A documentation on GiNaCDE library to solve Differential Equations

GiNaCDE (V1.0.0)

Mithun Bairagi
Department of Physics, The University of Burdwan, Golapbag 713104, West Bengal, India

November 21, 2021

CONTENTS

1. INTRODUCTION

The GiNaCDE library written in C++ programming language is complete implementations of three different methods F-expansion [1–5], modified F-expansion (in short mF-expansion) [6], and first integral methods (in short FIM) [7–10]. GiNaCDE can outputs exact analytical traveling-wave solutions of nonlinear partial differential equations (NLPDEs) automatically. GiNaCDE for its algebraic manipulations entirely depends on GiNaC library [11].

2. PROBLEM

Let us consider an NLPDE with independent variables $t, x_1, x_2, \ldots x_m = \mathbf{X}$ and dependent variable u in the following general form

$$F\left(\alpha_i, u, u_t, u_{x_1}, u_{x_1} \ldots, u_{x_m}, u_{tt}, u_{tx_1}, u_{tx_2}, \ldots, u_{tx_m}, u_{x_1x_1}, u_{x_1x_2} \ldots, u_{x_1x_m} \ldots\right) = 0, \tag{1}$$

where $\alpha_i (i = 1, \ldots, l)$ are the parameters, $u = u(t, x_1, x_2, \ldots x_m)$ and F is a polynomial about u and its derivatives. Equation (1) does not explicitly depend on the independent variables $t, x_1, x_2, \ldots x_m$. The current package GiNaCDE with the all three algorithms of F-expansion, mF-expansion and FIM described below, can solve the NLPDEs of the form (1). However, there is no guarantee that the code always give the complete solutions of all NLPDEs of the form (1), and sometimes the code may failure to give solutions due to the complexity in the problems. Here, it should be noted that our algorithms are also applicable to Eq. (1) without the parameters α_i. Equation (1) is called input-NLPDE throughout this documentation. In the next three sections, following the research works [1–10], we present the algorithms for the methods of F-expansion, mF-expansion and FIM respectively.

3. ALGORITHM OF F-EXPANSION METHOD

This section explains the algorithm for the F-expansion method following [1–5] to obtain closed-form traveling-wave solutions of NLPDEs automatically. However, in the proposed algorithms, we have to guide the solution process initially by providing some initial data. In that sense, the algorithms are not fully automated, but this makes the algorithms more powerful, and we can apply these algorithms to a huge variant of NLPDEs with different types of initial data. We divide the algorithm into five main steps (labeled F1-F5).

Step F1(Transform the NLPDE into a NLODE): At first we take a transformation in Eq. (1)

$$u = U(\xi) e^{I\theta}, \tag{2}$$

where

$$\xi = k_0 t + k_1 x_1 + k_2 x_2 + \ldots + k_m x_m = \boldsymbol{K} \cdot \mathbf{X},$$

and

$$\theta = p_0 t + p_1 x_1 + p_2 x_2 + \ldots + p_m x_m = \boldsymbol{P} \cdot \mathbf{X}. \tag{3}$$

Here ξ is called traveling-wave coordinate and $U(\xi)$ is the traveling-wave part of the solutions. The second part of Eq. (2) is $e^{I\theta}$ which is called the phase part of the solutions and θ is called phase coordinate. Usually, this part is present when differential equation (1) has an imaginary part. In our program library we must retain this phase part when Eq. (1) is complex otherwise we only retain traveling-wave part. Taking proper forms or values of the constant coefficients (k_i $(i = 0 \ldots m)$) of traveling-wave coordinate ξ and phase angle constants p_i $(i = 0 \ldots m)$, we can transform the NLPDE (1) into the nonlinear ordinary differential equation (NLODE) with single independent variable ξ and dependent variable $U(\xi)$. Repeatedly applying the chain rule

$$\frac{\partial \bullet}{\partial \mathbf{X}} = \left(\boldsymbol{K} \frac{\mathrm{d}}{\mathrm{d}\xi} + I\boldsymbol{P} \right) \bullet \tag{4}$$

on Eq. (2), Eq. (1) is transformed into NLODE. The form of NLODE in general form is given by

$$G\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right) = 0, \tag{5}$$

where $U^{(n)} (n = 1, 2 \ldots)$ indicates n times differentiation with respect to ξ. All the three algorithms (F-expansion, mF-expansion, FIM) work only when Eq. (5) is a polynomial in variable $U(\xi)$ and its derivatives, and does not explicitly depend on the independent variables ξ. Then our algorithm check the integrability of Eq. (5). If Eq. (5) is integrable, our method try to integrate Eq. (5) and if integration is successful then one can assign a value to each integrating constant (ic_i, $i = 1, 2, \ldots, \eta$, η is the number of integration) in own choices. In the case of complex NLPDE, if real and imaginary both parts are present in the transformed NLODE, one part is taken to solve. To select one part from complex NLODE, we follow [10, 12]. In this context, the following strategies are followed:

i. Assume all the parameters (α_i, k_i, p_i) are real.

ii. Express the NLODE (5) in the form

$$G\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right) = Re\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right) + Im\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right) * I = 0, \tag{6}$$

where $Re\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right)$ is real part and $Im\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right)$ is imaginary part of NLODE (6), and $I = \sqrt{-1}$.

iii. Check in which part (Re or Im) dependent variable $U(\xi)$ is not present, but minimum one number of parameter is present there. Get the constraint on that parameters contained in this part. Take other part (may be Re or Im) as NLODE whose solutions are to be solved.

Suppose in the real part Re, the dependent variable $U(\xi)$ is not present and there is only present the parameters α_i, k_i, p_i. Then Eq. (6) can be expressed by

$$Re\left(\alpha_i, k_i, p_i\right) + Im\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right) * I = 0, \tag{7}$$

and we have to solve the NLODE $Im\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right) = 0$ subject to the constraint $Re\left(\alpha_i, k_i, p_i\right) = 0$.

iv. If the above step is failure, compare the expressions of Re and Im to check whether they are the same NLODE for some constraint on the parameters of Re or Im. If they are the same NLODE, take anyone part as NLODE whose solutions are to be solved.

Suppose for the constraint

$$f(\alpha_i, k_i, p_i) = 0, \tag{8}$$

we get

$$Re\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right) = Im\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)} \ldots\right) \tag{9}$$

from Eq. (6), then the solutions can be determined either from the NLODE Re or Im subject to the constraint relation (8).

If any above criteria are not satisfied, the complex input-NLPDE equation cannot be transformed into single NLODE in our algorithm and the method does not work successfully.

Step F2(Determine the highest power N of finite power series): Now according to the F-expansion method, the solution of the NLODE (5) can be expressed as the finite power series, which is

$$U = \sum_{i=0}^{N} a_i F^i(\xi) + \sum_{i=1}^{N} \frac{b_i}{F^i(\xi)}, \tag{10}$$

where $a_i (i = 0 \ldots N)$ and $b_i (i = 1 \ldots N)$ are constants to be determined later. In Eq. (10) first term is positive part and second term is negative part in the solution. The value of N (positive integer or positive noninteger) in Eq. (10) can be determined by considering a homogeneous balance between the highest order nonlinear term with the highest order derivative of $U(\xi)$ in Eq. (5). However, to automate this process, we have employed a method described in [13]. At first, Eq. (5) is expanded in the sum of product (SOP) form. Our aim is to determine the highest possible value of N, and so it is sufficient to replace U with U^N. Assuming degree of $U(\xi)$ is $D[U(\xi)] = N$, we replace U by U^N and collect the degrees of each term appearing in Eq. (5) by a variable, say E. To determine the degree of an expression, we use the relations

$$D\left[\frac{d^n U(\xi)}{d\xi^n}\right] = N + n, \; D\left[U^n \left(\frac{d^n U(\xi)}{d\xi^n}\right)^m\right] = Nn + m(N + n). \tag{11}$$

Generally in ($max(\ E)$,N) plane we get a turning point and the value of N is taken at this point. Whole procedure is automated in the following order:

i. Substitute U by U^N and simplify.

ii. Expand and express in SOP form.

iii. Collect the degree of each term appearing in SOP form and store them in a list E.

iv. Replace N in E by a sequence of numbers whose first number is 0. In GiNaCDE, we take three number sequence with common differences- $1/2, 1/3, 1/4$ up to last number 11.

v. For each number in sequence, calculate $max(E)$.

vi. Calculate differences between the value of $max(E)$ for successive numbers in the sequence.

vii. To get highest power N, take the number in sequence for which differences are not same with previous one. More clearly if i is the current number in the the sequence and $(max(E_i) - max(E_{i-1})) \neq (max(E_{i-1}) - max(E_{i-2}))$, then the highest power $N = i - 1$.

It is clear that for a larger value of N the complexity in mathematical operations is increased. To avoid such complexity in derivations, one should set the maximum allowed value of N. However, sometimes in some cases, the auto-evaluation of N may fail.

Step F3(Derive the system of Nonlinear Algebraic Equations for the coefficients of $F(\xi)$): $F(\xi)$ satisfy the first-order nonlinear ODE (also called auxiliary equation (A.E.))

$$F'(\xi) = \mathcal{F}(F(\xi)), \tag{12}$$

where $\mathcal{F}(F(\xi))$ is some known functions of $F(\xi)$. The prime over $F(\xi)$ represents differentiation with respect to ξ. In case of F-expansion method

$$\mathcal{F}(F(\xi)) = \sqrt{A_0 + A_1 F + A_2 F^2 + \ldots + A_\delta F^\delta}, \tag{13}$$

where δ is a positive integer and $A_i (i = 0, 1, \ldots \delta)$ are coefficients of A.E. Here one can use any functional form of Eq. (13) by choosing any positive integer value of δ and real values of A_i. As a result the solutions of first-order NLODE (13) can be expressed in terms of a large variety of functions such as polynomial, exponential, trigonometric, hyperbolic, rational, Jacobi elliptic etc.. However, taking some well-known functional forms of $\mathcal{F}$ in Eq. (13), we have shown some solutions of $F(\xi)$ in the Appendix: C, D. The higher derivatives of $F(\xi)$ using Eq. (12) can be expressed by

$$F'' = \frac{\mathrm{d}\mathcal{F}}{\mathrm{d}F}\mathcal{F}, \quad F''' = \left(\frac{\mathrm{d}\mathcal{F}}{\mathrm{d}F}\mathcal{F}\right)^2 + \mathcal{F}^2\frac{\mathrm{d}^2\mathcal{F}}{\mathrm{d}F^2} \text{ and so on.} \tag{14}$$

Now substituting Eq. (10) into Eq. (5) and using Eq. (14) with (13) we get an expression, and the numerator of resulted expression contains $F(\xi)^j \mathcal{F}^k$ $(j = 0, 1, 2, \ldots; k = 0, 1)$ terms. Setting each coefficient of $F(\xi)^j \mathcal{F}^k$ to zero an overdetermined system of nonlinear algebraic equations are obtained where the constant parameters $a_i (i = 0 \ldots N)$, $b_i (i = 1 \ldots N)$, $k_i (i = 0 \ldots m)$, $p_i (i = 0 \ldots m)$, $A_i (i = 0, 1, \ldots \delta)$, parameters $\alpha_i (i = 1 \ldots l)$ appearing in input-NLPDE and integrating constants $ic_i (i = 1 \ldots \eta)$ if (5) is integrable, are present.

