hand, the class I Adomian polynomials always can be selected for simple nonlinearities, such as Example 1. Furthermore, due to the differences in collecting term u_s, the Class IV Adomian polynomials has the fastest relative rate of convergence in four known Class of Adomian polynomials [21]. Therefore, we compute highest order of approximations of nonlinear problems by selecting Class IV Adomian polynomials, such as Example 3. It always consumes the least computational time among four known Class of Adomian Polynomials that we obtain the truncated series by selection Class

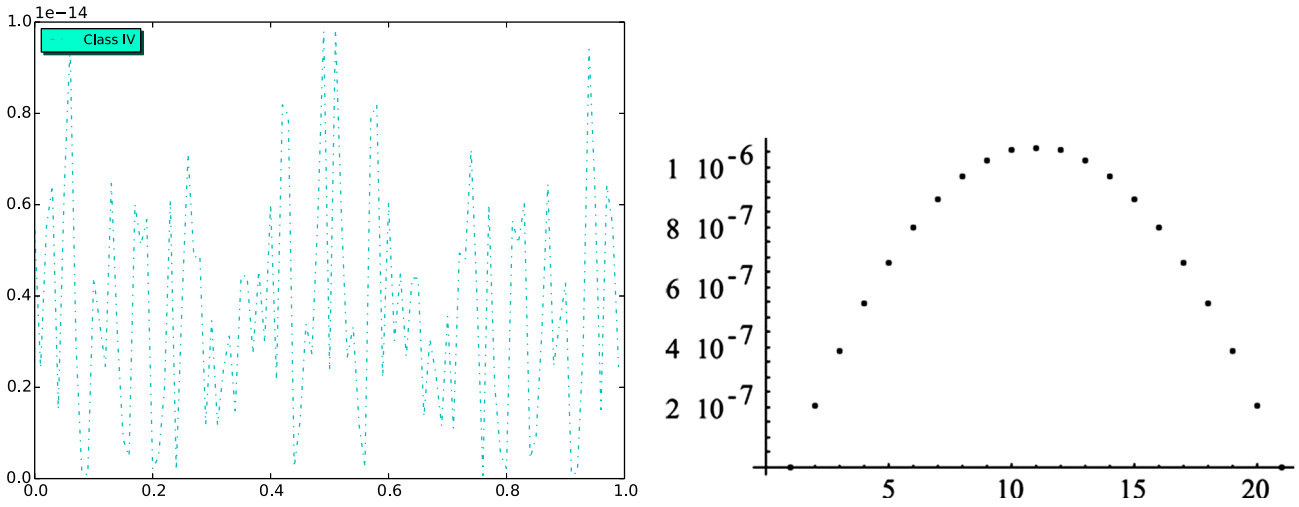

Fig. 10. The absolute error of Adomian–Rach decomposition method (left) and the finite difference method (right).

IV Adomian polynomials. Moreover, we have observed the approximations derived by selecting the Class II Adomian polynomials always have larger effective region than others. On the other hand, we observe that the computational speed, to a large extend, was improved when the number of processing is less than 4 in our laptop from Figs. 4 and 8. Especially, we obtain the super-linear speedup when the number of processing is used from 1 to 2 as show in Fig. 4.

5. Conclusion and future work

Based on our parallel algorithm, we developed the Python package AdomianPy to automatically derive analytic approximations. Some examples, such a system of ordinary or partial differential equation, initial value problems, and boundary value problems, has demonstrated its scope by applying various types of nonlinear problems, as well as the its effectiveness and speedup. We remark that the parallel algorithm, for most nonlinear problems, improve their computational speed. Therefore, our Python package AdomianPy provides an easy-to-extend tool to handle the decentralized system which is encountered frequently in science and engineering.

But our package currently does not work for more complex nonlinear problems including robin boundary value problems, fractional differential equations, and other decentralized system. Furthermore, the optimizing the multicore processors is not enough for some examples, especially QoS in parallel algorithm. In the future, we firstly will extend our package to solve more nonlinear problems, such as stochastic system. Secondly, We will improve the parallel algorithm to obtain less computational time by the multicore processors optimizes or GPU computing. Thirdly, it is a great significance that we can find the interval of convergence for most nonlinear problems.

