Research Article

Petr Veigend*, Gabriela Nečasová, and Václav Šátek

Solving linear and nonlinear problems using Taylor series method

https://doi.org/10.1515/comp-2024-0005
received January 15, 2024; accepted April 2, 2024

Abstract: The article deals with the solution of technical initial value problems. To solve such problems, an analytical or numerical approach is possible. The analytical approach can provide an accurate result; however, it is not available for all problems and it is not entirely suitable for calculation on a computer, due to the limited numerical accuracy. For this reason, the numerical approach is preferred. This approach uses ordinary differential equations to approximate the continuous behaviour of the real-world system. There are many known numerical methods for solving such equations, most of them limited in their accuracy, have a limited region of stability and can be quite slow to achieve the acceptable solution. The numerical method proposed in this article is based on the Taylor series and overcomes the biggest challenge, i.e. calculating higher derivatives. The aim of the article is therefore twofold: to introduce the method and show its properties, and to show its behaviour when solving problems composed of linear and nonlinear ordinary differential equations. Linear problems are modelled by partial differential equations and solved in parallel using the PETSc library. The parallel solution is demonstrated using the wave equation, which is transformed into the system of ordinary differential equations using the method of lines. The solution of nonlinear problems is introduced together with several optimisations that significantly increase the calculation speed. The improvements are demonstrated using several numerical examples that are solved using MATLAB software.

*** Corresponding author: Petr Veigend,** Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66, Brno, Czech Republic, e-mail: veigend@fit.vut.cz
Gabriela Nečasová: Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66, Brno, Czech Republic
Václav Šátek: Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66, Brno, Czech Republic; VSB - Technical University of Ostrava, IT4Innovations, 17. listopadu 15/2172, 708 33, Ostrava-Poruba, Czech Republic

Keywords: initial value problems, Taylor series, MTSM, MATLAB, PETSc

1 Introduction

This article deals with the numerical solution of technical initial value problems (IVPs) described by the systems of ordinary differential equations (ODEs). The ODEs are solved using a variable-step variable-order numerical integration method, the modern Taylor series method (MTSM), and the solution is compared with state-of-the-art ODE solvers. The complicated calculation of higher-order derivatives does not need to be calculated because MTSM recurrently calculates the Taylor series terms in each time step [1]. This article compares the numerical results of MTSM with the results obtained by the state-of-the-art Runge-Kutta solvers implemented in MATLAB software and PETSc library and shows the advantages of the MTSM over Runge-Kutta-based solvers. The aim of the article is to show that the MTSM can solve both linear and nonlinear problems faster and more accurately than the state-of-the-art Runge-Kutta-based solvers.

The first implementation of the MTSM was TKSL/386 software (TKSL stands for Taylor-Kunovsky simulation language) [2]. Currently, the MTSM has been implemented and tested in MATLAB [3], in C/C++ languages (FOS [4] and TKSL/C software [5]). Additionally, the method can be effectively implemented in hardware [6]. Several other implementations of the Taylor series method in a variable order and variable step context were presented by different authors e.g. TIDES software [7] and TAYLOR [8], which includes a detailed description of a variable-step-size version. Other implementations based on Taylor series include ATOMF [9], COSY INFINITY [10], and DAETS [11]. The variable step-size variable-order scheme is also described in previous studies [12–14], where simulations on a parallel computer are shown. An approach based on an approximate formulation of the Taylor methods can be found in the study by Baeza et al. [15].

The approach based on an approximate formulation of the Taylor methods can be found in the study by Baeza

et al. [15]. For example, further research was performed by Amodio et al. [16], which describes the generalised implementation of the Taylor series-based method with its order limited to three.

The MTSM allows for computation with arbitrary accuracy and step size if variable-precision arithmetic and higher-order of method are used. The article by Dimova et al. [17] focuses on the open multi-processing (OpenMP) parallelisation of multiple precision Taylor series method using one computational node. The model problem is the chaotic dynamic system – the classical Lorenz system. The article also briefly mentions the clean numerical simulation (CNS) concept, originally published by Liao [18]. The CNS provides reliable chaotic trajectories in a long enough interval of time.

A hybrid message passing interface (MPI) with OpenMP parallelisation strategy for multiple precision Taylor series method with fixed step size and fixed order is discussed by Hristov et al. [19]. The hybrid strategy was used because OpenMP scalability is slightly better than MPI when using one computational node. The authors claimed that this hybrid strategy can be applied to a large class of chaotic dynamical systems.