The nonlinear system of equations is a set of simultaneous equations in which the unknowns (the constant parameters) appear as variables of a polynomial of degree one or higher than one. Suppose the system of algebraic equations is solved for all parameters that are present in the system of algebraic equations. In that case, it takes a larger time to get solutions for the system of equations, even sometimes solutions are not obtained for a complicated system. To reduce the calculating time and the complexity in derivations, we categorize all the parameters into two different types. They are external parameters and internal parameters. External parameters are $\alpha_i (i = 1 \ldots l)$, $A_i (i = 0, 1, \ldots \delta)$ which are present in input-NLPDE equation and auxiliary equation respectively. When input-NLPDE equations are integrable, the generated integrating constant(s) $ic_i (i = 1 \ldots \eta)$ is also external parameter. On the other hand all the remaining parameters, such as $a_i (i = 0 \ldots N)$, $b_i (i = 1 \ldots N)$, $k_i (i = 0 \ldots m)$, $p_i (i = 0 \ldots m)$ are called internal parameters as they generate internally. Nonlinear algebraic systems are always solved for internal parameters. But one can have control over external parameters to choose the parameters for which nonlinear algebraic system is to be solved. For this purposes, the programming variables *ASolve* and *paraInDiffSolve* (detailed descriptions are given in user manual) are used to choose the unknowns from the external parameters in our choice, and it will reduce the calculating time and can handle more complicated algebraic expressions. At the same time, the exact solutions are determined subject to the conditions on chosen external parameters.

Step F4(Solve the system of Nonlinear Algebraic Equations): Analysing and solving the nonlinear algebraic system is a vital and challenging step among all steps of the method. In fact, the number of exact solutions of NLPDE derived by F-expansion, modified F-expansion, or first integral methods is entirely depending on how many solutions are obtained from a nonlinear algebraic system. The executing time of the software mainly depends on this step. Many methods are available to solve nonlinear algebraic systems such that Gröbner basis methods [14], the Ritt-Wu characteristic sets method implemented by Wang [15, 16], and the Reduced Involutive Form (Rif) code by Wittkopf and Reid [17]. D. Baldwin et al. [18] have employed a simple algorithm to design a powerful nonlinear solver in Mathematica. We have followed their algorithm to create a nonlinear solver in GiNaC symbolic system. The nonlinear solver implemented in [18] solves the entire system in an automated way using the built-in Mathematica function *Reduce*. Their solver can solve polynomial and non-polynomial systems both. The nonlinear

solver implemented by us can solve only the polynomial system required in this application, and its own C++ function solves the polynomial equations.

The steps used in this algorithm are very much like the steps used to solve a nonlinear algebraic system by hand. In this method, the simplest equation is solved for sorted unknown parameters. Then the solutions are substituted in the remaining equations. Such solving and substitution procedures are repeated until the system is completely solved. We operate the whole procedure in the following order:

i. Check whether each equation is polynomial in unknowns.

ii. Factor and simplify each equation.

iii. Measure complexity of each equation by the number of add containers, unknown parameters, and the degree of unknowns. Then, sort the system based on their complexity. If more than one equations have the same complexity, in GiNaCDE they are sorted according to the GiNaC in-built comparison predicate *ex_is_less*.

iv. Sort the unknown parameters contained in the simplest equation by their degree.

v. Solve the simplest equation for the lowest degree unknown. If the number of unknown for the lowest degree is greater than one, then GiNaCDE uses the comparison predicate *ex_is_less* to choose the unknown. If solutions are absent, solve the unknown for the next higher degree.

vi. Substitute the solutions into the remaining equations and simplify.

vii. Repeat the steps i-vi until all the equations are reduced to zero.

viii. Substitute all the unknowns which are present in the computed solutions with the help of other solutions.

ix. Test the solutions by substituting them into each equation.

x. Finally, collect all solutions branches.

Our solver is powerful and can easily handle nonlinear equations (of course, polynomials in unknowns) with multi-parameters. Sometimes, there are risks of missing some solutions due to numerous parameters in the system or if the system is high degree. In this solver, the unknowns from all parameters appearing in the system are chosen in order of complexity. Then the solutions for these unknowns are expressed in terms of other parameters that are to be regarded as arbitrary parameters. Sometimes it is observed that the solutions become simpler where these arbitrary parameters are taken as unknowns.

Step F5(Build solutions with calculating steps): Substitute the solutions obtained in step F4 into Eq. (13) and obtain the solutions of F using the Appendix: C, D. Then, to obtain traveling-wave solutions of Eq. (5), substitute F and the solutions obtained in step F4 into Eq. (10). Finally the explicit solutions in original variables are obtained using Eqs. (2), (3).

4. ALGORITHM OF MODIFIED F-EXPANSION METHOD

In this section, we present the algorithm of modified F-expansion method following [6]. The algorithm for the modified F-expansion method [6] is very similar to the algorithm of the F-expansion method. This algorithm also has five main steps (labeled MF1-MF5), and it has only one difference to the F-expansion method. The difference is that a different form of A.E. is taken in step MF3 in comparison to step F3. Therefore, one can check new exact solutions of the NLPDE applying both methods (F-expansion method and modified F-expansion method) to the same NLPDE with different forms of A.E. Details of all steps are described below:

Step MF1(Transform the NLPDE into an NLODE): Same as step F1.
Step MF2(Determine the highest power N of finite power series): Same as step F2.
Step MF3(Derive the system of Nonlinear Algebraic Equations for the coefficients of $(F(\xi))$: In the modified F-expansion method, the solution of the NLODE (5) is also expressed by a finite series like Eq. (10). In this method, we have generalized the modified F-expansion method [6] taking the A.E. in more general form

$$F'(\xi) = A_0 + A_1 F + A_2 F^2 + \ldots + A_\delta F^\delta, \tag{15}$$

where δ is a positive integer and $A_i (i = 0, 1, \ldots \delta)$ are coefficients of A.E. One can choose any functional form of Eq. (15) using any positive integer value of δ and any real value of A_i. Here interestingly, we note that by choosing various functional forms of Eq. (15) in our choices, one can get the final solutions of input-NLPDE in terms of a large variety of functions. For example, some well-known equations can be obtained from Eq. (15), such as Riccati equation with $\delta = 2$ and Bernouli equation with $A_i = 0$, $(i \neq 1$ and $i \neq \delta)$. The exact solutions of Riccati and Bernouli equations are known that are given in Appendix: A and Appendix: B respectively. Now substituting Eq. (10) into Eq. (5) and using Eq. (15) we get an expression appearing the terms $F(\xi)^j$ $(j = 0, 1, 2, \ldots)$ in the numerator. The equations must vanish identically. Hence, to generate a nonlinear algebraic system, equate to zero the coefficients of the power terms in F.

Step MF4(Solve the system of Nonlinear Algebraic Equation): Similar strategy as in step F4.

Step MF5(Build solutions with calculating steps): Substitute the solutions of step MF4 into Eq. (15). Obtain the solutions of F using Appendix: A, B. Then, substitute F along with the solutions of step MF4 into Eq. (10). To get the explicit solutions in original variables, Eqs. (2), (3) are used.

5. ALGORITHM OF FIRST INTEGRAL METHOD

In this section, we present the algorithm of first integral method following [7–10]. In first integral method [7–10], one important advantage over F-expansion and modified F-expansion methods is that one does not have to choose A.E. to solve NLPDEs; instead, the input-NLPDE is automatically reduced to a suitable first-order NLODE whose solutions have to be calculated.

The algorithm for the automated first integral method has eight main steps (labeled FIM1-FIM8). Now we give an outline of every step as follows:

Step FIM1(Transform the NLPDEs into NLODEs): Same as step F1. The condition for applying first integral method to the Eq. (5) is that Eq. (5) must be a second-order NLODE. Therefore Eq. (5) is expressed in the form

$$\mathbf{G}\left(\alpha_i, k_i, p_i, U, U^{(1)}, U^{(2)}\right) = 0. \tag{16}$$

Step FIM2(Convert into a system of NLODEs): We assume that $U(\xi) = X(\xi)$ and introducing a new independent variable $Y(\xi) = X_\xi(\xi)$, Eq. (16) can be rewritten as a system of NLODEs [7]

$$X_\xi(\xi) = Y(\xi), \tag{17a}$$

$$Y_\xi(\xi) = \frac{P(X(\xi), Y(\xi))}{H(X)} = \frac{1}{H(X)}\left(K_0(X) + K_1(X)Y + \ldots + K_d(X)Y^d\right). \tag{17b}$$

We have expressed $P(X(\xi), Y(\xi))$ as a polynomial in variable $Y(\xi)$ with degree d and $H(X), K_i(X)(i = 0, 1 \ldots d)$ are polynomials in variable X. $H(X)$ is the coefficient of the highest derivative term in Eq. (16).

Step FIM3(Apply Division Theorem): If $X(\xi), Y(\xi)$ are nontrivial solutions of Eq. (17), then applying the Division Theorem [7] there exist an irreducible polynomial in the complex domain $C[X, Y]$ such that

$$q(X(\xi), Y(\xi)) = \sum_{i=0}^{N} a_i(X)Y^i = 0, \tag{18}$$

where $a_i(i = 0 \ldots N)$ are polynomials of X and $a_N \neq 0$. Equation (18) is called the first integral to Eqs. (17a) and (17b). Using Division Theorem there exists a polynomial $(g(X) + h(X)Y)$ such that

$$\frac{dq}{d\xi} = \frac{\partial q}{\partial X}\frac{dX}{d\xi} + \frac{\partial q}{\partial Y}\frac{dY}{d\xi} = (g(X) + h(X)Y)\sum_{i=0}^{N} a_i(X)Y^i. \tag{19}$$

Using Eqs. (17), (18) in the Eq. (19), we get

$$\sum_{i=0}^{N} \dot{a}_i(X)Y^{i+1} + \sum_{i=0}^{N} i a_i(X)Y^{i-1}\frac{1}{H(X)}\left(K_0(X) + K_1(X)Y + \ldots + K_d(X)Y^d\right) = (g(X) + h(X)Y)\sum_{i=0}^{N} a_i(X)Y^i. \tag{20}$$

Dot over $a_i(X)$ denotes derivative with respect to X. The degree in variable Y of left hand side (L.H.S) in Eq. (20) is $i+d-1$ and the degree in variable Y of right hand side (R.H.S) in Eq. (20) is $i+1$. Balancing degrees between both sides we get $i+d-1=i+1$, hence $d=2$. So the method is applicable when the degree of Eq. (17b) in variable Y is less than or equal to 2. Taking the maximum degree 2, Eq. (20) can be rewritten as

$$\sum_{i=0}^{N}\dot{a}_i(X)Y^{i+1}+\sum_{i=0}^{N}ia_i(X)Y^{i-1}\frac{1}{H(X)}\left(K_0(X)+K_1(X)Y+K_2(X)Y^2\right)=\left(g(X)+h(X)Y\right)\sum_{i=0}^{N}a_i(X)Y^i. \tag{21}$$

Step FIM4(Derive the Algebraic System of equations for coefficients of Y^i): Comparing coefficients of Y^i $(i=N+1,N,\ldots,1,0)$ on both sides of (21), and for $H(X)\neq 0$ canceling $H(X)$ in denominator from both sides we obtain

$$Y^{N+1}:\ H(X)\dot{a}_N(X)+Na_N(X)K_2(X)=H(X)h(X)a_N(X), \tag{22a}$$
$$Y^{N}\ :\ H(X)\dot{a}_{N-1}(X)+NK_1(X)a_N(X)+(N+1)K_0(X)a_{N+1}(X)=H(X)g(X)a_N(X)+H(X)h(X)a_{N-1}(X), \tag{22b}$$
$$\vdots$$
$$Y^{1}\ :\ H(X)\dot{a}_0(X)+K_1(X)a_1(X)+2K_0(X)a_2(X)=H(X)g(X)a_1(X)+H(X)h(X)a_0(X), \tag{22c}$$
$$Y^{0}\ :\ K_0(X)a_1(X)=H(X)g(X)a_0(X), \tag{22d}$$

where $a_i(X)=0$ for $i<0$ and $i>N$.

