Research Article ISSN Online: 2783-4859 Computational Mathematics and Computer Modeling with Applications

Received 4 September 2026
Accepted 23 October 2026

DOI: 10.48308/CMCMA.5.1.26
AMS Subject Classification: 34A34

Implementation of an ODE Solver by Optimal Quadratization and Probabilistic Evolution Theory

Coşar Gözükırmızı[a]

Probabilistic evolution theory (PREVTH) solves a system of ODEs with purely second degree polynomial right-hand sides (each additive term of each of the right-hand sides is exactly second degree). Quadratization takes a system of ODEs with polynomial right-hand sides and makes it into an equivalent system of ODEs with more equations and unknowns such that the new system has right-hand sides which are at most second degree polynomials. Quadratization may also be used as polynomialization + quadratization so that the method is applicable to more problems. Polynomialization can get rid of negative integer powers, and in many cases, other function families such as trigonometric and exponential. Going from at most second degree terms on the right-hand sides to exactly (purely) second degree terms on the right-hand sides is always possible by constancy adding space extension (CASE). Therefore polynomialization + quadratization + CASE + PREVTH is a highly efficient sequence of steps to provide a solution for many ODE systems. Since PREVTH sums the Taylor terms of the solution, polynomialization + quadratization + CASE + PREVTH is especially effective for solutions in small t intervals. Up to now, the quadratization within the framework of PREVTH was either done manually or by a naive algorithm that does not provide optimal quadratization. Optimal quadratization (smallest number of equations and unknowns) is important since having redundancy in the equations greatly increases the necessary computational effort. In this work, an ODE solver is implemented that successfully combines optimal quadratization and PREVTH. Also, it is important to note that the Taylor series representation of the solution by PREVTH is exact. The convergence of the Taylor series depends on the system. The paper also focuses on different system of ODEs to show how the ODE solver can provide a Taylor series solution in powers of t, from which a numerical result can be obtained by substituting t with the desired value. **Copyright © 2026 Shahid Beheshti University.**

Keywords: Quadratization; Probabilistic evolution theory.

1. Introduction

This work combines polynomialization + quadratization with constancy adding space extension (CASE) + probabilistic evolution theory (PREVTH). A novel ODE solver that can be applied to a very large variety of explicit first order ODE systems is implemented. The solver is especially powerful in small t intervals.

The works related to PREVTH fall under two main categories: PREVTH for the solution of ordinary differential equations and PREVTH for the solution of quantum systems through expectation value dynamics. In this work, we focus on the solution of ordinary differential equations.

[9] is an extensive review of probabilistic evolution theory. [10] shows a quick and easy quadratization method for PREVTH applications. [11] shows the recent status of probabilistic evolution theory for the solution of ODEs and how it is related to quadratization. [14] details the recursion for the coefficients of the Taylor expansion of the solution and provides its proof. This recursion is vital in making probabilistic evolution theory an efficient method. [15] details an algorithm for optimal quadratization (optimal space extension). [17] shows the status of probabilistic evolution theory before the recursion and condensed Kronecker product operation was proposed. At that time, a squarification operation was used to avoid very rapid increase in size of the vectors as more terms from the Taylor series are computed. [18, 16, 17] are early applications. [12, 13] lay out the framework

[a] *Department of Computer Engineering, Istanbul Beykent University, Istanbul, Turkey.*
[*] *Correspondence to: C. Gözükırmızı Email: cosargozukirmizi@beykent.edu.tr*

 Copyright © 2026 Shahid Beheshti University.

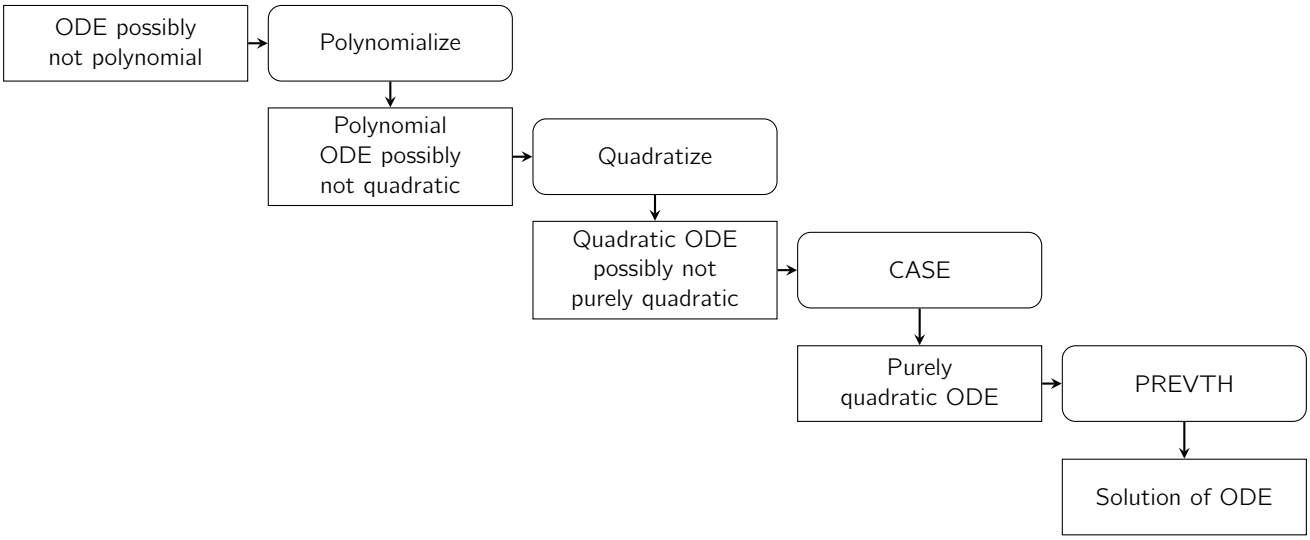

Figure 1. The proposed algorithm for the solution of system of explicit first order ordinary differential equations

for probabilistic evolution theory for solution of ODEs. [19] shows the design and implementation of a web interface connected to a database that stores the Taylor series coefficients of PREVTH for van der Pol ODE with different parameters and initial values.

[2] details optimal quadratization and provides information about the QBee software. [3] shows how to perform optimal quadratization so that the resulting ODE set shows certain characteristics. [1] details improvements in QBee software, including handling of non-autonomous ODEs.