The article by Hristov et al. [20] is based on the previous article by Hristov et al. [19] and introduces a modification of CNS with variable step size and fixed order. The order of the Taylor series method is higher than the fixed-order approach, but it results in a reduced number of integration steps thanks to the larger integration step size. Also, the higher-order method increases parallel efficiency.

The article consists of several sections. Section 2 introduces the Taylor-series-based numerical integration method called the modern Taylor series method. Section 3 focuses on a parallel approach of solving linear partial differential equations (PDEs). Section 4 shows the parallel solution of the wave equation, together with performance metrics and detailed numerical results. The formulation of the method for nonlinear problems is introduced in Section 5, including several optimisations. Numerical results for selected examples of nonlinear problems are summarised in Section 6.

2 Higher-order Taylor series method

This section introduces the developed method – MTSM. More information can be found in previous studies [1,21–23]. An ODE with an initial condition

$$y'(t) = f(t, y(t)), \quad y(0) = y_0, \tag{1}$$

is called an initial value problem. The best-known and the most accurate method of the numerical solution of (1) is to construct the Taylor series in the form

$$y_{i+1} = y_i + h \cdot f(t_i, y_i) + \frac{h^2}{2!} \cdot f'(t_i, y_i) + \ldots + \frac{h^n}{n!} \cdot f^{[n-1]}(t_i, y_i), \tag{2}$$

where h is the integration step size, $y_i \approx y(t_i)$ is the approximation of the current value, and $y_{i+1} \approx y(t_i + h)$ is the approximation of the next value of the function $y(t)$ [24].

MTSM very effectively implements the variable-step-size, variable-order numerical solution of IVPs using the Taylor series [1]. It is based on a recurrent calculation of the Taylor series terms for each integration step. Therefore, the complicated calculation of higher-order derivatives does not need to be performed, but rather, the value of each Taylor series term can be numerically calculated. Equation (2) can then be rewritten in the form

$$y_{i+1} = p(0) + p(1) + p(2) + \ldots + p(n), \tag{3}$$

where $p(j)$, $j = 0 \ldots n$ denotes the Taylor series terms. The MTSM transforms the input problem into a system of autonomous ODEs, which allows the recurrent calculation of the Taylor series terms.

The Taylor series terms in each step are truncated when the stopping rule for the last σ number of Taylor series terms is met:

$$\sum_{j=n-\sigma}^{n} ||\boldsymbol{p}(j)|| \leq \varepsilon, \tag{4}$$

where ε is the required accuracy of the calculation, which is chosen for each example. In this article, we consider $\sigma = 3$.

An important part of the method is an automatic integration order setting, i.e. using as many Taylor series terms as the defined accuracy requires. Let P denote the function which changes during the computation and defines the number of Taylor series terms used in the current integration step ($P_{i+1} = n$). More information about the method and additional comparison with state-of-the-art methods can be found in the study by Veigend et al. [21].

3 Linear problems and parallel solution

For linear systems of ODEs ($\boldsymbol{y}'(t) = \mathbf{A}\boldsymbol{y}(t) + \mathbf{b}$), and (2) can be rewritten in matrix-vector notation as

$$\boldsymbol{y}_{i+1} = \boldsymbol{y}_i + h(\mathbf{A}\boldsymbol{y}_i + \mathbf{b}) + \frac{h^2}{2!}\mathbf{A}(\mathbf{A}\boldsymbol{y}_i + \mathbf{b}) + \ldots + \frac{h^n}{n!}\mathbf{A}^{(n-1)}(\mathbf{A}\boldsymbol{y}_i + \mathbf{b}), \tag{5}$$

where $\mathbf{A}$ is the constant Jacobian matrix and $\mathbf{b}$ is the constant right-hand side. Moreover, Taylor series terms in (3) can be computed recurrently using

$$\begin{aligned} \boldsymbol{p}(0) &= \boldsymbol{y}_i, \quad \boldsymbol{p}(1) = h(\mathbf{A}\boldsymbol{y}_i + \mathbf{b}), \\ \boldsymbol{p}(j) &= \frac{h}{j}\mathbf{A}\boldsymbol{p}(j-1), \quad j = 2, \ldots, n. \end{aligned} \tag{6}$$