Acknowledgments

This research was supported by Zhejiang Provincial Natural Science Foundation of China under Grant No. LQ15A010009, LY16F030010, supported by Wenzhou Science & Technology Bureau, China (Approval No. Y20150086 and 2018ZG016), and also supported by China Scholarship Council.

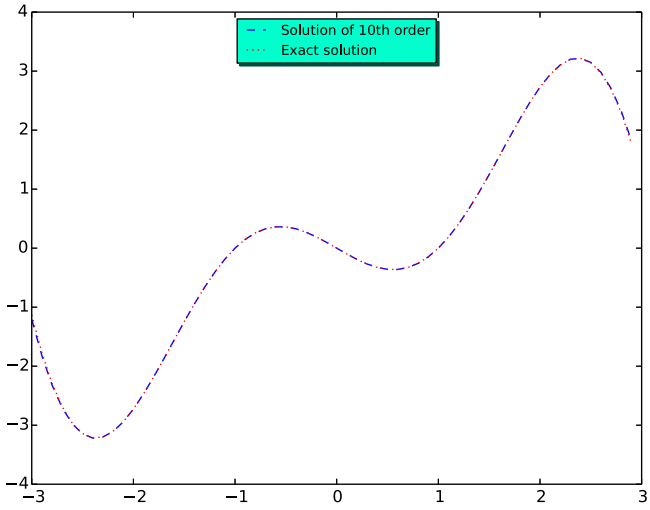

Fig. 11. The 10-term approximate solution $\xi_{10}^{(IV)}$ (blue dotted line), and its exact solution (red point marker).

Appendix. The interface and usage of our package AdomianPy

The main interface of Python package AdomianPy is *asolve*(*eq*, *ics*, *kwargs*), where the parameter *eq* can be any supported ordinary or partial differential equations, *ics* is the set of boundary conditions for the differential equations. The optional parameters *kwargs* includes $func = None, n = 5, type = 4$ and $core = 1$, where *func* is a function of variables whose derivatives in that variables make up the ordinary or partial differential equations, *n* represents the necessary highest order of approximations, $type = 1, 2, 3, 4$ represents the Class I, II, III, IV Adomian polynomials respectively, *core* is the number of processors used for computation, and their default value is *None*, 5, 4, 1 respectively.

For partial differential equation

$$\frac{\partial \phi}{\partial t} - \phi \frac{\partial \phi}{\partial x} = 0, \quad \phi(x, 0) = x, \tag{34}$$

we give the following Python program to calculate the truncated series solution with 2 cores by selecting $n = 7$ and $type = 4$.

```
import numpy as np
from sympy import *
import matplotlib.pyplot as plt
from adomian import asolve
```

```
def example():
  t, x = symbols('t x')
  phi = Function('phi')
  eq = Eq(phi(x,t).diff(t) -  phi(x,t)*phi(x,t).diff(x), 0)
  ics = {phi(x,0):x}
  asol = asolve(eq, ics, type= 4, n=7, core=2)

    t1 = np.arange(-4.0, 1.0, 0.1)
    x1 = []
    for asol_ in asol:
        f = Lambda(t, asol_.subs(x,1))
        temp = map(f, t1)
        temp = map(N, temp)
        x1.append(temp)
   f = Lambda(t,1/(1-t))
   temp = map(f, t1)
   temp = map(N, temp)
   ex_sol = temp
   fig, ax = plt.subplots()
   ax.set_ylim(-3, 8)
   ax.plot(t1, x4[0], 'b:', label='Class IV')
   plt.show()
def main():
    example()
```

References

[1] G. Adomian, Solving Frontier Problems of Physics: The Decomposition Method, Springer Science & Business Media, 2013.

[2] W. Wu, H. Zhang, S. Pirbhulal, S.C. Mukhopadhyay, Y.T. Zhang, Assessment of biofeedback training for emotion management through wearable textile physiological monitoring system, IEEE Sens. J. 15 (12) (2015) 7087–7095.

[3] Y.T. Yang, C.C. Chang, C.K. Chen, A double decomposition method for solving the annular hyperbolic profile fins with variable thermal conductivity, Heat Transf. Eng. 31 (2010) 1165–1172.

[4] G. Adomian, R. Rach, Coupled differential equations and coupled boundary conditions, J. Math. Anal. Appl. 112 (1985) 129–135.