In the next step FIM5 we take $a_N=1$ to derive the polynomial forms of $h(X),g(X),a_i(i=0.\ldots.N-1)$. For $a_N=1$ from Eq. (22a) we obtain

$$h(X)=\frac{NK_2(X)}{H(X)}. \tag{23}$$

If $H(X)$ is not a constant and at the same time degree of $P(X(\xi),Y(\xi))$ in variable Y is 2 then it is clear from Eqs. (22a), (23) that $h(X)$ will not be polynomial in X. In this case we avoid such non-polynomial form of $h(X)$ by making the transformation [9]

$$d\xi=H(X)d\eta, \tag{24}$$

in Eq. (17) temporarily. Applying the transformation in Eq. (17) we get

$$X_\eta(\eta)=H(X)Y, \tag{25a}$$
$$Y_\eta(\eta)=K_0(X)+K_1(X)Y+\ldots+K_d(X)Y^d. \tag{25b}$$

Consequently the nonlinear algebraic system becomes

$$Y^{N+1}:\ H(X)\dot{a}_N(X)+Na_N(X)K_2(X)=h(X)a_N(X), \tag{26a}$$
$$Y^{N}\ :\ H(X)\dot{a}_{N-1}(X)+NK_1(X)a_N(X)+(N+1)K_0(X)a_{N+1}(X)=g(X)a_N(X)+h(X)a_{N-1}(X), \tag{26b}$$
$$\vdots\qquad\qquad\qquad\qquad\vdots$$
$$Y^{1}\ :\ H(X)\dot{a}_0(X)+K_1(X)a_1(X)+2K_0(X)a_2(X)=g(X)a_1(X)+h(X)a_0(X), \tag{26c}$$
$$Y^{0}\ :\ K_0(X)a_1(X)=g(X)a_0(X). \tag{26d}$$

Now for $a_N=1$, from Eq. (26a) we get $h(X)=NK_2(X)$ which is polynomial in X. In the following steps, we explain all the procedures with the help of Eq. (22), because the same procedures are applicable when Eq. (26) is considered for the non-polynomial case of $h(X)$.

Step FIM5(Determine degrees of $h(X),g(X),a_i(X)(i=0\ldots N)$ and express them in polynomial forms): For simplicity substitute $a_N=1$ in Eq. (22a) (in Eq. (26a) for non-polynomial case of $h(X)$) and obtain polynomial form of $h(X)$.

To determine polynomial forms of $g,a_i(i<N)$, the degrees of X between L.H.S and R.H.S in each equation of Eqs. (22) (in Eqs. (26) for non-polynomial case of $h(X)$) are balanced. The balancing process is implemented in the following order:

i. Replace $g \to X^{d_g}, h(X), a_i \to X^{d_{a_i}} (i < N)$ in each equation from (22b) to (22d). Here should be noted, if degree of $P(X(\xi), Y(\xi))$ in variable Y is 2 and $H(X)$ is not a constant (i.e. the non-polynomial case of $h(X)$) then the replacements are done in Eqs. (26) instead of Eqs. (22).

ii. d_g, d_{a_i} are replaced by the sequence $0, 1, 2, \ldots$ of all positive integers arranged in increasing order. To avoid an infinite loop, sequences are taken up to a certain maximum number. In our library, the maximum number is 5. Sometimes it is impossible to balance the equations for any positive integer numbers

iii. Take the numbers from sequences at which L.H.S and R.H.S of each equation are balanced in the degree of X.

After balancing, if d_g, d_{a_i} $(i < N)$ are degrees of g, a_i $(i < N)$ respectively, then g, a_i are expressed by

$$g = g_0 + g_1 X + \ldots + g_{d_g} X^{d_g}, \tag{27a}$$

$$a_0 = a_{00} + a_{01} X + \ldots + a_{0 d_{a_0}} X^{d_{a_0}}, \tag{27b}$$

$$a_1 = a_{10} + a_{11} X + \ldots + a_{1 d_{a_1}} X^{d_{a_1}}, \tag{27c}$$

$$\vdots = \qquad \vdots$$

$$a_{N-1} = a_{(N-1)0} + a_{(N-1)1} X + \ldots + a_{(N-1) d_{a_{(N-1)}}} X^{d_{a_{(N-1)}}}. \tag{27d}$$

Where $g_i (i = 0, 1, \ldots, d_g), a_{ij} (i = 0, 1, \ldots, N-1, j = 0, 1, \ldots, d_{a_i})$ are arbitrary constants. In GiNaCDE, all possible combinations of balanced degrees (whose values are < 6) for g, a_i are calculated, and for each combination, the solutions of input-NLPDE are derived.

Step FIM6(Derive the Nonlinear Algebraic System for the parameters g_i, a_{ij}): Substitute $a_N = 1$ and (27) into each equation of (22). The coefficients of the power in X, Y in each equation of (22) must vanish. Collect the coefficients and generate a nonlinear algebraic system of equations parametrized by $g_i, a_{ij}, \boldsymbol{K}, \boldsymbol{P}$, integrating constants (for integrable NLPDE) and parameters appearing in input-NLPDE.

Step FIM7(Solve the Nonlinear Parameterized Algebraic System): Here, the external parameters are parameters appearing in input-NLPDE and integrating constants. Internal parameters are $g_i, a_{ij}, \boldsymbol{K}, \boldsymbol{P}$. The nonlinear algebraic system is solved following a similar process in step F4. Like step F4, here also, the runtime of this algorithm mainly depends on this step.

Step FIM8(Build solutions): The solutions in step FIM7 are substituted in Eq. (18) and using $Y(\xi) = U_\xi(\xi)$, Eq. (18) converts into first-order NLODE called first integral form of Eq. (16). Some well-known forms of first-order NLODE with solutions have been listed in Appendix: A, B, C, D. If first integral form matches with any form of listed NLODE in Appendix, solutions are shown, otherwise the program shows only the first integral forms. Combining Eqs. (2), (3), we obtain final solutions of input-NLPDE in original variables.

6. SOFTWARE IMPLEMENTATION

We have implemented the algorithms described in Secs. 3, 4 and 5, into GiNaCDE. GiNaCDE is a C++ library that is built on a pure C++ symbolic library GiNaC [11]. Besides this library version, we have also developed a GUI version of GiNaCDE called GiNaCDE GUI. When we solve differential equations using GiNaCDE GUI, we do not have to write any C++ code, and compilation of any code is not required. This GUI version guides us in each step to obtain the output results. However, a complete guide of GiNaCDE GUI has been provided in the GiNaCDE repository. In both versions of GiNaCDE (library and GUI), the output results are saved in a text file with calculating steps. Output results can be saved in Maple, Mathematica or GiNaC programming language by assigning the C++ macros *maple*, *mathematica* or *ginac* to the programming variable *output*.

F-expansion and mF-expansion methods can be applied to higher-order NLPDEs. But, FIM method is applicable to an NLPDE when its transformed NLODE (16) is second-order only. Actually, there are no rules to know in advance the appropriate method among three to solve the given NLPDE.

In GiNaCDE, to derive the solutions of NLPDEs we assume all the constant parameters ($g_i, a_{ij}, \boldsymbol{K}, \boldsymbol{P}$ etc.) are strictly real, positive, and the rules like $\sqrt{a^2} \to a, \sqrt{-a^2} \to Ia, \sqrt{-a} \to I\sqrt{a}$ are successively applied. Such simplification rules are used in Maple CAS with *simplify* routine along with *assume=positive* or *symbolic* option. One demerit of such rules is that the solutions which exist for negative values of the parameters may be missed.

In order to start a solution process for a given NLPDE or NLODE in GiNaCDE, we require some initial data (the options and parameters specified by the user). Some programming variables such as *twcPhase, positivePart, negativePart, NValue, degAcoeff, ASolve, paraInDiffSolve* (detailed descriptions of these variables are given in the Sec. 10) are available, which can be initially set up by the users in their own choices for getting better results for a given NLPDE before starting the solution process in the GiNaCDE software. There is no rule to know in advance a specific type of initialization to get better results.

In all three methods, if the input NLPDE or NLODE is complex, then the software tries to separate the real and imaginary parts following the step F1. In all three methods, if the input NLPDE or NLODE is integrable, the software tries to integrate them after starting the solution process. If the integration is successful, then the software gives an option to us to assign a numerical or symbolic value to the integrating constant(s) ic_i $(i = 1, 2, ..)$ in our choices. All these options make the software more powerful and flexible, enhancing its ability to find many new exact solutions to huge variants of NLPDEs.

Now we shall discuss some implementation details of each method separately:

F-expansion and mF-expansion methods: These methods have been implemented in *F_expns_methd.cpp* and *F_expns_methd.h* files. The F-expansion and mF-expansion methods are chosen by the C++ macros *F_expansion, mF_expansion* respectively. One can use the coordinates $\boldsymbol{K}, \boldsymbol{P}$ in own choices with the help of the programming variable *twcPhase*. We can take any one or both parts in the solutions (10) with the help of the programming variables *positivePart*, *negativePart*. In GiNaCDE, we set the maximum allowed value of N at 10. However, sometimes in some cases, the auto-evaluated value of N exceeds 10. Then this step fails to find N. In this case, we can check the solutions of input-NLPDE by specifying the value of N in our choice lower than 10 with the help of the programming variable *NValue*. If we do not assign any value to the variable *NValue*, the value of N is auto-evaluated following the criteria in step F2. In these methods, we initially have to input A.E. to start the solution process in GiNaCDE manually. One can choose the parameters $A_i (i = 0, 1, \ldots \delta)$ in own choices with the help of the programming variable *degAcoeff*. There are no rules to know in advance what type of A.E. can give exact solutions for a given NLPDE. In this context, we have shown the solutions of some well-known A.E. in the Appendix: A, B, C, D. If our chosen A.E. matches with any form of listed A.E. in Appendix, solutions are shown, otherwise the program shows only the A.E. The variable *ASolve* confirms whether the nonlinear algebraic system will be solved for the parameters contained in A.E. (i.e., the parameters A_i where $i = 0, 1, \ldots \delta$) along with other parameters. The parameters appearing in input-NLPDE (these parameters are belong to external parameters) are supplied in the programming variable *paraInDiffSolve* to solve the nonlinear algebraic system for those parameters also. This will determine the conditions on that external parameters so that exact solutions are obtained.