When solving linear problems in parallel, (5) can be rewritten as follows:

$$\boldsymbol{y}_{i+1} = \mathbf{A}_y\boldsymbol{y}_i + \mathbf{A}_b\mathbf{b}, \tag{7}$$

where the matrices $\mathbf{A}_y$ and $\mathbf{A}_b$ are defined as follows:

$$\mathbf{A}_y = \sum_{k=0}^{n} \frac{h^k}{k!}\mathbf{A}^k, \quad \mathbf{A}_b = \sum_{k=1}^{n} \frac{h^k}{k!}\mathbf{A}^{k-1}. \tag{8}$$

Let $\mathbf{A}_{D_l}$ denote the submatrix of the matrix $\mathbf{A}$ decomposed by rows, where $l = \{1, 2, \ldots, n_P\}$ and n_P is the number of processes. Let $\mathbf{A}$ be a matrix of size $m \times m$. The number of rows of matrix $\mathbf{A}$ is evenly divided among processes. Matrices $\mathbf{A}_y$ and $\mathbf{A}_b$ are constant, and matrices are precalculated only once at the beginning of the calculation:

$$\hat{\mathbf{A}}_{yD_l} = \sum_{k=0}^{n} \frac{h^k}{k!}\mathbf{A}_{D_l}^k, \quad \hat{\mathbf{A}}_{bD_l} = \sum_{k=1}^{n} \frac{h^k}{k!}\mathbf{A}_{D_l}^{k-1}, \tag{9}$$

where $l = \{1, 2, \ldots, n_P\}$. After the parallel precalculation, the matrix $\hat{\mathbf{A}}_{yG}$ is obtained by gathering individual matrices $\hat{\mathbf{A}}_{yD_l}$. Similarly, the matrix $\hat{\mathbf{A}}_{bG}$:

$$\begin{aligned} \hat{\mathbf{A}}_{yG} &= (\hat{\mathbf{A}}_{yD_1}, \hat{\mathbf{A}}_{yD_2}, \ldots, \hat{\mathbf{A}}_{yD_p})^T, \\ \hat{\mathbf{A}}_{bG} &= (\hat{\mathbf{A}}_{bD_1}, \hat{\mathbf{A}}_{bD_2}, \ldots, \hat{\mathbf{A}}_{bD_p})^T. \end{aligned} \tag{10}$$

Final matrix $\hat{\mathbf{A}}$ and vector $\hat{\mathbf{b}}$ are calculated afterwards, and $\mathbf{I}$ is the identity matrix

$$\hat{\mathbf{A}} = \hat{\mathbf{A}}_{yG}\mathbf{A} + \mathbf{I}, \quad \hat{\mathbf{b}} = \hat{\mathbf{A}}_{bG}\mathbf{b}. \tag{11}$$

Using (11), we can rewrite (7) and solve it in parallel using the row-wise decomposition of matrix $\hat{\mathbf{A}}$

$$\boldsymbol{y}_{i+1} = \hat{\mathbf{A}}\boldsymbol{y}_i + \hat{\mathbf{b}}. \tag{12}$$

4 Linear problem example

The analytic notation of the wave equation is the second-order hyperbolic PDE [25] follows:

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial^2 u(x,t)}{\partial t^2} \quad (x,t) \in (0, L) \times \langle 0, T \rangle. \tag{13}$$

The homogeneous Dirichlet boundary conditions are set:

$$u(0,t) = u(L,t) = 0, \quad 0 \le t \le T, \tag{14}$$

where L is the string length and T is the maximum simulation time. The initial values follow:

$$u(x,0) = \sin(\pi x), \tag{15}$$

$$\frac{\partial u(x,0)}{\partial t} = 0, \quad 0 < x < L. \tag{16}$$

The wave equation describes the oscillations of an ideal string of a specified length. Both ends of the string are fixed in time (see the boundary conditions (14)). The initial velocity of the string is zero (16). The initial position of the string is modelled as a sine function (15). The resulting system of ODEs $\boldsymbol{y}'(t) = \mathbf{A}\boldsymbol{y}(t) + \mathbf{b}$, with initial conditions $\boldsymbol{y}(0) = \boldsymbol{y}_0$ arising from the Method Of Lines (MOL) is in the form:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix}, \quad \boldsymbol{y}_0 = \begin{pmatrix} \boldsymbol{u}(x,0) \\ \dfrac{\partial \boldsymbol{u}(x,0)}{\partial t} \end{pmatrix}, \quad \mathbf{b} = \mathbf{0}, \tag{17}$$

where $\mathbf{A}_{12}$ is the spatial discretisation matrix, the three-point central difference formula (coefficients $1, -2, 1$) was used. The $\mathbf{A}_{21}$ is the identity matrix, and matrices $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ are zero matrices. The size of the problem is denoted as S. The sparsity patterns of the matrices $\mathbf{A}$ and $\hat{\mathbf{A}}$ precalculated using (12) for $S = 100$ are in Figure 1(a) and (b), respectively.