[4, 6] present a novel method for the solution of fractional integro-differential equations using fractional Lagrange functions. [8] combines Generalized Pseudospectral Method with linearization to solve second order BVPs. [5] explains the Generalized Pseudospectral Method. [20] shows the solution of Thomas-Fermi equation using fractional order of rational Chebyshev functions, giving a series solution in integer and fraction powers of t. [7] shows the solution of a PDE by using rational Chebyshev function collocation method. For other references related to ODE solving, the reader may refer to the references cited in [11].

In this work, we have used the QBee software with Version 0.8.2 for polynomialization + quadratization which is currently available at the web address https://github.com/AndreyBychkov/QBee. This efficient software by Bychkov and Pogudin is a Python program.

Within the framework of this article, we have made a program in Python that performs constancy adding space extension and a C++ program that solves the resulting ODE by PREVTH. This whole software is named prevth and is available at https://gitlab.com/cosargozukirmizi/prevth/ and the documentation effort is ongoing. The software needs QBee to perform polynomialization + quadratization + CASE + PREVTH. Therefore, QBee needs to be installed to use prevth.

Figure 1 shows the proposed ODE solver. In the figure, the rectangles show the data and the rectangles with rounded corners show the operations. The four operations are polynomialize, quadratize, CASE and PREVTH. Here, polynomial ODE means a system of explicit first order ODEs where the right-hand sides are polynomials of the unknowns. Quadratic ODE is a polynomial ODE where the degrees of the right-hand sides are at most 2. Purely quadratic ODE is a quadratic ODE where each additive term for each of the right-hand sides has the degree 2, meaning constant additive terms and first degree additive terms are not allowed. In previous works of the author, the term multinomial is used in order to mean multivariate polynomial [15]; but in this work, polynomial is used in order to also mean multivariate polynomial.

This paper also provides certain trivial examples to familiarize the reader with the application of CASE + PREVTH.

2. Probabilistic evolution theory

This section shows the application of probabilistic evolution theory for the solution of initial value problem of system of ODEs with polynomial right-hand sides.

In probabilistic evolution theory (PREVTH), the condensed Kronecker product operation is used. The condensed Kronecker product operation is defined between two vectors of the same size. Consider the two vectors

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}, \tag{1}$$

 Copyright © 2026 Shahid Beheshti University.

where each vector has N elements.

Then, the condensed Kronecker product operation is defined by

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} \otimes_c \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 + a_2 b_1 \\ a_1 b_3 + a_3 b_1 \\ \vdots \\ a_1 b_N + a_N b_1 \\ a_2 b_2 \\ a_2 b_3 + a_3 b_2 \\ \vdots \\ \vdots \\ a_N b_N \end{bmatrix}, \tag{2}$$

where the resulting vector has $N(N+1)/2$ elements. For each element of $\mathbf{a}$, the $\mathbf{b}$ is considered from the current index in $\mathbf{a}$ up to N. For the same index i; $a_i b_i$ is put to the resulting vector. For different indices i and j; $a_i b_j + a_j b_i$ is put to the resulting vector. The condensed Kronecker product of vectors with three elements is

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes_c \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 + a_2 b_1 \\ a_1 b_3 + a_3 b_1 \\ a_2 b_2 \\ a_2 b_3 + a_3 b_2 \\ a_3 b_3 \end{bmatrix}, \tag{3}$$

forming a vector with six elements. The condensed Kronecker product of vectors with four elements is

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \otimes_c \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 + a_2 b_1 \\ a_1 b_3 + a_3 b_1 \\ a_1 b_4 + a_4 b_1 \\ a_2 b_2 \\ a_2 b_3 + a_3 b_2 \\ a_2 b_4 + a_4 b_2 \\ a_3 b_3 \\ a_3 b_4 + a_4 b_3 \\ a_4 b_4 \end{bmatrix}, \tag{4}$$

forming a vector with ten elements. The condensed Kronecker product is a commutative operation, therefore

$$\mathbf{a} \otimes_c \mathbf{b} = \mathbf{b} \otimes_c \mathbf{a} \tag{5}$$

holds. The algorithm for condensed Kronecker product is given below, using the convention where vector indexing starts from 0.

Algorithm 1: Condensed Kronecker Product

Input : Vectors a and b of size n
Output: Vector $result$ containing the condensed Kronecker product

```
myIndex ← 0;
for i ← 0 to n − 1 do
    result[myIndex] ← a[i] × b[i];
    myIndex ← myIndex + 1;
    for j ← (i + 1) to n − 1 do
        result[myIndex] ← (a[i] × b[j]) + (a[j] × b[i]);
        myIndex ← myIndex + 1;
```

It is also possible to take the condensed Kronecker product of a vector with itself. This operation is also called condensed Kronecker squaring. The notation and the definition of condensed Kronecker squaring is

$$\mathbf{a}^{\otimes_c^2} \equiv \mathbf{a} \otimes_c \mathbf{a}, \tag{6}$$

where it is also possible to say that condensed Kronecker squaring is condensed Kronecker powering with the value 2.

 Copyright © 2026 Shahid Beheshti University.

Probabilistic evolution theory solves the initial value problem of explicit ODE systems, where the right-hand sides are purely quadratic. This means that all the additive terms on the right-hand sides are second degree terms (first degree terms and constant terms are not allowed). The initial value problem of explicit ODE systems where the right-hand sides are purely quadratic can be shown by

$$\dot{\mathbf{y}}(t) = \mathbf{F}\left(\frac{1}{2}\mathbf{y}(t)^{\otimes_c^2}\right), \quad \mathbf{y}(0) = \boldsymbol{\xi}, \tag{7}$$

where the $\mathbf{F}$ is an $N \times (N(N+1)/2)$ matrix. N is the number of equations and therefore the number of unknowns. Finding the elements of $\mathbf{F}$ is straightforward. The elements of $\mathbf{F}$ are found by reading the coefficients of the right-hand sides, being careful about how the condensed Kronecker squaring works and the $1/2$ next to the Kronecker squared entity. Then the solution of the initial value problem is

$$\mathbf{y}(t) = \sum_{j=0}^{\infty} t^j \boldsymbol{\rho}_j, \quad \boldsymbol{\rho}_0 = \boldsymbol{\xi}, \tag{8}$$

where truncating the Taylor series on the right-hand side and plugging in values for t gives numerical results. The vectors in the Taylor series are computed through the recursion where, for the convenience of notation, the odd and even indices are separated. For even j