[5] W. Wu, S. Pirbhulal, H. Zhang, S.C. Mukhopadhyay, Quantitative assessment for self-tracking of acute stress based on triangulation principle in wearable sensor system, IEEE J. Biomed. Health Inf. 1 (1) (2018) 1–10.

[6] Britta Klagge, Ron Martin, Decentralized versus centralized financial systems: is there a case for local capital markets?, J. Econ Geogr. 5 (4) (2005) 387–421.

[7] W. Wu, S. Pirbhulal, A.K. Sangaiah, S.C. Mukhopadhyay, G. Li, Optimization of signal quality over comfortability of textile electrodes for ECG monitoring in fog computing based medical applications, Future Gener. Comput. Syst. 18 (2018) 515–526.

[8] H.S.Y. Chan, K.W. Chung, Z. Xu, Stability and bifurcations of limit cycles by the perturbation-incremental method, J. Sound Vib. 206 (1997) 589–604.

[9] M. Abdou, On the variational iteration method, Phys. Lett. A 366 (2007) 61–68.

[10] S. Liao, Homotopy Analysis Method in Nonlinear Differential Equations, Springer, 2012.

[11] G. Adomian, R. Rach, A new algorithm for matching boundary conditions in decomposition solutions, Appl. Math. Comput. 57 (1993) 61–68.

[12] Y. Cherruault, G. Adomian, K. Abbaoui, R. Rach, Further remarks on convergence of decomposition method, Int. J. Bio-med. Comput. 38 (1) (1995) 89–93.

[13] Y. Lin, Y. Liu, Z. Li, Symbolic computation of analytic approximate solutions for nonlinear differential equations with boundary conditions, Appl. Math. Comput. 222 (2013) 145–166.

[14] J.S. Duan, New recurrence algorithms for the nonclassic Adomian polynomials, Comput. Math. Appl. 62 (2011) 2961–2977.

[15] Mathias Jacquelin, Lin Lin, Chao Yang, Pselinv—A distributed memory parallel algorithm for selected inversion: The symmetric case, ACM Trans. Math. Software 43 (3) (2017) 21.

[16] J.L. Awange, B. Paláncz, R.H. Lewis, L. Völgyesi, Parallel computations, in: Mathematical Geosciences, Springer, 2018.

[17] R.H. Halstead, T. Fujita, Masa: A multithreaded processor architecture for parallel symbolic computing, in: Proc. 15th Annu. Int. Symp. Comput. Architecture, 1988, pp. 443–451.

[18] S. Pirbhulal, H. Zhang, S.C. Mukhopadhyay, W. Wu, Y.T. Zhang, Heart-beats based biometric random binary sequences generation to secure wireless body sensor networks, IEEE Trans. Biomed. Eng. (2018) 1–9.

[19] S. Pirbhulal, H. Zhang, S.C. Mukhopadhyay, W. Wu, Y.T. Zhang, An efficient biometric-based algorithm using heart rate variability for securing body sensor networks, Sensors 15 (2015) 15067–15089.

[20] S. Pirbhulal, H. Zhang, M.E. Alahi, H. Ghayvat, W. Wu, S.C. Mukhopadhyay, Y.T. Zhang, A novel secure iot-based smart home automation system using a wireless sensor network, Sensors 17 (2016) 69.

[21] R. Rach, A new definition of the adomian polynomials, Kybernetes 37 (2008) 910–955.

[22] Y. Lin, Y. Liu, Z. Li, Symbolic computation of analytic approximate solutions for nonlinear differential equations with initial conditions, Comput. Phys. Comm. 183 (2012) 106–117.

[23] A. Meurer, C.P. Smith, M. Paprocki, O. Čertí k, S.B. Kirpichev, M. Rocklin, A. Kumar, et al., Sympy: Symbolic computing in python, PeerJ Comput. Sci. 3 (2017) e103.

[24] G. Adomian, R. Rach, Modified adomian polynomials, Math. Comput. Model 24 (1996) 39–46.

[25] A.M. Wazwaz, Partial Differential Equations and Solitary Waves Theory, Springer Science & Business Media, 2010.

[26] A.S. Mahmood, L. Casasús, W. Al-Hayani, The decomposition method for stiff systems of ordinary differential equations, Appl. Math. Comput. 167 (2005) 964–975.