FIM: This method has been implemented in *fim.cpp* and *fim.h* files. This method is chosen by the C++ macro *FIM*. For the first integral method, we do not have to input A.E. Here, for initializations, we have only three programming variables *twcPhase, NValue, paraInDiffSolve*. The variables *twcPhase, paraInDiffSolve* have been discussed above. Like F-expansion method the value of N is also assigned by the variable *NValue* and the default value is $N = 1$. However, in our library, the allowed values of *NValue* are 1 and 2.

7. COMPILING AND INSTALLING

We need to make sure that we already have a few tools installed, including pkg-config (>= 0.29.2), CMake(>=3.1), CLN (>= 1.3.4), GiNaC (>= 1.7.6). We need a decent ISO C++11 compiler. We suggest to use the C++ compiler from the GNU compiler collection, GCC >= 4.9. The GTK+ 3.xx libraries (optional) are required to build the graphical user interface (GUI) of GiNaCDE library. To install all these dependencies please see the README file.

The source files are compiled using CMake build system by executing the following commands with Make and the GCC compiler on the command-line:

```
1. mkdir build-dir # generate a separate directory
2. cd build-dir
3. cmake -DGINACDE_GUI_BUILD=on <path-to-source> # generate Makefiles
4. make
5. make install
```

On line 3, the option *-DGINACDE_GUI_BUILD=on* is used to build GUI of GiNaCDE library, which requires GTK+ 3.xx libraries. If GTK+ 3.xx libraries are absent in the system, one can omit this option. The name of source directories is used in place of *<path-to-source>*. We have checked that the source files are successfully compiled on the Windows platform using MSYS2 when the commands on line 3 are simply replaced by

```
3. cmake -G "MSYS Makefiles" -DGINACDE_GUI_BUILD=on <path-to-source>
```

If you have just now installed the library, sometime it is necessary to run

```
1. ldconfig
```

which creates the necessary links and cache to the most recent shared libraries installed in the system. To test GiNaCDE library after building it, simply type the following

```
1. ctest
```

8. ARITHMETIC OPERATORS AND SOME SUPPORTED FUNCTIONS IN LIBRARY

In writing an algebraic expressions GiNaCDE follows GiNaC rules. Like GiNaC, GiNaCDE supports all the arithmetic operators $*$ (multiplication), $+$ (addition), $-$ (subtraction), / (division). GiNaCDE's C++ library version does not support the operator ^. For this purpose, an object of class *power* (in short *pow*) play the role of ^ operator and this *power* object has two slots, one for the basis, one for the exponent, such as x^5 is written as $pow(x, 5)$. Here one should note that GiNaCDE's GUI version supports the operator ^ also. The imaginary unit in GiNaCDE has been predefined in GiNaC as a numeric object with the name I. GiNaCDE supports the function *conjugate()* which finds the complex conjugate of mathematical expressions, such as complex conjugate of x is determined by *conjugate(x)*. GiNaCDE also supports the function *sqrt()* which finds square root.

The relational operators which express a relation for two *ex* objects, signals equality, inequality, and so on between them. These relations are created by simply using the C++ operators $==, != , <, <=, >$ and $>=$ between two expressions.

9. THE PARAMETERS OF THE DIFFERENT METHODS USED IN LIBRARY

Many parameters are used in the three different methods. All parameters have been well described in Secs. 3, 4 and 5. In the software, we have used the same parameter's names as used in the above descriptions of the algorithms, only in the case of first integral method some parameter's (g, h) and algebraic symbol's (X, Y) name followed by an underscore in the software to avoid conflict with user-provided names. These parameters are $g\_, h\_$ and symbols are $X\_, Y\_$. During derivations, the software always generate some parameters, their names are $a_i (i = 0, 1, 2, \ldots), b_i (i = 0, 1, 2, \ldots), a_{ij} (i = 0, 1, 2, \ldots, j = 0, 1, 2, \ldots), g_i (i = 0, 1, 2, \ldots), N, d_g, d_{a_i} (i = 0, 1, 2, \ldots)$. When using GiNaCDE, we will always choose the name of the parameter such that its name does not conflict with these parameters generated by the software.

Here one should note that since the underscore has a special meaning in Mathematica, it is not allowed in Mathematica identifiers. So underscore is not used in Mathematica output format of GiNaCDE.

10. SOME USEFUL COMMANDS IN LIBRARY

In the following we have dealt with some useful commands of GiNaCDE library required to write C++ code. In particular, for more details about the commands *ex, lst*, we refer to the tutorial of GiNaC library [11].

ex: Any algebraic expressions or numbers are handled by the GiNaC class *ex* [11].

lst: The GiNaC class *lst* is a container which can stores list of arbitrary expressions [11].

NValue: *NValue* is an *extern ex* variable. The value of N is assigned by the variable *NValue*. In the case of F-expansion and mF-expansion methods, if *NValue* is not assigned by the user, it is auto-evaluated following the algorithm in step F2. In the case of FIM, the default value of *NValue* is 1, i.e., if *NValue* is not assigned by the user, its value is 1.

positivePart, negativePart: Due to the presence of a huge number of parameters in the problem, sometimes it is tough to get solutions. To overcome such a problem as far as possible, we have introduced these *boolean* variables. In F-expansion and mF-expansion methods, the power series (10) has two parts, first term is positive part and second term is negative part. We can take any one or both parts in the solutions with the help of these *boolean* variables. The definitions *positivePart = true; negativePart = false;* will take the positive part. The definitions *positivePart = false; negativePart = true;* will take negative part, and the both parts are taken with the definitions *positivePart = true; negativePart = true;*. By default the last definition is active. These *boolean* variables are only available in F-expansion and mF-expansion methods.

ASolve: Sometimes, due to the presence of a significant number of parameters in the nonlinear algebraic system, the solver may fail to solve the system. To remove such a problem as far as possible, we have also introduced another boolean variable *ASolve*. It confirms whether the nonlinear algebraic system will be solved for the parameters contained in A.E. (13) or (15) (i.e., the parameters A_i where $i = 0, 1, \ldots \delta$) along with other parameters. When the definition $ASolve = false$ is taken, the nonlinear algebraic system will not be solved for A_i, and A_i becomes arbitrary. With the definition $ASolve = true$ (by default), the algebraic system will be solved for A_i along with other parameters and also determine the conditions among A_i to get exact solutions. It takes more solving time than before. This *boolean* variable is only available in F-expansion and mF-expansion methods.

paraInDiffSolve: *paraInDiffSolve* is an *extern lst* variable. The parameters appearing in input-NLPDE (this parameters are belong to external parameters) are supplied in *lst* variable *paraInDiffSolve* to solve the nonlinear algebraic system for those parameters also. This will determine the conditions on that external parameters so that exact solutions are obtained.

twcPhase: *twcPhase* is an *extern lst* variable. The parameters $\boldsymbol{K}, \boldsymbol{P}$ are provided through this variable in following manner

$$twcPhase = \{lst\{k_0, k_1, \ldots, k_m\}, lst\{p_0, p_1, \ldots, p_m\}\}; \tag{28}$$

degAcoeff: *degAcoeff* is an *extern lst* variable. The positive integer (δ) in A.E., the parameters contained in A.E. ($A_i, i = 0, 1, \ldots \delta$) are provided through this variable in this way

$$degAcoeff = \{\delta, A_0, A_1, \ldots, A_\delta\}; \tag{29}$$

This variable is only available in F-expansion and mF-expansion methods.

ex reader(string "algbExpr"): We can create algebraic expressions from the *string "algbExpr"* using the command *reader*. The generated expressions are *ex* objects.

ex simplify(string "algbExpr"): The *simplify* command is used to apply simplification rules (such as power rule ($(x^m)^n \to x^{mn}$), combine rule ($x^m x^n \to x^{m+n}$), factor, normalization etc.) to an algebraic expression. Generally when an algebraic symbol (let a) is real, the algebraic rule $\sqrt{a^2} \to |a|$ can be applied. But in our case to derive the solutions of NLPDEs we assume all the constant parameters ($g_i, a_{ij}, \boldsymbol{K}, \boldsymbol{P}$ etc.) are strictly real, positive, and the rules like $\sqrt{a^2} \to a, \sqrt{-a^2} \to Ia, \sqrt{-a} \to I\sqrt{a}$ are successively applied. In Maple CAS, *simplify* routine along with *assume=positive* or *symbolic* option apply similar rules. One demerit of such rules is that the solutions which exist for negative values of parameters may be missed.

depend(ex f1, lst f2): This sets up a dependency of a variable in the first argument *f1* on the list of algebraic variables *f2*. The dependency of *f1* on a particular variable can be removed by the command *depend.clear(f1, variable)* and the dependencies on all variables are removed by the command *depend.clear(f1)*. One should maintain the same order of independent variables and the corresponding coefficients when they are placed in list *f2* and in the programming variables *twcPhase* respectively.

ex pdiff(ex expression, ex variable, ex order): The *pdiff* command computes the partial derivative of the *expression* with respect to *variable*. *order* is the times of differentiation.

ex Diff(ex expression, ex variable, ex order): *Diff* command returns the partial derivative in unevaluated form.

ex integrate(ex expression, ex variable): The integrate command computes the integration of the *expression* with respect to *variable*.

ex Integrate(ex expression, ex variable, ex order): *Integrate* command returns the integration in unevaluated form.

ex evaluate(ex expression): The evaluations of *Diff* and *Integrate* commands are made by *evaluate* function.

exsetlst solve(lst equations, lst variables): A list of equations in 1st argument are solved for a list of variables in 2nd argument. The equations must be the polynomials in the variables for which the equations are solved. It returns the solutions in *exsetlst* container. *exsetlst* is the *typedef* of $std :: set < lst, ex\_is\_less >$.

desolve(ex differential_equation, lst dependent_variable, int method): The *differential_equation* is solved for the *dependent_variable*. To choose the solution method, one has to be assigned the 3rd argument by a *macro*. There are three *macros- F_expansion, mF_expansion, FIM* and the corresponding solution methods are respectively F-expansion, modified F-expansion and first integral methods.

ex checkSolu(string diff_equ, string solutions, string algebraic_solutions, string solutions_conditions) : This function checks the solutions of differential equation reported by GiNaCDE. Here all the function arguments are std::string class in C++. This function checks the solutions *solutions* of differential equation *diff_equ* for the solutions *algebraic_solutions* of nonlinear algebraic equations with the conditions *solutions_conditions* of solutions *solutions*.