$$\boldsymbol{\rho}_{j+1} = \frac{1}{j+1}\mathbf{F}\left(\frac{1}{2}\left(\boldsymbol{\rho}_{\frac{j}{2}} \otimes_c \boldsymbol{\rho}_{\frac{j}{2}}\right) + \sum_{k=0}^{\frac{j}{2}-1} \boldsymbol{\rho}_k \otimes_c \boldsymbol{\rho}_{j-k}\right), \quad j = 0, 2, 4, \ldots \tag{9}$$

is used and for odd j

$$\boldsymbol{\rho}_{j+1} = \frac{1}{j+1}\mathbf{F}\left(\sum_{k=0}^{\frac{j-1}{2}} \boldsymbol{\rho}_k \otimes_c \boldsymbol{\rho}_{j-k}\right), \quad j = 1, 3, 5, \ldots \tag{10}$$

is used. The recursion only involves matrix and vector operations. The proof of (8), (9) and (10) is given in [14]. The convergence of the series in (8) is also studied in previous works. This approach is powerful in small t intervals.

Numerical results for different ODE systems is also given in previous works. In the next section, we discuss the solution of trivial ODEs using probabilistic evolution theory to provide hands-on experience with the method.

3. Trivial examples

3.1. The function whose derivative is 0

Consider the initial value problem given below.

$$\dot{p}(t) = 0, \quad p(0) = 1. \tag{11}$$

This is the simplest initial value problem. Although the right-hand side of (11) is not a second degree term, it can be considered as a second degree term since 0 is $0p(x)p(x)$. Therefore the right-hand side of (11) is purely quadratic with the coefficient as 0. The system vector is

$$\mathbf{y} = [p], \tag{12}$$

consisting of a single element. The target notation given in (7) is

$$\dot{\mathbf{y}}(t) = \mathbf{F}\left(\frac{1}{2}\mathbf{y}(t)^{\otimes_c^2}\right), \quad \mathbf{y}(0) = [1] \tag{13}$$

where the condensed Kronecker square of a vector with single element is the vector with the square of the element. Therefore, on the right-hand side, the coefficient of $(1/2)p(x)p(x)$ is sought. Since the right-hand side is 0,

$$\mathbf{F} = [0] \tag{14}$$

is the core matrix of the recursion. From (9) and (10), it is possible to see that all $\boldsymbol{\rho}$ with positive index are 0 because the computation involves multiplication by $\mathbf{F}$ which is [0]. Therefore only $\boldsymbol{\rho}_0$ is nonzero. Using this in (8), the solution of the initial value problem is

$$\mathbf{y}(t) = t^0 \boldsymbol{\rho}_0, \tag{15}$$

where only the beginning term of the Taylor series is nonzero. Since $\boldsymbol{\rho}_0$ is the initial value, plugging in the value gives

$$\mathbf{y}(t) = [1], \tag{16}$$

where the solution is the constant function 1. Here, we did not need to truncate the series to get a numerical result since only the beginning term of the series is nonzero.

It is important to emphasize here that for any such application of probabilistic evolution theory for the solution of system of ODEs, the method does not use discretization. It gives the Taylor expansion of the solution. The Taylor expansion itself is exact, it is not an approximation.

 Copyright © 2026 Shahid Beheshti University.

3.2. The function whose derivative is 1

Consider the ODE

$$\dot{p}(t) = 1\,, \quad p(0) = 0\,, \tag{17}$$

where the right-hand side is a nonzero constant. The nonzero constant is not a second degree polynomial. To get only second degree terms on the right-hand side, constancy adding space extension (CASE) is used. One more equation is added to the set so that all additive terms of all the right-hand sides become second degree. Using CASE, the ODE set becomes

$$\dot{p}(t) = 1x(t)x(t)\,, \qquad p(0) = 0\,, \tag{18}$$

$$\dot{x}(t) = 0\,, \qquad x(0) = 1\,, \tag{19}$$

where all additive terms on the right-hand sides are second degree. The zero constant is not a problem since 0 is $0x(t)x(t)$, thus a second degree term with 0 as its coefficient. Now, the system vector is

$$\mathbf{y} = \begin{bmatrix} p \\ x \end{bmatrix}, \tag{20}$$

consisting of two elements. The target notation is

$$\dot{\mathbf{y}}(t) = \mathbf{F}\left(\frac{1}{2}\mathbf{y}(t)^{\otimes_c^2}\right), \quad \mathbf{y}(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{21}$$

and performing the condensed Kronecker squaring and multiplying by $1/2$ gives

$$\dot{\mathbf{y}}(t) = \mathbf{F}\begin{bmatrix} p^2/2 \\ px \\ x^2/2 \end{bmatrix}, \quad \mathbf{y}(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{22}$$

Rewriting the vector on left-hand side explicitly

$$\begin{bmatrix} \dot{p} \\ \dot{x} \end{bmatrix} = \mathbf{F}\begin{bmatrix} p^2/2 \\ px \\ x^2/2 \end{bmatrix}, \quad \mathbf{y}(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{23}$$

appears. Then, the question is finding the matrix $\mathbf{F}$. The right-hand side of (18) has $x(t)^2$. To obtain the coefficient 1 of $x(t)^2$, the element in first row third column in $\mathbf{F}$ should be 2. Then the 2s will cancel and the coefficient will be 1. So the ODE set is

$$\begin{bmatrix} \dot{p} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} p^2/2 \\ px \\ x^2/2 \end{bmatrix}, \quad \mathbf{y}(0) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{24}$$

where $p(t)$ is the sought function. $x(t)$ is an auxiliary function that is used in order to make the right-hand side terms second degree polynomials. The initial value of the recursion is the initial value of the ODE. Therefore

$$\boldsymbol{\rho}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{25}$$

holds. In (9), substituting 0 for j,

$$\boldsymbol{\rho}_1 = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}\left(\frac{1}{2}\left(\boldsymbol{\rho}_0 \otimes_c \boldsymbol{\rho}_0\right)\right) \tag{26}$$

is obtained. For the finite sum from 0 to $\left(\frac{j}{2} - 1\right)$ in (9), a sum where the upper limit is smaller than the lower limit is taken as 0, by convention). Substituting and using (25) in (26) and performing the condensed Kronecker product

$$\boldsymbol{\rho}_1 = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}\left(\frac{1}{2}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right) \tag{27}$$

appears. Going step by step, $\boldsymbol{\rho}_1$ is

$$\boldsymbol{\rho}_1 = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} \end{bmatrix}, \tag{28}$$

and performing the multiplication $\boldsymbol{\rho}_1$ is found to be

$$\boldsymbol{\rho}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{29}$$

 Copyright © 2026 Shahid Beheshti University.