The sparsity of the matrices $\mathbf{A}$ and $\hat{\mathbf{A}}$ increases with the problem size. The sparsity of $\hat{\mathbf{A}}$ decreases during precalculation as seen in Figure 1. Let us denote the size of the submatrix as $m = S - 1$. The following text will elaborate number of nonzero elements (nnz) of matrices $\mathbf{A}$ and $\hat{\mathbf{A}}$ in more detail. Total number of nonzero elements in matrix $\mathbf{A}$:

$$nnz_{\mathbf{A}} = nnz_{\mathbf{A}_{12}} + m, \tag{18}$$

$$nnz_{\mathbf{A}_{12}} = 3(m-2) + 4. \tag{19}$$

First, let us define the number of nonzero elements for matrix $\mathbf{A}_{12}$. Note that the first and last rows contain two elements (four overall), all other rows contain three elements. The last term, m, reflects the number of nonzero elements of the identity matrix $\mathbf{A}_{21}$. The term $P_{\max}$ denotes the maximum order of MTSM_PRECALC, that is the number of Taylor series terms (Table 2). The total number of nonzero elements in matrix $\hat{\mathbf{A}}$ for a given maximum order $P_{\max}$ can be calculated as follows:

$$nnz_{\hat{\mathbf{A}}} = nnz_{\mathbf{A}} + nnz_{\hat{\mathbf{A}}1} + nnz_{\hat{\mathbf{A}}2}. \tag{20}$$

The expression $nnz_{\hat{\mathbf{A}}1}$ denotes the number of nonzero elements for odd orders of the MTSM_PRECALC:

$$nnz_{\hat{\mathbf{A}}1} = \sum_{i=1}^{P_1} 2(m-i) + 2(m-i-1), \tag{21}$$

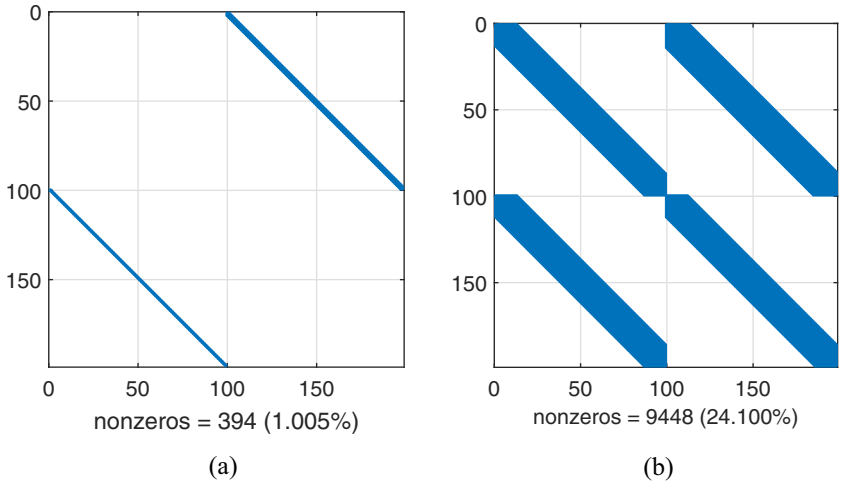

Figure 1: Sparsity patterns of input matrices, wave equation, three-point central difference formula, $S = 100$. (a) Matrix $\mathbf{A}$ and (b) matrix $\hat{\mathbf{A}}$.

where $2(m - i)$ denotes the number of nonzero elements in the submatrix $\mathbf{A}_{21}$ and $2(m - i - 1)$ in the submatrix $\mathbf{A}_{12}$. If $P_{\max}$ is odd, then $P_1 = (P_{\max} - 1)/2$, if $P_{\max}$ is even, then $P_1 = P_{\max}/2 - 1$.