This function uses substitution method to check the solution returned by GiNaCDE. The substitution method involves substituting the solution back into the differential equation. If the solution is valid, this function returns 0. However, currently, GiNacDE is unable to check all the solutions reported by GiNaCDE due to some simplification problems. I hope this problem can be fixed in the future release of GiNaCDE. Now to verify the solutions, I recommend to use Maple or Mathematica software.

output: Output results can be saved in Maple, Mathematica or GiNaC programming language by assigning *maple*, *mathematica* or *ginac* macros to this variable. This option gives us an additional advantage to make further calculations using the output results in the commercially available softwares such as Maple or Mathematica.

filename: The file name in a *string* is assigned to this variable. Output results are saved in this file.

11. INPUT AND OUTPUT OF LIBRARY

In order to solve the NLPDEs with the help of GiNaCDE three tools are available, they are GiNaCDE library (programming interface), GiNaCDE GUI (graphical user interface) and gtools (console interface). In this tutorial we have described about the GiNaCDE library. A short tutorials on GiNaCDE GUI and gtools are given in separate manuals. For solving NLPDE, we have to input an NLPDE in proper syntax. For example we consider the following NLPDEs

$$u_{xt} - 4u_x u_{xy} - 2u_y u_{xx} + u_{xxxy} = 0; \tag{30}$$

$$Iu_t + u_{xx} + 2\Big(|u|^2\Big)_x u + |u|^4 u = 0; \tag{31}$$

and the input syntax of the above NLPDEs are

```
Diff(Diff(u,x,1),t,1) - 4*Diff(u,x,1)*Diff(Diff(u,x,1),y,1) - 2*Diff(u,y,1)*Diff(u,x,2)
+ Diff(Diff(u,x,3),y,1)==0;
I*Diff(u,t,1) + Diff(u,x,2) + 2*u*Diff(u*conjugate(u),x,1) +
u*u*conjugate(u)*conjugate(u)*u==0;
```

The software determines exact solutions with calculating steps, and the results are saved in an output file.

Beside this, the solutions of the NLPDE are collected by a variable *solutionClt*. The variable *solutionClt* is a container of type *vector<lst>*. *solutionClt* holds the list of solutions. Invariably, the first element in each list is the solutions of the nonlinear algebraic system, and the remaining elements are the exact solutions of the given NLPDE. Now, we exemplify the usage of the variable *solutionClt*. Let us consider *solutionClt* has collected four set of solutions which can be obtained by calling *solutionClt[0]*, *solutionClt[1]*, *solutionClt[2]* and *solutionClt[3]*. Let us assume, *solutionClt[0]* has stored the following solutions

```
{{a_0==f_1(k_0,k_1),a_1==f_2(k_0,k_1)},u==g_1(k_0,k_1),{u==g_3(k_0,k_1),h(k_0,k_1)==0}}
```

In the aboe example, the first element *solutionClt[0][0]* is GiNaC *lst* container which is the list of solutions of nonlinear algebraic system. The remaining elements *solutionClt[0][1]* and *solutionClt[0][2]* represent the exact solutions of NLPDE with dependent variable u. Here, it should be noted that the solutions stored in *solutionClt[0][2]* is a GiNaC *lst* container. In this container, the first element *solutionClt[0][2][0]* is the exact solution of u and the second element *solutionClt[0][2][1]* is the condition for which this solution exists. In similar way, other set of solutions are collected by *solutionClt[1]*, *solutionClt[2]* and so on.

There is also an another variable named *constraints* which is a GiNaC *lst* container. *constraints* contains the conditions on the parameters for which all the exact solutions of u are determined and stored in the variable *solutionClt*.

11.1. Additional Note

Here one should note that one important property of GiNaC that differentiates it from other computer algebra programs we may have used: GiNaC assigns a unique (hidden) serial number for each newly created symbol object, and GiNaC uses this unique serial number instead of its name for algebraic manipulations. The serial number for the same name of a symbol may be changed in each running session of the GiNaC program. As a result, the symbols in the same algebraic expressions in the results obtained in the output files, may be ordered differently during each running session of the GiNaCDE program. This happens because to order the symbols of an algebraic expression GiNaC internally uses a comparison predicate, called *ex_is_less* which uses an internal symbol id counter.

12. WRITING PROGRAMS USING LIBRARY

In this section, we explain how to write C++ programs using the GiNaCDE library for solving NLPDEs. We use two examples (NLS and KdVB equations) to illustrate each line of the C++ program.

12.1. NLS Equation

For solving one dimensional cubic nonlinear Schrödinger (NLS) equation

$$Iu_t - pu_{xx} + q|u|^2u = 0, \tag{32}$$

by *F_expansion* and *FIM* method using GiNaCDE library, the C++ codes are (here p, q are non-zero real constants and $u(x,t)$ is a complex-valued function depends on the variables t, x.)

```
// NLS.cpp
#include <GiNaCDE/GiNaCDE.h>
int main()
{
1.    const ex u=reader("u"), t=reader("t"), x=reader("x"), p=reader("p"), q=reader("q"),
k_0=reader("k_0"),k_1=reader("k_1"),p_0=reader("p_0"),p_1=reader("p_1"),
A_0=reader("A_0"),A_2=reader("A_2");
2.    depend(u, {t, x});
3.    ex pde = I*Diff(u,t,1)-p*Diff(u,x,2)+q*pow(u,2)*conjugate(u);
4.    output = maple;
5.    twcPhase={lst{k_0,k_1},lst{p_0,p_1}};
6.    degAcoeff = {2,A_0,0,A_2};
7.    ASolve=false;
8.    positivePart = true;
9.    negativePart = true;
10.   paraInDiffSolve={};
11.   filename = "NLS_Fexp.txt";
12.   desolve(pde,{u},F_expansion);
13.   output = mathematica;
14.   filename = "NLS_FIM.txt";
15.   desolve(pde, {u}, FIM);
16.   return 0;
}
```

To compile the above code into an executable, we run the following command from a terminal:

```
g++ -Wall -g NLS.cpp -o NLS -lcln -lginac -lGiNaCDE
```

Next we explain the commands in each line of the above program. In line 1 we define some mathematical objects that are handled by the GiNaC class *ex* [11]. In line 2 we make dependency of u on the independent variables *t, x*. In line 3 we are storing Eq. (32) in a variable *pde* which is a GiNaC class *ex*. We can save all the output results in Maple (line 4) or Mathematica (line 13) format. We now transform NLPDE (32) to NLODE using the relation

$$u(t,x) = U(\xi)e^{Ip_0t+Ip_1x}, \quad \text{where } \xi = k_0t + k_1x. \tag{33}$$

The traveling-wave coordinate (ξ) is related to independent variables by the relation (33) and we set the constant coefficients of independent variables in 1st list of the *lst* variable *twcPhase* in line 5. Since Eq. (32) is a complex NLPDE, we define constant coefficients of independent variables in phase part in 2nd list of the *lst* variable *twcPhase* in line 5. It is worth noting that the order of the constant coefficients in line 5 must match the corresponding order of independent variables that is here $\{t, x\}$ in line 2. We have taken A.E. in the form

$$F' = \sqrt{A_0 + A_2 F^2}. \tag{34}$$

which is set by *lst* variable *degAcoeff* in line 6. The first element of the variable *degAcoeff* is positive integer $\delta = 2$ in the A.E. and the rest of the elements are the constant coefficients $A_0, 0, A_2$ of $F^i (i = 0\ ..2)$ respectively. In line 7, *ASolve* is assigned to *false* and so the nonlinear algebraic system is not solved for the parameters A_0, A_2 here. The definitions used in lines 8 and 9 ask the library to put positive and negative both parts in the solutions. In line 10, we have not supplied the parameters p, q appearing in input-NLPDE (32). In line 11, we give a name of a file in which all the output results with calculating steps are saved. Finally, the equation is solved using the F-expansion method by the command *desolve* in line 12, and all the output results with calculating steps are saved in the file *NLS_Fexp.txt*.

The Command in line 15 solve Eq. (32) by *FIM* method. Output results with calculating steps are saved in the file *NLS_FIM.txt*.

12.2. KdVB Equation

For solving well-known KdVB equation

$$u_t + uu_x - pu_{xx} + qu_{xxx} = 0, \tag{35}$$

by modified F-expansion method the C++ codes are (here p, q are non-zero real constants)

```
// KdVB.cpp
#include <GiNaCDE/GiNaCDE.h>
int main()
{
1.    const ex u=reader("u"), t=reader("t"), x=reader("x"),p=reader("p"), q=reader("q"),
k_0=reader("k_0"), k_1=reader("k_1"), A_0=reader("A_0"), A_1=reader("A_1"),
A_2=reader("A_2");
2.    depend(u, {t,x});
3.    ex pde =Diff(u,t,1)+Diff(u,x,1)*u-p*Diff(u,x,2)+q*Diff(u,x,3);
4.    output = maple;
5.    twcPhase = {lst{k_0,k_1},lst{}};
6.    degAcoeff = {2,1,1,A_2};
7.    ASolve=true;
8.    positivePart = true;
9.    negativePart = true;
10.   paraInDiffSolve={};
11.   filename = "KdVB_mF.txt";
12.   desolve(pde, {u}, mF_expansion);
13.   return 0;
}
```

The input-NLPDE (35) is assigned to the variable *pde* in line 3 and it is transformed into NLODE with the traveling-wave coordinate (ξ) using the transformation

$$u(t, x) = U(\xi) \ \text{ where } \ \xi = k_0 t + k_1 x, \tag{36}$$

which is provided in line 5. Our program library then checks the integrability of transformed NLODE. The transformed NLODE is integrated one time, and the generated integrating constants are assigned with the values in our choice after running the program. After compiling and running the program, the generated output screen with our supplied values for the integrating constant is

```
The Diff. Equ. is integrable;
Do you assign a value to integrating constant (ic_1)? y
ic_1: 0
```

In the above output screen, the integrating constant (ic_1) has been assigned to 0 value. Next the A.E. in the form

$$F' = 1 + F + A_2 F^2 \tag{37}$$

has been provided by the command in line 6. The command in line 7 instructs the library to solve the nonlinear algebraic system also for the variable A_2 here. Here positive and negative, both parts are retained in the solutions using the definitions in lines 8 and 9. In line 10, we have not supplied the parameters p, q appearing in input-NLPDE (35). The command in line 12 solves *pde* by the modified F-expansion method. The solutions with the calculating steps are saved in *KdVB_mF.txt* file.

13. COMPLEX AND INTEGRABLE NLPDE CASE

In the example below, we demonstrate how the GiNaCDE handles a complex and integrable NLPDE. We consider the perturbed NLS equation with Kerr law nonlinearity

$$Iu_t + u_{2x} + A|u|^2 u + I\left(G_1 u_{3x} + G_2|u|^2 u_x + G_3\left(|u|^2\right)_x u\right) = 0, \tag{38}$$

where $u(t,x)$ represents the complex function and the parameters G_1, G_2 and G_3 are the higher order dispersion coefficient, the coefficient of Raman scattering, the coefficient of nonlinear dispersion term respectively, while A represents fiber loss. The model equation (38) has important application in various fields, such as semiconductor materials, optical fiber communications, plasma physics, fluid and solid mechanics.