Substituting j with 1 in (10)

$$\boldsymbol{\rho}_2 = \frac{1}{2}\begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}(\boldsymbol{\rho}_0 \otimes_c \boldsymbol{\rho}_1) \tag{30}$$

appears. Using (25) and (29) in (30),

$$\boldsymbol{\rho}_2 = \frac{1}{2}\begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes_c \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) \tag{31}$$

is computed. Performing the operations, $\boldsymbol{\rho}_2$ is

$$\boldsymbol{\rho}_2 = \mathbf{0}\,. \tag{32}$$

Any condensed Kronecker product with the zero vector results in the zero vector. Only nonzero condensed Kronecker product results for this example are condensed Kronecker product of two $\boldsymbol{\rho}_0$s and the condensed Kronecker product of two $\boldsymbol{\rho}_1$s. The condensed Kronecker product of two $\boldsymbol{\rho}_1$s in the computing of $\boldsymbol{\rho}_3$ will not be making a contribution because of the structure of $\mathbf{F}$. With all this in mind, it is possible to come to the conclusion that $\boldsymbol{\rho}_i = \mathbf{0}$ for $i > 1$. Therefore, using (8), the solution of the initial value problem is

$$\begin{aligned} \begin{bmatrix} p(t) \\ x(t) \end{bmatrix} &= t^0\boldsymbol{\rho}_0 + t^1\boldsymbol{\rho}_1 \\ &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} + t\begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} t \\ 1 \end{bmatrix}, \end{aligned} \tag{33}$$

where $p(t)$ is found to be just t. For this example, from an empirical perspective, it is possible to truncate the series in (8) to take only the first two terms (or more), without knowing that the truncation from the series is also the exact solution of the initial value problem.

3.3. Two decoupled equations where the functions' derivatives are equal to themselves

Consider the initial value problem below.

$$\dot{p}(t) = p(t)\,, \qquad p(0) = 1\,, \tag{34}$$

$$\dot{q}(t) = q(t)\,, \qquad q(0) = 1\,. \tag{35}$$

The right-hand sides are first degree; therefore, CASE is necessary. After CASE, the new ODE set is

$$\dot{p}(t) = p(t)x(t)\,, \qquad p(0) = 1\,, \tag{36}$$

$$\dot{q}(t) = q(t)x(t)\,, \qquad q(0) = 1\,, \tag{37}$$

$$\dot{x}(t) = 0\,, \qquad x(0) = 1\,, \tag{38}$$

consisting of three equations and three unknowns. The system vector is

$$\mathbf{y} = \begin{bmatrix} p \\ q \\ x \end{bmatrix}, \tag{39}$$

where the first two rows are the sought functions and the last row is the auxiliary function of CASE. The desired representation is

$$\dot{\mathbf{y}}(t) = \mathbf{F}\left(\frac{1}{2}\mathbf{y}(t)^{\otimes_c^2}\right), \quad \mathbf{y}(0) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \tag{40}$$

where $\mathbf{F}$ is a 3×6 matrix. Writing explicitly the vector that $\mathbf{F}$ acts upon,

$$\dot{\mathbf{y}}(t) = \mathbf{F}\begin{bmatrix} p^2/2 \\ pq \\ px \\ q^2/2 \\ qx \\ x^2 \end{bmatrix}, \quad \mathbf{y}(0) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{41}$$

is obtained. The next step is to write $\mathbf{F}$ explicitly. (36) has $p(t)x(t)$ as right-hand side. Therefore, for the first row, only the third column is nonzero. For the first row, the third column is 1, so that the coefficient of $p(t)x(t)$ is 1. (37) has $q(t)x(t)$ as

 Copyright © 2026 Shahid Beheshti University.

right-hand side. Therefore, for the first row, only the fifth column is nonzero. For the second row, the fifth column is 1, so that the coefficient of $q(t)x(t)$ is 1. Showing $\mathbf{F}$ explicitly in (41),

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p^2/2 \\ pq \\ px \\ q^2/2 \\ qx \\ x^2 \end{bmatrix}, \quad \mathbf{y}(0) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{42}$$

is obtained. The initial value of the recursion will be the initial value of (40). Therefore, it is

$$\boldsymbol{\rho}_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \tag{43}$$

shown explicitly. In (9), substituting 0 for j,

$$\boldsymbol{\rho}_1 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \left(\frac{1}{2} \left(\boldsymbol{\rho}_0 \otimes_c \boldsymbol{\rho}_0 \right) \right) \tag{44}$$

is obtained. For the finite sum from 0 to $\left(\frac{j}{2} - 1\right)$ in (9), a sum where the upper limit is smaller than the lower limit is taken as 0, by convention. Substituting and using (43) in (44) and performing the condensed Kronecker product

$$\boldsymbol{\rho}_1 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/2 \\ 1 \\ 1 \\ 1/2 \\ 1 \\ 1/2 \end{bmatrix} \tag{45}$$

appears. $\boldsymbol{\rho}_1$ is found to be

$$\boldsymbol{\rho}_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}. \tag{46}$$

Substituting j with 1 in (10)

$$\boldsymbol{\rho}_2 = \frac{1}{2} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \left(\boldsymbol{\rho}_0 \otimes_c \boldsymbol{\rho}_1 \right) \tag{47}$$

appears. Using (43) and (46) in (47),

$$\boldsymbol{\rho}_2 = \frac{1}{2} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \tag{48}$$

and performing the algebra,

$$\boldsymbol{\rho}_2 = \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \end{bmatrix} \tag{49}$$

is obtained. Putting it together using (8), the solution is

$$\begin{aligned} \begin{bmatrix} p(t) \\ q(t) \\ x(t) \end{bmatrix} &= t^0 \boldsymbol{\rho}_0 + t^1 \boldsymbol{\rho}_1 + t^2 \boldsymbol{\rho}_2 + \cdots \\ &= \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + t \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + t^2 \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \end{bmatrix} + \cdots \\ &= \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} t \\ t \\ 0 \end{bmatrix} + \begin{bmatrix} t^2/2 \\ t^2/2 \\ 0 \end{bmatrix} + \cdots, \end{aligned} \tag{50}$$

 Copyright © 2026 Shahid Beheshti University.

where there are infinitely many terms on the right. From an empirical perspective, a truncation can be performed to get numerical approximations. For this example, the first and second rows both build up the Taylor expansion of e^t. The series itself is exact. Truncation from the series is the numerical method.