The expression $nnz_{\hat{\mathbf{A}}2}$ denotes the number of nonzero elements for even orders of the MTSM_PRECALC:

$$nnz_{\hat{\mathbf{A}}2} = 2nnz_{\mathbf{A}_{12}} + \sum_{i=1}^{P_2} 4(m - i - 1), \tag{22}$$

where $2nnz_{\mathbf{A}_{12}}$ denotes the number of nonzero elements for $P = 2$ and $4(m - i - 1)$ is number of nonzero elements for even $P > 2$. If $P_{\max}$ is odd, then $P_2 = (P_{\max} - 1)/2$, if $P_{\max}$ is even, then $P_2 = P_{\max}/2$.

The density (d) of a matrix (percentage of nonzero elements) can be expressed as follows:

$$d\ [\%] = \frac{nnz}{(2S - 2)^2} \cdot 100, \tag{23}$$

where nnz is number of nonzero elements in matrix $\mathbf{A}$ or $\hat{\mathbf{A}}$ and $(2S - 2)^2$ is the total number of elements. The density of the matrix $\mathbf{A}$ for a given problem size $S = 256{,}000$ is $3.91 \times 10^{-4}\%$. The matrix $\hat{\mathbf{A}}$ precalculated to the order 25 is approximately 25.5 times less sparse, so the density is $9.96 \times 10^{-3}\%$.

The numerical solution of a given PDE is obtained by the MOL, which discretises the problem in spatial dimension and integrates the semi-discrete system of ODEs. The resulting system of ODEs is solved as IVP in the time domain using numerical integration methods. Table 1 shows the PETSc solvers used for the numerical experiments. The results obtained by Taylor series-based solvers were compared with two selected Runge-Kutta solvers. The TSRK5DP implements the fifth-order Dormand-Prince 5(4) method with a fourth-order embedded method (known as ode45 in MATLAB). The TSRK8VR is the eighth-order robust Verner scheme with a seventh-order embedded method with thirteen stages. All Taylor series-based solvers were implemented using PETSc library routines.

Table 1: PETSc numerical integration methods

Solver	Method
TSRK5DP	Dormand-Prince [26] (equivalent with ode45 in MATLAB)
TSRK8VR	Verner Runge-Kutta [27]
MTSM	Linear MTSM (5)
MTSM_PRECALC	MTSM with the $\hat{\mathbf{A}}$ precalculation (11) and (12)

The simulation experiments were performed on the Barbora supercomputer cluster, IT4Innovations National Supercomputing Center, Ostrava, Czech Republic [28]. The 32 compute nodes were utilised, each running 36 processes, resulting in a total count of 1152 MPI processes. The PETSc library [29–32] leverages the MPI standard for message-passing communication and offers data structures along with numerical methods that incorporate automatic step size control. It is explicitly designed for the parallel solution of scientific applications modelled by PDEs. All Taylor series-based solvers were implemented using PETSc library routines. Walltime was set to 15 min for all experiments. The parameters for the simulation experiments are shown in Table 2. Note that the maximum order for MTSM is 64, and the maximum order for MTSM_PRECALC is 25 (matrix $\hat{\mathbf{A}}$ was precalculated to the order of 25). Three-point central difference formula is used to discretise the spatial domain. The integration step size was chosen based on the numerical stability analysis of Taylor-series-based solvers [33,34].

Table 2: Parameters for linear simulation experiments

Parameter	Value
Problem size	$S = 256{,}000$
Number of first-order ODEs	$2S-2$
Length of the string	$L = 25{,}600$ (mm)
Spatial step size	$\Delta x = L/S = 0.1$ (mm)
Accuracy	$\text{rel}_{\text{TOL}} = \text{abs}_{\text{TOL}} = 1 \times 10^{-10}$
Integration step size	$h = 0.4$ (s)
Maximum simulation time	$T = 10{,}000 \cdot h$ (s)
Maximum order of MTSM	$P_{\max} = 64$
Maximum order of MTSM_PRECALC	$P_{\max} = 25$

4.1 Performance metrics

The performance metrics evaluate, characterise, diagnose, and tune parallel performance [35]. Numerical results are presented using the following performance metrics: average time, speedup, speedup against the TSRK5DP solver, parallel efficiency, and parallel cost metrics.

The **average computation time** (t_a) is defined as follows:

$$t_a \text{ [s]} = \frac{\sum_{i=1}^{n_R} t}{n_R}, \tag{24}$$

where n_R denotes the total number of runs.