We run GiNaCDE software applying all the three available methods on Eq. (38). Here we have used the following initializations:

F-expansion:	**mF-expansion:**	**FIM:**

`twcPhase=`
`lst{lst{k_0,k_1},lst{p_0,p_1}};`
`degAcoeff=`
`lst{4,0,0,A_2,A_3,A_4};`
`ASolve=true;`
`positivePart=true;`
`negativePart=true;`
`paraInDiffSolve=lst{};`
`filename="kerrNLS_Fexp.txt";`
	`twcPhase=`
	`lst{lst{k_0,k_1},lst{p_0,p_1}};`
	`degAcoeff=lst{2,A_0,A_1,A_2};`
	`ASolve=false;`
	`positivePart=true;`
	`negativePart=true;`
	`paraInDiffSolve=lst{};`
	`filename="kerrNLS_mF.txt";`
		`twcPhase=`
		`lst{lst{k_0,k_1},lst{p_0,p_1}};`
		`paraInDiffSolve=lst{};`
		`filename="kerrNLS_FIM.txt";`

The complete C++ code is given in the examples/ folder of the GiNaCDE repository with a file name *kerrNLS.cpp*.

In all methods, the software substitutes the traveling-wave solution (33) into Eq. (38), and separates the real part and the imaginary part following the step F1. The imaginary part is integrated one time and the software assigns the constant of integration to zero. Then the algebraic expressions of real part and imaginary part are compared and the software detects that they are same equations subject to the following conditions

$$\frac{k_0 + 2p_1k_1 - 3G_1k_1p_1^2}{-p_1^2 - p_0 + G_1p_1^3} = \frac{G_1k_1^3}{-3G_1p_1k_1^2 + k_1^2} = \frac{2k_1G_3 + k_1G_2}{3A - 3p_1G_2}. \tag{39}$$

Same conditions were obtained in [10]. Therefore GiNaCDE evaluates the exact analytical solutions of imaginary part only. We get exact solutions of the NLPDE (38) with the F-expansion and mF-expansion methods. FIM is unable to solve the NLPDE (38), because the NLPDE (38) in terms of traveling-wave coordinate ξ is not a second-order NLODE.

14. NON-POLYNOMIAL CASE IN FIM

In the following example, we show how GiNaCDE handles the non-polynomial case in FIM. We consider the following well-known Kudryashov–Sinelshchikov equation:

$$u_{3x} + guu_x - nu_{2x} - \left(uu_{2x} + u_x^2\right)d - ku_xu_{2x} - e\left(uu_{3x} + u_xu_{2x}\right) + u_t = 0, \tag{40}$$

where g, n, k, d and e are real parameters. Equation (40) models the pressure waves in a liquid and gas bubbles mixture when the viscosity of liquid and the heat transfer are both considered. We have employed the first integral method (FIM) to find exact traveling-wave solutions of Eq. (40) with the help of GiNaCDE using the following initializations:

```
//KS.cpp
#include <GiNaCDE/GiNaCDE.h>

int main()
{
   const ex u=reader("u"), t=reader("t"), x=reader("x"), k_0=reader("k_0"), k_1=reader("k_1"),
   A_0=reader("A_0"), A_1=reader("A_1"), A_2=reader("A_2"), A_3=reader("A_3"), g=reader("g"),
   n=reader("n"), d=reader("d"), e=reader("e"), k=reader("k");

   const ex pde = Diff(u,t,1)+g*u*Diff(u,x,1)+Diff(u,x,3)-e*Diff(u*Diff(u,x,2),x,1)
   -k*Diff(u,x,1)*Diff(u,x,2)-n*Diff(u,x,2)-d*Diff(u*Diff(u,x,1),x,1);

   depend(u, {t, x});

   twcPhase=lst{lst{k_0,k_1},lst{0,0}};
   paraInDiffSolve=lst{};
   filename="KS_FIM.txt";
   desolve(pde, {u}, FIM);

   return 0;

}
```

GiNaCDE makes the traveling-wave transformation (36) on the Eq. (40) and then it integrates the transformed NLPDE one time. We assign the integrating constant ic_1 to 0. The software automatically detects that $h(X)$ is not polynomial in X. To avoid such non-polynomial form of $h(X)$, we have implemented the procedure explained in [9] in step FIM4 of our algorithm. Following step FIM4, GiNaCDE performs a transformation to avoid singularity temporarily and the corresponding part of output where GiNaCDE makes the transformation is:

```
We make the transformation, d xi = (-1+e*X_)*d eta to avoid singularity -1+e*X_ = 0 temporarily.
Let U = X_, Diff(U,eta, 1) = Y_*(-1+e*X_), then we get
Diff(X_,eta, 1) = Y_*(-1+e*X_),
Diff(Y_,eta, 1) = -1/2*k*Y_^2-(d*k_1^(-1)*X_+n*k_1^(-1))*Y_+1/2*g*k_1^(-2)*X_^2
+k_0*k_1^(-3)*X_,
```

After that transformation, assuming $a_1 = 1$, the software evaluates $h = -\frac{k}{2}$. Then the degrees of a_0, g are auto-evaluated following the strategy in step FIM5 and it finds two sets of balanced degrees which are $deg(a_0, g) = (2, 0), (1, 1)$. Second set of balanced degrees was obtained in [9].

15. GUI OF THE LIBRARY

In sec. 7 we have mentioned that the source code can be compiled using the option *-DGINACDE_GUI_BUILD=on* to build GiNaCDE GUI. The GTK+3 library is required to build a GUI version of the GiNaCDE library. GiNaCDE GUI can interact graphically with us more easily without any programming knowledge, and there does not need for compilation each time which saves time. When we solve differential equations using GiNaCDE GUI, this GUI version guides us in each step to obtain the output results. For more details about the GiNaCDE GUI, we refer to the tutorial *GiNaCDE_guiTutorial.pdf*, which is available with pre-compiled windows binary here.

Appendix A: Solutions of Riccati equation

In case of Riccati equation, Eq. (15) take the form

$$F'(\xi) = A_0 + A_1 F + A_2 F^2. \tag{A-1}$$

The solutions of the equation (A-1) are [19]

$$F(\xi) = -\frac{A_1}{2A_2} - \frac{S}{2A_2}\tanh\left(\frac{S}{2}\xi + C\right) \quad (\text{If } A_2 \neq 0 \text{ and } A_1 \text{or} A_0 \neq 0), \tag{A-2a}$$

$$F(\xi) = -\frac{A_1}{2A_2} - \frac{S}{2A_2}\coth\left(\frac{S}{2}\xi + C\right) \quad (\text{If } A_2 \neq 0 \text{ and } A_1 \text{or} A_0 \neq 0), \tag{A-2b}$$

$$F(\xi) = \left(-\frac{A_1}{2A_0} + \frac{S}{2A_0}\tanh\left(\frac{S}{2}\xi + C\right)\right)^{-1} \quad (\text{If } A_0 \neq 0 \text{ and } A_1 \text{or} A_2 \neq 0), \tag{A-2c}$$

$$F(\xi) = \left(-\frac{A_1}{2A_0} + \frac{S}{2A_0}\coth\left(\frac{S}{2}\xi + C\right)\right)^{-1} \quad (\text{If } A_0 \neq 0 \text{ and } A_1 \text{or} A_2 \neq 0), \tag{A-2d}$$

$$F(\xi) = -\frac{A_1}{2A_2} - \frac{S}{2A_2}\tanh\left(\frac{S\xi}{2} + C\right) + \frac{\operatorname{sech}\left(\frac{S\xi}{2} + C\right)}{C\cosh\left(\frac{S\xi}{2} + C\right) - \frac{2A_2}{S}\sinh\left(\frac{S\xi}{2} + C\right)} \tag{A-2e}$$

$$(\text{If } A_2 \neq 0 \text{ and } A_1 \text{or} A_0 \neq 0),$$

$$F(\xi) = -\frac{A_0}{A_1} + Ce^{A_1\xi}, \quad (\text{If } A_2 = 0 \text{ and } A_1 \neq 0), \tag{A-2f}$$

$$F(\xi) = A_0\xi + C \quad (\text{If } A_2 = A_1 = 0 \text{ and } A_0 \neq 0), \tag{A-2g}$$

where $S = \sqrt{A_1^2 - 4A_0A_2}$ and C is auxiliary constant. In the above solutions from Eq. (A-2a) to (A-2e), the condition $A_1^2 - 4A_0A_2 > 0$ must be satisfied. In the following solutions if not mentioned C has to be assumed as an auxiliary constant.

Appendix B: Solutions of Bernoulli equation

In case of Bernoulli equation, Eq. (15) is reduced to

$$F'(\xi) = A_1F + A_\delta F^\delta. \tag{B-1}$$

The solutions of the equation (B-1) are [19]

$$F(\xi) = \left(\frac{A_1\left(\cosh\left(A_1(\delta-1)\xi + CA_1\right) + \sinh\left(A_1(\delta-1)\xi + CA_1\right)\right)}{1 - A_\delta\cosh\left(A_1(\delta-1)\xi + CA_1\right) - A_\delta\sinh\left(A_1(\delta-1)\xi + CA_1\right)}\right)^{(\delta-1)^{-1}} \quad (\text{If } A_1 \neq 0 \text{ and } \delta \neq 1), \tag{B-2a}$$

$$F(\xi) = \left(-\frac{A_\delta}{A_1} + Ce^{A_1(1-\delta)\xi}\right)^{\frac{1}{1-\delta}} \quad (\text{If } A_1 \neq 0 \text{ and } \delta \neq 1), \tag{B-2b}$$

$$F(\xi) = \left(-\frac{A_1}{2A_\delta} - \frac{A_1}{2A_\delta}\tanh\left(\frac{(\delta-1)A_1}{2}\xi + C\right)\right)^{\frac{1}{\delta-1}} \quad (\text{If } A_1 \neq 0 \text{ and } \delta \neq 1), \tag{B-2c}$$

$$F(\xi) = \left(-\frac{A_1}{2A_\delta} - \frac{A_1}{2A_\delta}\coth\left(\frac{(\delta-1)A_1}{2}\xi + C\right)\right)^{\frac{1}{\delta-1}} \quad (\text{If } A_1 \neq 0 \text{ and } \delta \neq 1), \tag{B-2d}$$

$$F(\xi) = \left(\left(A_\delta\xi(1-\delta) + C\right)^{(\delta-1)^{-1}}\right)^{-1} \quad (\text{If } A_1 = 0 \text{ and } \delta \neq 1)), \tag{B-2e}$$

$$F(\xi) = Ce^{(A_1+A_\delta)\xi} \quad (\text{If } \delta = 1). \tag{B-2f}$$

Appendix C: first-order NLODEs related to Jacobi Elliptic Functions

The incomplete elliptic integral of the first kind, is defined by

$$u(\phi, m) = \int_0^\phi \frac{d\theta}{\sqrt{1 - m^2\sin^2\theta}}, \tag{C-1}$$

where m is the elliptic modulus, and $\phi = \text{JacobiAM}(u, m)$ is the Jacobi amplitude which is the inverse of elliptic integral (C-1). The three principal elliptic functions are denoted $\text{JacobiSN}(u, m), \text{JacobiCN}(u, m), \text{JacobiDN}(u, m)$, which are in turn defined in terms of the amplitude function JacobiAM satisfying

$$\text{JacobiSN}(u, m) = \sin\left(\text{JacobiAM}(u, m)\right), \tag{C-2}$$

$$\text{JacobiCN}(u,m) = \cos\left(\text{JacobiAM}(u,m)\right), \tag{C-3}$$

$$\text{JacobiDN}(u,m) = \frac{\partial}{\partial u}\text{JacobiAM}(u,m) = \sqrt{1-m^2\text{JacobiSN}(u,m)^2}. \tag{C-4}$$