3.4. The function whose derivative is t

Consider the ODE

$$\dot{x}_1(t) = t\,, \qquad x_1(0) = 0\,. \tag{51}$$

This trivial ODE is non-autonomous. The way to work with this ODE is to manually introduce another unknown. This new unknown is t. To avoid the name collision, the new unknown can be named as x_2. Then, the ODE set becomes

$$\dot{x}_1(t) = x_2(t)\,, \qquad x_1(0) = 0\,, \tag{52}$$
$$\dot{x}_2(t) = 1\,, \qquad x_2(0) = 0\,, \tag{53}$$

where the right-hand sides are autonomous. To obtain purely second degree right-hand sides, one more equation is introduced to the set. Then the ODE set becomes

$$\dot{c}(t) = 0\,, \qquad c(0) = 1\,, \tag{54}$$
$$\dot{x}_1(t) = x_2(t)c(t)\,, \qquad x_1(0) = 0\,, \tag{55}$$
$$\dot{x}_2(t) = 1c(t)c(t)\,, \qquad x_2(0) = 0\,, \tag{56}$$

where the newly introduced equation is written at the top. The system vector is

$$\mathbf{y} = \begin{bmatrix} c \\ x_1 \\ x_2 \end{bmatrix}\,, \tag{57}$$

where the element in the second row is the sought function. The desired representation is

$$\dot{\mathbf{y}}(t) = \mathbf{F}\left(\frac{1}{2}\mathbf{y}(t)^{\otimes_c^2}\right)\,, \quad \mathbf{y}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\,, \tag{58}$$

where $\mathbf{F}$ is a 3×6 matrix. Writing explicitly the vector that $\mathbf{F}$ acts upon,

$$\dot{\mathbf{y}}(t) = \mathbf{F}\begin{bmatrix} c^2/2 \\ cx_1 \\ cx_2 \\ x_1^2/2 \\ x_1x_2 \\ x_2^2/2 \end{bmatrix}\,, \quad \mathbf{y}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{59}$$

is obtained. Finding $\mathbf{F}$ and showing it explicitly in (59),

$$\begin{bmatrix} \dot{c} \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} c^2/2 \\ cx_1 \\ cx_2 \\ x_1^2/2 \\ x_1x_2 \\ x_2^2/2 \end{bmatrix}\,, \quad \mathbf{y}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{60}$$

is obtained. The initial value of the recursion will be the initial value of (60). Therefore it is

$$\boldsymbol{\rho}_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}\,, \tag{61}$$

and the other $\boldsymbol{\rho}$ values can be computed by (9) and (10). It is observed that

$$\boldsymbol{\rho}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\,, \quad \boldsymbol{\rho}_2 = \begin{bmatrix} 0 \\ 1/2 \\ 0 \end{bmatrix}\,, \tag{62}$$

 Copyright © 2026 Shahid Beheshti University.

and

$$\boldsymbol{\rho}_i = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, i > 2, \tag{63}$$

therefore, the only contribution for $x_1(t)$ comes from $\boldsymbol{\rho}_2$. Putting it together using (8), the solution is

$$x_1(t) = \frac{1}{2}t^2, \tag{64}$$

where the other two introduced functions are not shown explicitly.

4. Nontrivial example

4.1. Rabinovich-Fabrikant ODE

Consider the following ODE

$$\dot{x}(t) = y(t)z(t) - y(t) + x(t)x(t)y(t) + x(t), \qquad x(0) = 1/2, \tag{65}$$
$$\dot{y}(t) = 3x(t)z(t) + x(t) - x(t)x(t)x(t) + y(t), \qquad y(0) = 1/2, \tag{66}$$
$$\dot{z}(t) = -2z(t) - 2x(t)y(t)z(t), \qquad z(0) = 1/2, \tag{67}$$

with three equations and three unknowns. After quadratization, the ODE set becomes

$$\dot{x}(t) = w_0(t)y(t) + x(t) + y(t)z(t) - y(t), \qquad x(0) = 1/2, \tag{68}$$
$$\dot{y}(t) = -w_0(t)x(t) + 3x(t)z(t) + x(t) + y(t), \qquad y(0) = 1/2, \tag{69}$$
$$\dot{z}(t) = -2w_1(t)z(t) - 2z(t), \qquad z(0) = 1/2, \tag{70}$$
$$\dot{w_0}(t) = 2w_0(t)w_1(t) + 2w_0(t) + 2w_1(t)z(t) - 2w_1(t), \qquad w_0(0) = 1/4, \tag{71}$$
$$\dot{w_1}(t) = -w_0(t)^2 + 3w_0(t)z(t) + w_0(t) + w_1(t)^2 + 2w_1(t) + w_2(t)z(t) - w_2(t), \qquad w_1(0) = 1/4, \tag{72}$$
$$\dot{w_2}(t) = -2w_0(t)w_1(t) + 6w_1(t)z(t) + 2w_1(t) + 2w_2(t), \qquad w_2(0) = 1/4, \tag{73}$$

with six equations and six unknowns. The functional dependence of the new functions on the original functions are as follows.

$$w_0(t) = x(t)^2, \tag{74}$$
$$w_1(t) = x(t)y(t), \tag{75}$$
$$w_2(t) = y(t)^2. \tag{76}$$