The **speedup** (s) of the solver is defined as follows:

$$s = \frac{t_1}{t_N}, \tag{25}$$

where t_1 denotes the serial computation time of one compute node and t_N denotes the parallel computation time of n_N compute nodes. When $s > 1$, the speedup metric conveys performance improvement. On the contrary, when $s < 1$, the speedup metric conveys performance degradation. The ideal speedup (ideal scaling) is defined as $s = n_N$. The superlinear speedup is achieved when $s > n_N$.

The **speedup-against** (s_a) ratio of the solver is defined as follows:

$$s_a = \frac{t_{a1}}{t_{a2}}, \tag{26}$$

where t_{a1} denotes the average time of computation of the reference solver TSRK5DP against t_{a2} which denotes the calculation time using one of the solvers in Table 1.

$s_a \gg 1$ indicates a significantly faster computation time using the given solver.

The **parallel efficiency** (e_N) is defined as follows:

$$e_N \text{ [\%]} = \frac{s}{n_N} \cdot 100 = \frac{t_1}{t_N n_N} \cdot 100, \tag{27}$$

The **parallel cost** (c_N) of an algorithm is defined as follows:

$$c_N \text{ [node-hours]} = \frac{t_a}{3{,}600} \cdot n_N, \tag{28}$$

where $t_a/3{,}600$ is the average time in hours.

The **parallel cost ratio** (c_R) is defined as follows:

$$c_R = \frac{c_{N1}}{c_{N2}}, \tag{29}$$

where c_{N1} is the parallel cost for one compute node and c_{N2} is the parallel cost for more than one compute node. Parallel cost is calculated using (28).

The **parallel speedup-cost ratio** (s_R) is defined as follows:

$$s_R = \frac{s}{c_R}, \tag{30}$$

where s is calculated by (25) and c_R by (29).

4.2 Numerical results

Selected problem sizes (denoted as S) are 64,000, 128,000, 256,000, 512,000, and 1,024,000 second-order ODEs. In this section, results for $S = 256{,}000$ will be introduced in detail. Since the wave equation is a second-order PDE, it is necessary to reduce the order of derivatives to obtain the system of the first-order ODEs. Therefore, the number of first-order ODEs in (17) is two times bigger, that is, $2S - 2$ (Table 2). For simplicity, $ne = 2S$ denotes the problem size including two Dirichlet boundary conditions (14).

In the final part of this section, results for other problem sizes will be discussed. The average order of MTSM is 19. The density of matrices $\mathbf{A}$ and $\hat{\mathbf{A}}$ (23) is 3.91×10^{-4}% and 9.96×10^{-3}%, respectively.

The number of integration steps and the average step sizes for each solver are shown in Table 3. The MTSM uses a step size approximately 7.8 times larger than the TSRK5DP solver and approximately 3.2 times larger than the TSRK8VR solver.

Table 3: Number of integration steps, average step sizes

Solver	# steps	Average h
MTSM_PRECALC	10,000	4.00×10^{1}
MTSM	10,000	4.00×10^{1}
TSRK5DP	78,238	5.11×10^{2}
TSRK8VR	32,000	1.26×10^{1}

Figure 2 visualize the performance metrics introduced in Section 4.1 for all selected solvers. The results for MTSM_PRECALC are marked in red, for MTSM in blue, for TSRK5DP in green, and for TSRK8VR in black. Figure 2(a) shows the average computation time (24) where the dashed lines show the ideal average times for the given number of processes. Figure 2(b) shows the parallel efficiency (27). Figure 2(c) depicts the speedup for each solver (25), and the dashed line shows the ideal speedup for the given number of processes. Figure 2(d) shows the speedup ratio with respect to the TSRK5DP solver (26), and $s \gg 1$ indicates significantly faster computation using the given solver defined in Table 1. We can clearly see that the MTSM_PRECALC solver overperforms the state-of-the-art solvers. Only