There are total twelve Jacobian functions that can be expressed in general by a name JacobiXY which follows the identities $\text{JacobiXY} = \frac{1}{\text{JacobiYX}} = \frac{\text{JacobiPR}}{\text{JacobiQR}}$. Here X, Y, R are any three of S, C, N, D. Following these rules of notations other nine subsidiary Jacobian elliptic functions can be defined in terms of the three JacobiSN, JacobiCN, JacobiDN by the following identities:

$$\begin{aligned}
&\text{JacobiNS}(u,m) = (\text{JacobiSN}(u,m))^{-1}, \text{JacobiND}(u,m) = (\text{JacobiDN}(u,m))^{-1}, \text{JacobiSC}(u,m) = (\text{JacobiCS}(u,m))^{-1},\\
&\text{JacobiSD}(u,m) = (\text{JacobiDS}(u,m))^{-1}, \text{JacobiDC}(u,m) = (\text{JacobiCD}(u,m))^{-1}, \text{JacobiNC}(u,m) = (\text{JacobiCN}(u,m))^{-1},\\
&\text{JacobiCS}(u,m) = \frac{\text{JacobiCN}(u,m)}{\text{JacobiSN}(u,m)}, \text{JacobiDS}(u,m) = \frac{\text{JacobiDN}(u,m)}{\text{JacobiSN}(u,m)}, \text{JacobiCD}(u,m) = \frac{\text{JacobiCN}(u,m)}{\text{JacobiDN}(u,m)}.
\end{aligned} \tag{C-5}$$

Jacobian elliptic functions can also be defined as solutions to the differential equations [1, 2, 20]

$$F'(\xi) = \sqrt{A_0 + A_2F^2 + A_4F^4}. \tag{C-6}$$

Solutions of Eq. (C-6) in terms of Jacobi elliptic functions are listed in below (with the conditions that all the algebraic expressions within square-root must be greater than 0):

$$F(\xi) = \frac{\sqrt{2A_2A_4\left(-{A_2}^2 + \sqrt{{A_2}^2S}\right)}}{2A_2A_4}\text{JacobiSN}\left(\frac{\sqrt{2A_2\left(-{A_2}^2 - \sqrt{{A_2}^2S}\right)}\xi}{2A_2}, \frac{\sqrt{2A_0A_4\left(-2A_0A_4 + {A_2}^2 - \sqrt{{A_2}^2S}\right)}}{2A_0A_4}\right), \tag{C-7a}$$

$$F(\xi) = \frac{\sqrt{2A_2A_4\left(-{A_2}^2 + \sqrt{{A_2}^2S}\right)}}{2A_2A_4}\text{JacobiCN}\left(\frac{\sqrt{-A_2\sqrt{{A_2}^2S}}\xi}{A_2}, \frac{\sqrt{-2S\left(\sqrt{{A_2}^2S} - S\right)}}{2S}\right), \tag{C-7b}$$

$$F(\xi) = \frac{\sqrt{2A_2A_4\left(-{A_2}^2 - \sqrt{{A_2}^2S}\right)}}{2A_2A_4}\text{JacobiDN}\left(\frac{\sqrt{2A_2\left({A_2}^2 + \sqrt{{A_2}^2S}\right)}\xi}{2A_2}, \frac{\sqrt{2A_0A_4\left(\sqrt{{A_2}^2S} - S\right)}}{2A_0A_4}\right), \tag{C-7c}$$

$$F(\xi) = \frac{\sqrt{2A_4\left(-A_2 + \sqrt{S}\right)}}{2A_4}\text{JacobiNS}\left(1/2\sqrt{-2A_2 + 2\sqrt{S}}\xi, \frac{\sqrt{2A_0A_4\left(A_2\sqrt{S} - 2A_0A_4 + {A_2}^2\right)}}{2A_0A_4}\right), \tag{C-7d}$$

$$F(\xi) = \frac{\sqrt{-2A_2 - 2\sqrt{S}}}{2\sqrt{A_4}}\text{JacobiNC}\left(\sqrt{-\sqrt{S}}\xi, \frac{\sqrt{-2A_0A_4}}{\sqrt{A_2\sqrt{S} + S}}\right), \tag{C-7e}$$

$$F(\xi) = \frac{\sqrt{-2A_2 + 2\sqrt{S}}}{2\sqrt{A_4}}\text{JacobiND}\left(\frac{\sqrt{-2A_0A_4}\xi}{\sqrt{-A_2 + \sqrt{S}}}, \frac{\sqrt{2A_2\sqrt{S} - 2S}}{2\sqrt{A_0A_4}}\right), \tag{C-7f}$$

$$F(\xi) = -\frac{\sqrt{2}\sqrt{A_0}}{\sqrt{A_2 + \sqrt{S}}}\text{JacobiSC}\left(\frac{\sqrt{A_2\sqrt{S} - 2A_0A_4 + {A_2}^2}\xi}{\sqrt{A_2 + \sqrt{S}}}, \frac{\sqrt{A_2\sqrt{S} + S}}{\sqrt{A_2\sqrt{S} - 2A_0A_4 + {A_2}^2}}\right), \tag{C-7g}$$

$$F(\xi) = \frac{\sqrt{2}\sqrt{A_0}\sqrt{-A_2\sqrt{S} + 2A_0A_4 - {A_2}^2}}{\sqrt{A_2 + \sqrt{S}}\sqrt{-A_2\sqrt{S} - S}}\text{JacobiSD}\left(\frac{\sqrt{A_2\sqrt{S} + S}\xi}{\sqrt{A_2 + \sqrt{S}}}, \frac{\sqrt{-A_2\sqrt{S} + 2A_0A_4 - {A_2}^2}}{\sqrt{-A_2\sqrt{S} - S}}\right), \tag{C-7h}$$

$$F(\xi) = \frac{\sqrt{2}\sqrt{A_0}}{\sqrt{A_2 + \sqrt{S}}}\text{JacobiCS}\left(\frac{\sqrt{2A_0A_4}\xi}{\sqrt{A_2 + \sqrt{S}}}, \frac{\sqrt{-A_2\sqrt{S} - S}}{\sqrt{2A_0A_4}}\right), \tag{C-7i}$$

$$F(\xi) = \frac{\sqrt{A_2\sqrt{S} - S}}{\sqrt{A_2A_4 - A_4\sqrt{S}}}\text{JacobiDS}\left(\frac{\sqrt{A_2\sqrt{S} - S}\xi}{\sqrt{A_2 - \sqrt{S}}}, \frac{\sqrt{2A_0A_4}}{\sqrt{A_2\sqrt{S} - S}}\right), \tag{C-7j}$$

where $S = \left(-4A_0A_4 + {A_2}^2\right)$.

The solutions of an another first-order NLODE [3, 20]

$$F'(\xi) = \sqrt{A_1F + A_2F^2 + A_3F^3}, \tag{C-8}$$

can also be expressed in terms of Jacobi elliptic functions. The solutions of Eq. (C-8) are (with the conditions that all the algebraic expressions within square-root must be greater than 0)

$$F(\xi) = \frac{-A_2 + \sqrt{S}}{2A_3}\text{JacobiSN}^2\left(\frac{\sqrt{2A_1A_3}\xi}{2\sqrt{-A_2 + \sqrt{S}}}, \frac{\sqrt{{A_2}^2 - A_2\sqrt{S} - 2A_1A_3}}{\sqrt{2A_1A_3}}\right), \tag{C-9a}$$

$$F(\xi) = -\frac{2A_1}{A_2 - \sqrt{S}}\text{JacobiCN}^2\left(\frac{\sqrt[4]{S}\xi}{2}, \frac{\sqrt{A_2 + \sqrt{S}}}{\sqrt{2}\sqrt[4]{S}}\right), \tag{C-9b}$$

$$F(\xi) = -\frac{2A_1}{A_2 - \sqrt{S}}\text{JacobiDN}^2\left(\frac{\sqrt{2A_1A_3}\xi}{2\sqrt{A_2 - \sqrt{S}}}, \frac{\sqrt{4A_1A_3 + \sqrt{S}A_2 - {A_2}^2}}{\sqrt{2A_1A_3}}\right), \tag{C-9c}$$

$$F(\xi) = -\frac{2A_1}{A_2 + \sqrt{S}}\text{JacobiNS}^2\left(\frac{\sqrt{2}}{4}\sqrt{-A_2 + \sqrt{S}}\xi, \frac{\sqrt{A_2 + \sqrt{S}}}{\sqrt{A_2 - \sqrt{S}}}\right), \tag{C-9d}$$

$$F(\xi) = -\frac{2A_1}{A_2 + \sqrt{S}}\text{JacobiNC}^2\left(\frac{\sqrt[4]{S}\xi}{2}, \frac{\sqrt{A_2 + \sqrt{S}}}{\sqrt{2}\sqrt[4]{S}}\right), \tag{C-9e}$$

$$F(\xi) = -\frac{2A_1}{A_2 + \sqrt{S}}\text{JacobiND}^2\left(\frac{\sqrt{2}\sqrt{A_2 + \sqrt{S}}\xi}{4}, \frac{\sqrt{2}\sqrt[4]{S}}{\sqrt{h_2 + \sqrt{S}}}\right), \tag{C-9f}$$

$$F(\xi) = \frac{2A_1}{A_2 + \sqrt{S}}\text{JacobiSC}^2\left(\frac{\sqrt{\sqrt{S}A_2 - 2A_1A_3 + {A_2}^2}\xi}{2\sqrt{A_2 + \sqrt{S}}}, \frac{\sqrt{\sqrt{S}A_2 + S}}{\sqrt{\sqrt{S}A_2 - 2A_1A_3 + {A_2}^2}}\right), \tag{C-9g}$$

$$F(\xi) = \frac{2A_1\left(-2A_1A_3 + \sqrt{S}A_2 + {A_2}^2\right)}{\left(A_2 + \sqrt{S}\right)\left(\sqrt{S}A_2 + S\right)}\text{JacobiSD}^2\left(\frac{\sqrt{\sqrt{S}A_2 + S}\xi}{2\sqrt{A_2 + \sqrt{S}}}, \frac{\sqrt{2A_1A_3 - \sqrt{S}A_2 - {A_2}^2}}{\sqrt{-\sqrt{S}A_2 - S}}\right), \tag{C-9h}$$

$$F(\xi) = \frac{2A_1}{A_2 - \sqrt{S}}\text{JacobiCS}^2\left(\frac{\sqrt{A_1A_3}\xi}{\sqrt{2}\sqrt{A_2 - \sqrt{S}}}, \frac{\sqrt{\sqrt{S}A_2 - S}}{\sqrt{2A_1A_3}}\right), \tag{C-9i}$$

$$F(\xi) = \frac{-\sqrt{S}A_2 - S}{A_3\left(A_2 + \sqrt{S}\right)}\text{JacobiDS}^2\left(\frac{\sqrt{-\sqrt{S}A_2 - S}\xi}{2\sqrt{A_2 + \sqrt{S}}}, \frac{\sqrt{2A_1A_3}}{\sqrt{-\sqrt{S}A_2 - S}}\right), \tag{C-9j}$$

where

$$S = -4A_1A_3 + {A_2}^2. \tag{C-10}$$

Appendix D: Solutions of some more different types of first-order NLODEs

In this section, we discuss the solutions of more different types of the first-order NLODE. Among them, we can choose an auxiliary equation suitable for input-NLPDE.