After CASE, the ODE set becomes

$$\dot{c}(t) = 0c(t)c(t), \qquad c(0) = 1, \tag{77}$$
$$\dot{x}(t) = w_0(t)y(t) + x(t)c(t) + y(t)z(t) - y(t)c(t), \qquad x(0) = 1/2, \tag{78}$$
$$\dot{y}(t) = -w_0(t)x(t) + 3x(t)z(t) + x(t)c(t) + y(t)c(t), \qquad y(0) = 1/2, \tag{79}$$
$$\dot{z}(t) = -2w_1(t)z(t) - 2z(t)c(t), \qquad z(0) = 1/2, \tag{80}$$
$$\dot{w_0}(t) = 2w_0(t)w_1(t) + 2w_0(t)c(t) + 2w_1(t)z(t) - 2w_1(t)c(t), \qquad w_0(0) = 1/4, \tag{81}$$
$$\dot{w_1}(t) = -w_0(t)^2 + 3w_0(t)z(t) + w_0(t)c(t) + w_1(t)^2 + 2w_1(t)c(t) + w_2(t)z(t) - w_2(t)c(t), \qquad w_1(0) = 1/4, \tag{82}$$
$$\dot{w_2}(t) = -2w_0(t)w_1(t) + 6w_1(t)z(t) + 2w_1(t)c(t) + 2w_2(t)c(t), \qquad w_2(0) = 1/4, \tag{83}$$

where all the right-hand sides are purely second degree polynomials. The initial value of the recursion $\boldsymbol{\rho}_0$ is

$$\boldsymbol{\rho}_0 = \begin{bmatrix} 1 \\ 1/2 \\ 1/2 \\ 1/2 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix}, \tag{84}$$

and the core matrix $\mathbf{F}$ is a 7×28 matrix whose elements can be computed from the coefficients in (77-83). The $\boldsymbol{\rho}$ with higher subindex can be computed from the recursion given in (9) and (10).

 Copyright © 2026 Shahid Beheshti University.

Using PREVTH, the solution is

$$
\begin{bmatrix} c(t) \\ x(t) \\ y(t) \\ z(t) \\ w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} = t^0 \begin{bmatrix} 1 \\ 1/2 \\ 1/2 \\ 1/2 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} + t^1 \begin{bmatrix} 0 \\ 3/8 \\ 1 \\ 13/8 \\ 3/8 \\ 13/8 \\ -5/4 \end{bmatrix} + t^2 \begin{bmatrix} 0 \\ -3/32 \\ 19/32 \\ 91/32 \\ -15/64 \\ 13/64 \\ 17/16 \end{bmatrix}
$$

$$
+ t^3 \begin{bmatrix} 0 \\ -7/12 \\ -35/64 \\ 7/12 \\ -313/768 \\ -59/768 \\ -1/4 \end{bmatrix} + t^4 \begin{bmatrix} 0 \\ -79/768 \\ -403/768 \\ 55/768 \\ 1817/12288 \\ 1147/4096 \\ 17/128 \end{bmatrix} + t^5 \begin{bmatrix} 0 \\ 787/7680 \\ 169/480 \\ 3127/2560 \\ -49021/245760 \\ 84197/245760 \\ -1487/3840 \end{bmatrix}
$$

$$
+ t^6 \begin{bmatrix} 0 \\ -18193/92160 \\ -14657/30720 \\ 75881/92160 \\ -852839/5898240 \\ -2416787/5898240 \\ 3821/46080 \end{bmatrix} + t^7 \begin{bmatrix} 0 \\ -16921/80640 \\ -93563/143360 \\ -919/768 \\ -12280969/165150720 \\ 917963/23592960 \\ 30049/80640 \end{bmatrix} + t^8 \begin{bmatrix} 0 \\ 229643/1032192 \\ -37253/573440 \\ 385921/5160960 \\ 45843611/1761607680 \\ 470386561/5284823040 \\ -658991/2580480 \end{bmatrix}
$$

$$
+ t^9 \begin{bmatrix} 0 \\ 5796601/92897280 \\ 109727/483840 \\ 11438083/18579456 \\ -9673494109/190253629440 \\ 10600942621/190253629440 \\ -99341/1032192 \end{bmatrix} + t^{10} \begin{bmatrix} 0 \\ 5615261/265420800 \\ -623428567/1857945600 \\ -133873907/1857945600 \\ 105398116961/7610145177600 \\ -48258832873/281857228800 \\ 20398591/309657600 \end{bmatrix}
$$

$$
+ t^{11} \begin{bmatrix} 0 \\ 3946651/102187008 \\ -3618985451/40874803200 \\ -682083937/851558400 \\ -3374656336441/334846387814400 \\ 1834585218869/334846387814400 \\ 90077611/510935040 \end{bmatrix} + t^{12} \begin{bmatrix} 0 \\ 61919980021/490497638400 \\ 1199620237/10899947520 \\ 17593354549/70071091200 \\ 93168427773007/2296089516441600 \\ 1076683967607601/16072626615091200 \\ -38130733757/245248819200 \end{bmatrix}
$$

$$
+ t^{13} \begin{bmatrix} 0 \\ 496482289447/12752938598400 \\ 2841407047/16605388800 \\ 22947959357/77290536960 \\ 1586050459130633 9/835776583984742400 \\ -8064701075885131/835776583984742400 \\ -290302281959/6376469299200 \end{bmatrix}
$$

$$
+ t^{14} \begin{bmatrix} 0 \\ 1248656275409/51011754393600 \\ -10415742425393/119027426918400 \\ -68422090630783/357082280755200 \\ -20797378054523453/15601162901048524800 \\ -2690586590362612211/46803488703145574400 \\ 139777194901/2318716108800 \end{bmatrix}
$$

$$
+ t^{15} \begin{bmatrix} 0 \\ -23085576070393/669529276416000 \\ 1904574369839/396758089728000 \\ -20855862768733/60866297856000 \\ 2471923976798433233/401172760312676352000 \\ 31956360033994551229/2808209322188734464000 \\ 1636772760121/20288765952000 \end{bmatrix}
$$

$$
+ \ldots, \tag{85}
$$

 Copyright © 2026 Shahid Beheshti University.

where truncating the series and substituting a value for t gives numerical results.

4.1.1. Remarks on possibly nonoptimal quadratization: [10] shows a fast and easy way for possibly nonoptimal quadratization. Although finding a possibly nonoptimal quadratization is fast, the nonoptimality causes PREVTH to work on matrices that are larger than necessary. The quadratization of QBee uses branch and bound search and therefore, optimality is guaranteed. The method in [10] uses beam search therefore optimality is not guaranteed. The method in [10] relies on dividing the powers by 2 recursively and introducing new functions until quadratization is reached. When dividing odd powers by 2, the option of rounding up or down is available. Changing this option and changing the way that CASE is implemented, different varieties are proposed in [10]. None of these varieties can give the seven equations and seven unknowns in (83).