[27] J. Ali, S. Islam, S. Islam, G. Zaman, The solution of multipoint boundary value problems by the optimal homotopy asymptotic method, Comput. Math. Appl. 59 (2010) 2000–2006.

[28] S. Cuomo, A. Marasco, A numerical approach to nonlinear two-point boundary value problems for ODEs, Comput. Math. Appl. 55 (2008) 2476–2489.

Yezhi Lin was born in Wenzhou, China in 1984. He received the B.S. in Information and computation science from the Wenzhou University, China, in 2006 and M.S. degrees in Applied Mathematics from the Wenzhou University, China, in 2010. He received the Ph.D. degree in East China Normal University, China, in 2013.

Since 2013, he has been a Lecturer with School of Information and Engineering, Wenzhou Medical University, China. His research interests include symbolic computation, data analysis, and dynamical system.

Xinyuan Jin currently works as a Lecturer of Oujiang College in Wenzhou university of Management Science. He has achieved the Master in management from The Zhejiang Normal University in 2015. His major interest is numerical calculation, Management index, risk management system.

Mr. Author's awards and honors include municipal 551 talents, 100 young people of social science, presided over 11 vertical projects.

Jiuqiang Chen was born in Wenzhou, China in 1983. He received the B.S. in E-commerce from the Lanzhou University, China, in 2006 and M.S. degrees in Computation science from the Lanzhou University, China, in 2010. He received the Ph.D. degree in Université Paris-Sudn, France, in 2013.

Since 2014, he has been a Lecturer with School of Information and Engineering, Wenzhou Medical University, China. His research interests include data analysis.

Ali Hassan Sodhro works as the Postdoctoral Research Fellow in Computer and Information Science Department, Linkoping University, Linkoping-58183, Sweden . He is reviewer of many peer-reviewed international journals, IEEE Sensor Journal, International Journal of Distributed Sensor Networks, Digital Communication and Networks-Elsevier, Wireless Personal Communication-Springer, etc. He is TPC member of ACM's 12th International Conference on BondyNets 2017. Dr Ali Sodhro completed his Ph.D. from Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences (SIAT, CAS) and University of Chinese Academy of Sciences (UCAS) China in Biomedical Electronic Systems on Energy-efficient Communication for Wireless Body Sensor Networks, 2016. He did his M.E (Communication Systems and Networks) and B.E (Telecommunication) both from Mehran University of Engineering and Technology, Jamshoro, Sindh, Pakistan in 2008 and 2010, respectively. He has a vast experience of 9 years in Academia & Research. He is a recipient of Full funded Merit Scholarship Award from Sindh Technical Board, Karachi, Pakistan, for B.Engg:, from Jan 2004 to Dec. 2007. Fully funded Merit Scholarship Award from Sindh Technical Board, Karachi and Illm-e-Foundation Lahore, Pakistan, for M.Engg:, from Jan 2008 to July 2009. Fully funded Ph.D. scholarship Award from University of Chinese Academy of Sciences and Shenzhen Institutes of Advanced Technology, Chinese Academy

of Sciences, China, from August 2011 to July 2016. He is also recipient of many research awards, i.e., Third Best Paper Award with Cash Prize o=&, Certificate at Guangzhou Annual Report Competition at Guangzhou cy=China p=2012,; Best Student Presentation Award with Cash Prize at 3rd Global Health Informatics Summit 2015 in Conjunction with The 11th IEEE-EMBS International Summer School & Symposium on MDBS'2015 & The 10th International School & Symposium on BHE'2015; outstanding Graduate Award, 2015 with Cash Prize and Certificate. He is Author of 35 Research Journal & International conference Proceeding Papers, 6 Book Chapters.

Zhifang Pan was born in Wenzhou, China in 1973. He received the Bachelor's degree in Physics from Wenzhou Normal College (now Wenzhou University), China, in July 1996. He received the Master's degree in Medicine from Wenzhou Medical College (now Wenzhou Medical University), China, in July 1999. He received the Ph.D. degree in the Department of Computer Science and Engineering from Shanghai Jiao Tong University in China. In August 1999, he joined Wenzhou Medical University, where he is currently an Associate Professor and a deputy director of information technology center. His research interests include analysis of medical image and biomedical data.