Type-1 first-order NLODE:

Let us first consider the first-order NLODE as follows

$$F'(\xi) = \sqrt{A_0 + A_2F^2 + A_4F^4 + A_6F^6}. \tag{D-1}$$

The above equation admits following special hyperbolic solutions [4]:

If $A_0 = \frac{8A_2^2}{27A_4}$ and $A_6 = \frac{A_4^2}{4A_2}$ then it has a bell profile solution

$$F(\xi) = \left(-\frac{8A_2\left(\tanh\left(\sqrt{-\frac{A_2}{3}}\xi + C\right)\right)^2}{3A_4\left(3 + \left(\tanh\left(\sqrt{-\frac{A_2}{3}}\xi + C\right)\right)^2\right)}\right)^{\frac{1}{2}}, \tag{D-2a}$$

and a singular solution

$$F(\xi)=\left(-\frac{8A_2\left(\coth\left(\sqrt{-\frac{A_2}{3}}\xi+C\right)\right)^2}{3A_4\left(3+\left(\coth\left(\sqrt{-\frac{A_2}{3}}\xi+C\right)\right)^2\right)}\right)^{\frac{1}{2}}. \tag{D-2b}$$

Type-2 first-order NLODE:

For $A_0=0$ Eq. (D-1) is reduced to

$$F'(\xi)=\sqrt{A_2F^2+A_4F^4+A_6F^6}. \tag{D-3}$$

Equation (D-3) has a triangular periodic solution [4]

$$F(\xi)=\left(\frac{2A_2\text{sech}^2(\sqrt{A_2}\xi+C)}{2\sqrt{A_4^2-4A_2A_6}-\left(\sqrt{A_4^2-4A_2A_6}+A_4\right)\text{sech}^2(\sqrt{A_2}\xi+C)}\right)^{\frac{1}{2}}, \tag{D-4a}$$

and a singular triangular periodic solution

$$F(\xi)=\left(\frac{2A_2\text{csch}^2(\sqrt{A_2}\xi+C)}{2\sqrt{A_4^2-4A_2A_6}+\left(\sqrt{A_4^2-4A_2A_6}-A_4\right)\text{csch}^2(\sqrt{A_2}\xi+C)}\right)^{\frac{1}{2}}. \tag{D-4b}$$

If $A_6=\frac{A_4^2}{4A_2}$, Eq. (D-3) also admits a kink profile solution

$$F(\xi)=\left(-\frac{A_2}{A_4}\left(1+\tanh\left(\sqrt{A_2}\xi+C\right)\right)\right)^{\frac{1}{2}}, \tag{D-5a}$$

and a singular solution

$$F(\xi)=\left(-\frac{A_2}{A_4}\left(1+\coth\left(\sqrt{A_2}\xi+C\right)\right)\right)^{\frac{1}{2}}. \tag{D-5b}$$

When $A_4=0$ Eq. (D-3) is reduced to

$$F'(\xi)=\sqrt{A_2F^2+A_6F^6}. \tag{D-6}$$

It is clear that the Eqs. (D-4a), (D-4b) with $A_4=0$ are also solutions of Eq. (D-6), but Eqs. (D-5a), (D-5b) are not solutions of Eq. (D-6) as they are undefined at $A_4=0$.

Type-3 first-order NLODE:

An another type of first-order NLODE is

$$F'(\xi)=\sqrt{A_2F^2+A_3F^3+A_4F^4}, \tag{D-7}$$

which has different type solitary wave solutions [5]

$$F(\xi)=-\frac{A_2A_3\left(\text{sech}\left(\frac{\sqrt{A_2}}{2}\xi\right)\right)^2}{{A_3}^2-A_2A_4\left(1-\tanh\left(\frac{\sqrt{A_2}}{2}\xi\right)\right)^2},\ F(\xi)=\frac{2A_2\text{sech}\left(\sqrt{A_2}\xi\right)}{\sqrt{-4A_2A_4+{A_3}^2}-A_3\text{sech}\left(\sqrt{A_2}\xi\right)}. \tag{D-8a}$$

For $A_4=0$ Eq. (D-7) is reduced to

$$F'(\xi)=\sqrt{A_2F^2+A_3F^3}. \tag{D-9}$$

One can get the solutions of above equations substituting $A_4 = 0$ in solutions (D-8). If $A_3 = 0$ Eq. (D-7) is simplified to

$$F'(\xi) = \sqrt{A_2F^2 + A_4F^4}. \tag{D-10}$$

Solutions (D-8) with $A_3 = 0$ also exist for (D-10). Beside these solutions Eq. (D-10) has following two extra solutions:

$$F(\xi) = \frac{4A_2\mathrm{e}^{(\xi+C)\sqrt{A_2}}}{-4A_2A_4\mathrm{e}^{2\sqrt{A_2}\xi} + \mathrm{e}^{2C\sqrt{A_2}}},\ F(\xi) = \frac{4A_2\mathrm{e}^{(\xi+C)\sqrt{A_2}}}{-4A_2A_4\mathrm{e}^{2C\sqrt{A_2}} + \mathrm{e}^{2\sqrt{A_2}\xi}}. \tag{D-11a}$$

Type-4 first-order NLODE:

We will now consider one more simplified first-order NLODE

$$F'(\xi) = \sqrt{A_0 + A_2F^2}. \tag{D-12}$$

We find some exact solutions of Eq. (D-12) containing exponential, hyperbolic functions, which are listed below

$$F(\xi) = \frac{\left(-A_0\mathrm{e}^{2\xi\sqrt{A_2}} + \mathrm{e}^{2C\sqrt{A_2}}\right)\mathrm{e}^{-(C+\xi)\sqrt{A_2}}}{2\sqrt{A_2}},\ F(\xi) = \frac{\left(-\mathrm{e}^{2\xi\sqrt{A_2}} + A_0\mathrm{e}^{2C\sqrt{A_2}}\right)\mathrm{e}^{-(C+\xi)\sqrt{A_2}}}{2\sqrt{A_2}}, \tag{D-13a}$$

$$F(\xi) = \pm\sqrt{\frac{-A_0}{A_2}}\cosh\left(\xi\sqrt{A_2} + C\right),\ F(\xi) = \pm\sqrt{\frac{A_0}{A_2}}\sinh\left(\xi\sqrt{A_2} + C\right). \tag{D-13b}$$

[1] Y. Zhou, M. Wang, and Y. Wang, Periodic wave solutions to a coupled KdV equations with variable coefficients. Physics Letters A. 308 (2003) 31–36.

[2] A.H. BHRAWY, M.A. ABDELKAWY, S. KUMAR and A. BISWAS, SOLITONS AND OTHER SOLUTIONS TO KADOMTSEV-PETVIASHVILI EQUATION OF B-TYPE, Rom. Journ. Phys. 58, (2013) 729–748.

[3] Y. He, New Jacobi Elliptic Function Solutions for the Kudryashov-Sinelshchikov Equation Using Improved F-Expansion Method, Mathematical Problems in Engineering, 2013, (2013) 104894.

[4] D.J. Huang, D.S. Li, H.G. Zhang, Explicit and exact travelling wave solutions for the generalized derivative Schrödinger equation. Chaos, Solitons and Fractals. 31 (2007) 586–593.

[5] Sirendaoreji, New exact travelling wave solutions for the Kawahara and modified Kawahara equations. Chaos, Solitons and Fractals 19 (2004) 147–150.

[6] G. Cai, Q. Wang, J, Huang, A Modified F-expansion Method for Solving Breaking Soliton Equation. International Journal of Nonlinear Science. 2 (2006) 122.

[7] Z. Feng, On Explicit Exact Solutions to the Compound Burgers-KdV Equation. Physics Letters A. 293 (2002) 57-66. doi:10.1016/S0375-9601(01)00825-8.

[8] Z. Feng, The first integer method to study the Burgers-Korteweg-de Vries equation. Journal of Physics A: Mathematical and General. 35 (2002) 343-349.

[9] M. Mirzazadeh, M. Eslami, Exact solutions of the Kudryashov–Sinelshchikov equation and nonlinear telegraph equation via the first integral method. Nonlinear Analysis: Modelling and Control. 17 (2012) 481–488.

[10] Z.Y. Zhang, Z.H. Liu, X.J. Miao, Y.Z. Chen, New exact solutions to the perturbed nonlinear Schrödinger's equation with Kerr law nonlinearity, Applied Mathematics and Computation. 216 (2010) 3064-3072.

[11] C. W. Bauer, A. Frink and R. Kreckel, Introduction to the GiNaC Framework for Symbolic Computation within the C++ Programming Language. arXiv:cs/0004015; the GiNaC library is available at http://www.ginac.de.

[12] N. Taghizadeh, M. Mirzazadeh, F. Farahrooz, Exact solutions of the nonlinear Schrödinger equation by the first integral method. In Journal of Mathematical Analysis and Applications. 374 (2011) 549-553. https://doi.org/10.1016/j.jmaa.2010.08.050

[13] Zhi-Bin Li & Yin-Ping Liu, RATH: A Maple package for finding travelling solitary wave solutions to nonlinear evolution equations. Comput. Phys. Comm. 148 (2002) 256-266.

[14] T. Becker, V. Weispfenning, Gröbner Bases: A Computational Approach to Commutative Algebra. Springer-Verlag, Berlin, 1993.

[15] D. Wang, Elimination Methods. Springer-Verlag, New York, 2001.

[16] D. Wang. A generalized algorithm for computing characteristic sets. World Scientific Publishing Company, Singapore. 165–174 (2001).

[17] A. Wittkopf, G. Reid, Introduction to the Rif Package Version 1.1. See: http://www.cecm.sfu.ca/ wittkopf/Rif.html, 2003.

[18] D. Baldwin, Ü. Göktaş, W. Hereman, Symbolic computation of hyperbolic tangent solutions for nonlinear differential–difference equations. Comput. Phys. Comm. 162 (2004) 203–217.

[19] X. F. Yang, Z. C. Deng, Y. Wei. Riccati-Bernoulli sub-ODE method for nonlinear partial differential equations and its application. Advances in Difference Equations. 2015 (2015) 117.
[20] W. A. Schwalm, Lectures on Selected Topics in Mathematical Physics: Elliptic Functions and Elliptic Integrals; Morgan & Claypool Publishers, (2015).