The observant reader may also notice that optimal quadratization is not unique. QBee gives one of the optimal quadratizations. That quadratization may be one of the quadratizations that necessitate CASE, whereas another one may not necessitate CASE. Therefore, the resulting pure quadratization may have one more equation than the actual optimal pure quadratization. This is a small technical detail.

5. Implementation details

A program has been implemented for solving ODEs using optimal quadratization and PREVTH. The procedure for the solution is polynomialization + quadratization + CASE + PREVTH. For polynomialization + quadratization, the QBee software by Andrey Bychkov and Gleb Pogudin is used[†]. This program is a very powerful program written in Python for polynomializing and quadratizing ODE systems. The novelty in this work is to combine the output of the QBee software with CASE + PREVTH.

For PREVTH, there is a software implemented by Coşar Gözükırmızı [10]. This software is able to quadratize systems but by a very naive approach[‡]. Quadratizing nonoptimally greatly increases the computational burden of PREVTH. Therefore changing the PREVTH software and putting it together with QBee had appeared as a necessity.

For this purpose, the PREVTH software is simplified to directly take the core matrix (without any quadratization and CASE). The PREVTH software is quite a powerful and efficient software written in C++ language.

For CASE, a Python script is written that makes the terms of right-hand side functions only second degree terms and also makes an output that can be read by the PREVTH software.

To summarize, we use directly (without any changes) the QBee software for polynomialization + quadratization. A novel Python script is implemented by Coşar Gözükırmızı that takes the output of quadratization as input, performs CASE and puts an output in JSON. JSON is a light-weight data interchange format. A new PREVTH software is written by Coşar Gözükırmızı that takes the output of CASE, and uses probabilistic evolution theory to compute the result of the initial value problem.

The final ODE solver is polynomialization + quadratization + CASE + PREVTH. Of course, depending on the problem, some of the first three steps may be skipped. If the ODE system is already polynomial, polynomialization will be skipped. If quadratization gives an output that is purely quadratic, CASE will not perform pure quadratization and will just put the output together in JSON format so that PREVTH can take it as input. It is also possible to run PREVTH from the JSON input made manually, without worrying about the first three steps. Therefore, the program is quite modular.

Many different example ODE systems come with the program, so that the user can easily familiarize with the usage of the program. In addition to the examples discussed in this paper, common benchmark ODEs such as van der Pol are also available as input.

If polynomialization is necessary, the program asks the user the initial values of the newly introduced unknowns. The initial values of the newly introduced unknowns by quadratization are computed automatically by the program.

The program works with rational numbers in fraction form to avoid error accumulation due to the finite nature of floating-point arithmetic. Therefore, the initial values and coefficients can be entered in fraction form in the input file.

The JSON string that is the output of CASE and input of PREVTH, is exemplified below by using the Rabinovich-Fabrikant ODE.

Listing 1: JSON string for Rabinovich-Fabrikant ODE

```
{
        "name": "rabinovich fabrikant",
        "coreMatrix": [
        "1"  , "2"  , "-2"  ,
        "1"  , "17"  , "2"  ,
        "1"  , "1"  , "2"  ,
        "1"  , "8"  , "2"  ,
        "2"  , "3"  , "-1"  ,
```

[†]https://github.com/AndreyBychkov/QBee
[‡]https://github.com/cosargozukirmizi/space-extension-of-explicit-ODEs

 Copyright © 2026 Shahid Beheshti University.

```
        "2" , "21" , "1" ,
        "2" , "2" , "2" ,
        "2" , "13" , "2" ,
        "2" , "1" , "1" ,
        "2" , "12" , "3" ,
        "2" , "7" , "-2" ,
        "3" , "3" , "2" ,
        "3" , "2" , "2" ,
        "3" , "17" , "6" ,
        "3" , "8" , "-2" ,
        "4" , "5" , "-1" ,
        "4" , "26" , "1" ,
        "4" , "4" , "1" ,
        "4" , "11" , "1" ,
        "5" , "5" , "1" ,
        "5" , "4" , "1" ,
        "5" , "24" , "3" ,
        "5" , "10" , "-1" ,
        "6" , "6" , "-2" ,
        "6" , "17" , "-2"
        ],
        "initVals": [
        "1" , "1/4" , "1/4" , "1/4" , "1/2" , "1/2" , "1/2"
        ],
        "stringsForUnknowns": [
        "c" , "w0" , "w1" , "w2" , "x" , "y" , "z"
        ],
        "t_max": "1/10",
        "floatingPointNumberOfBits": 500,
        "floatingPointPrintNumberOfDigits": 50

}
```


The name *coreMatrix* stores **F** in coordinate format. The triples refer to row, column, and value. Indexing starts from 0. The name *initVals* stores the initial values. The name *stringsForUnknowns* stores the unknown functions. The name *t_max* stores the maximum t. Evaluation is performed in 10 equidistant t values from 0 to t_{max}. All computations are done using fractions. The values are converted to floating-point at the very end using the number of bits specified by the name *floatingPointNumberOfBits*. In the output of floating-point numbers, the number of decimal digits to be printed is determined by the name *floatingPointPrintNumberOfDigits*.

Since the core matrix of PREVTH, the **F** matrix, is stored sparsely, the use of memory is also efficient. To make the program faster, PREVTH also uses concurrency. The details of this can be found in [11].

5.1. Setting up the ODE Solver

In this subsection, certain remarks about setting up the software are given. The first step is to set up QBee from https://github.com/AndreyBychkov/QBee. This software has extensive documentation. The best way to get it working is through a Python virtual environment. After ensuring that QBee works, the next step is to download the contents of the prevth repository[§]. This repository has both Python and C++ files. The Python part will work without any problem if the QBee is working correctly. The C++ code is written using the C++20 standard, therefore, compiling needs to be done using this standard or a later one. The C++ code depends on the libraries fstream, nlohmann/json.hpp, iostream, vector, map, gmpxx.h, iomanip, thread, semaphore, algorithm, cassert, chrono. Of all these, only nlohmann/json.hpp and gmpxx.h do not come with C++. They should be installed separately. nlohmann/json.hpp is a free header-only JSON parser. The installation details can be found in https://github.com/nlohmann/json. It is also available in many modern linux repositories. The gmpxx.h is the multiple precision arithmetic library and the details about how to set it up can be reached at https://gmplib.org/manual/Installing-GMP. This library is also available in many modern linux repositories. On a Linux system, the command for compiling the C++ program (a single C++ file) is

```
g++ -std=c++20 main.cpp -lgmp -lgmpxx -o main.x
```

[§]https://gitlab.com/cosargozukirmizi/prevth/

 Copyright © 2026 Shahid Beheshti University.

Once the C++ program (a single C++ file) is compiled without any problem, the next step is to test it. Considering that there is a file circular.json in the QBee directory and main.x (the compiled main.cpp) is in the current directory, the command

```
./main.x < QBee/circular.json
```

will input the json to the C++ program. This should give the solution of the ODE system.

The next step is to combine everything. Assuming that we are in the directory where main.x is, and the file qbee2jsonDriverCircular.py is in the QBee directory, the command

```
python3 QBee/qbee2jsonDriverCircular.py | ./main.x
```

will do Quadratization + CASE + PREVTH to give the solution. For ODEs where the polynomialization is also necessary, the new initial values introduced by the polynomialization are asked to the user for keyboard input.

Efforts for providing an extensive documentation for the software are ongoing.

6. Concluding remarks

An ODE solver is implemented by combining optimal quadratization and PREVTH. The ODE solver can be used for the solution of a very large variety of first-order explicit ODE sets. To give hands-on experience with the method, solutions to certain problems are included in the paper. Many other nontrivial ODE examples are provided within the software as input files to the ODE solver.

It is also important to point out that the choice of the quadratization (optimal or any of the nonoptimal) does not affect the result. It only affects the running time of PREVTH. For very large systems, finding the optimal quadratization may be challenging. The process will be time-consuming for very large systems and memory limitation will also be a problem. The tree in which the optimal quadratization is searched, grows rapidly when the number of equations increases. This issue is beyond the scope of this paper.

References

1. A. Bychkov, O. Issan, G. Pogudin, and B. Kramer. Exact and optimal quadratization of nonlinear finite-dimensional nonautonomous dynamical systems. *SIAM Journal on Applied Dynamical Systems*, 23(1):982–1016, 2024.
2. A. Bychkov and G. Pogudin. Optimal monomial quadratization for ODE systems: Extended abstract. *ACM Commun. Comput. Algebra*, 54(3):119–123, Mar. 2021.
3. Y. Cai and G. Pogudin. Dissipative quadratizations of polynomial ODE systems. In B. Finkbeiner and L. Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 323–342, Cham, 2024. Springer Nature Switzerland.
4. M. Delkhosh. A new computational method based on fractional integration operational matrices for solving fractional integro–differential equations. *International Journal of Differential Equations*, 2026(1):4658137, 2026.
5. M. Delkhosh and K. Parand. Generalized pseudospectral method: Theory and applications. *Journal of Computational Science*, 34:11–32, 2019.
6. M. Delkhosh and K. Parand. A new computational method based on fractional lagrange functions to solve multi-term fractional differential equations. *Numerical Algorithms*, 88(2):729–766, Oct 2021.
7. M. Delkhosh, K. Parand, and D. Domiri Ganji. An efficient numerical method to solve the boundary layer flow of an Eyring-Powell non-Newtonian fluid. *Journal of Applied and Computational Mechanics*, 5(2):454–467, 2019.
8. M. Delkhosh and R. A. Shirvan. A hybrid numerical method based on the generalized pseudospectral method for solving nonlinear differential equations. *Journal of Computational Mathematics and Computer MODEling with Applications*, 1(2):129–138, 2022.
9. M. Demiralp. Probabilistic evolution theory in its basic aspects, and, its fundamental progressive stages. *Mathematics in Engineering, Science & Aerospace (MESA)*, 9(2):245, 2018.
10. C. Gözükırmızı. Beam search for space extension in explicit ordinary differential equation conicalization. *Computational Technologies*, 27(6):100–114, 2022.
11. C. Gözükırmızı. Pure quadratization and solution of ordinary differential equations by probabilistic evolution theory with concurrent computation of coefficients using exact arithmetic. *Journal of Mathematical Chemistry*, 62(3):654–680, 2024.
12. C. Gözükırmızı and M. Demiralp. Probabilistic evolution approach for the solution of explicit autonomous ordinary differential equations. Part 1: Arbitrariness and equipartition theorem in kronecker power series. *Journal of Mathematical Chemistry*, 52(3):866–880, 2014.
13. C. Gözükırmızı and M. Demiralp. Probabilistic evolution approach for the solution of explicit autonomous ordinary differential equations. Part 2: Kernel separability, space extension, and, series solution via telescopic matrices. *Journal of Mathematical Chemistry*, 52(3):881–898, 2014.
14. C. Gözükırmızı and M. Demiralp. Probabilistic evolution theory for explicit autonomous ordinary differential equations: Recursion of squarified telescope matrices and optimal space extension. *Journal of Mathematical Chemistry*, 56(7):1826–1848, Dec 2017.
15. C. Gözükırmızı and M. Demiralp. Solving ODEs by obtaining purely second degree multinomials via branch and bound with admissible heuristic. *Mathematics*, 7(4):367, 2019.
16. C. Gözükırmızı and M. E. Kırkın. Classical symmetric fourth degree potential systems in probabilistic evolution theoretical perspective: Most facilitative conicalization and squarification of telescope matrices. *AIP Conference Proceedings*, 1798(1):020061, 2017.
17. C. Gözükırmızı, M. E. Kırkın, and M. Demiralp. Probabilistic evolution theory for the solution of explicit autonomous ordinary differential equations: Squarified telescope matrices. *Journal of Mathematical Chemistry*, 55(1):175–194, 2017.

 Copyright © 2026 Shahid Beheshti University.

18. C. Gözükırmızı and E. Tataroğlu. Squarification of telescope matrices in the probabilistic evolution theoretical approach to the two particle classical mechanics as an illustrative implementation. *AIP Conference Proceedings*, 1798(1):020062, 2017.
19. M. Öztürk. *Creating a database for the solution of Van der Pol equation with probabilistic evolution method*. Masters thesis, İstanbul Beykent University, 2024. (Turkish translation: Van der Pol denkleminin olasılıksal evrim yöntemi ile çözümü için veri tabanı oluşturma).
20. K. Parand and M. Delkhosh. Accurate solution of the Thomas–Fermi equation using the fractional order of rational Chebyshev functions. *Journal of Computational and Applied Mathematics*, 317:624–642, 2017.

 Copyright © 2026 Shahid Beheshti University.