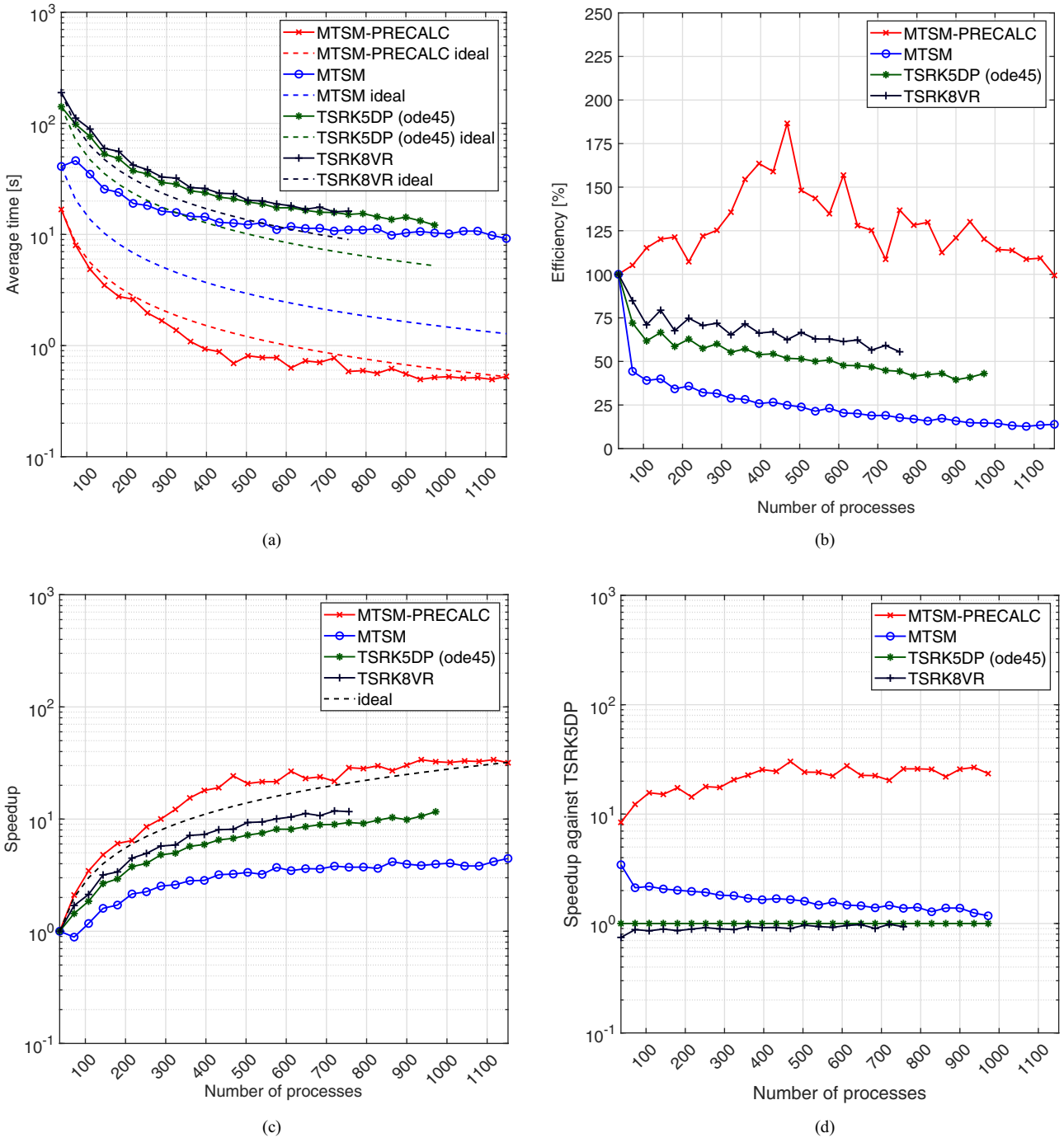

Figure 2: Performance metrics: average time, parallel efficiency, parallel speedup, and speedup against the TSRK5DP solver, $ne = 512{,}000$.
(a) Average time and (b) parallel efficiency, (c) parallel speedup, and (d) speedup against the TSRK5DP solver.

the MTSM and MTSM_PRECALC solvers can provide results for all 1–32 compute nodes.

Figure 3 shows the parallel cost metrics for each solver. Each subfigure contains three curves. The curve labelled with the solver's name shows the parallel speedup of a given solver (25). The magenta curve represents the parallel cost ratio (29), and the cyan curve shows the parallel speedup-cost tradeoff (30). Figure 3(a) shows that the ideal number of compute nodes for the MTSM_PRECALC solver is 32 (1152 MPI processes) because the speedup (red curve) increases proportionally with the parallel cost ratio (magenta curve). Therefore, the speedup-cost ratio (cyan

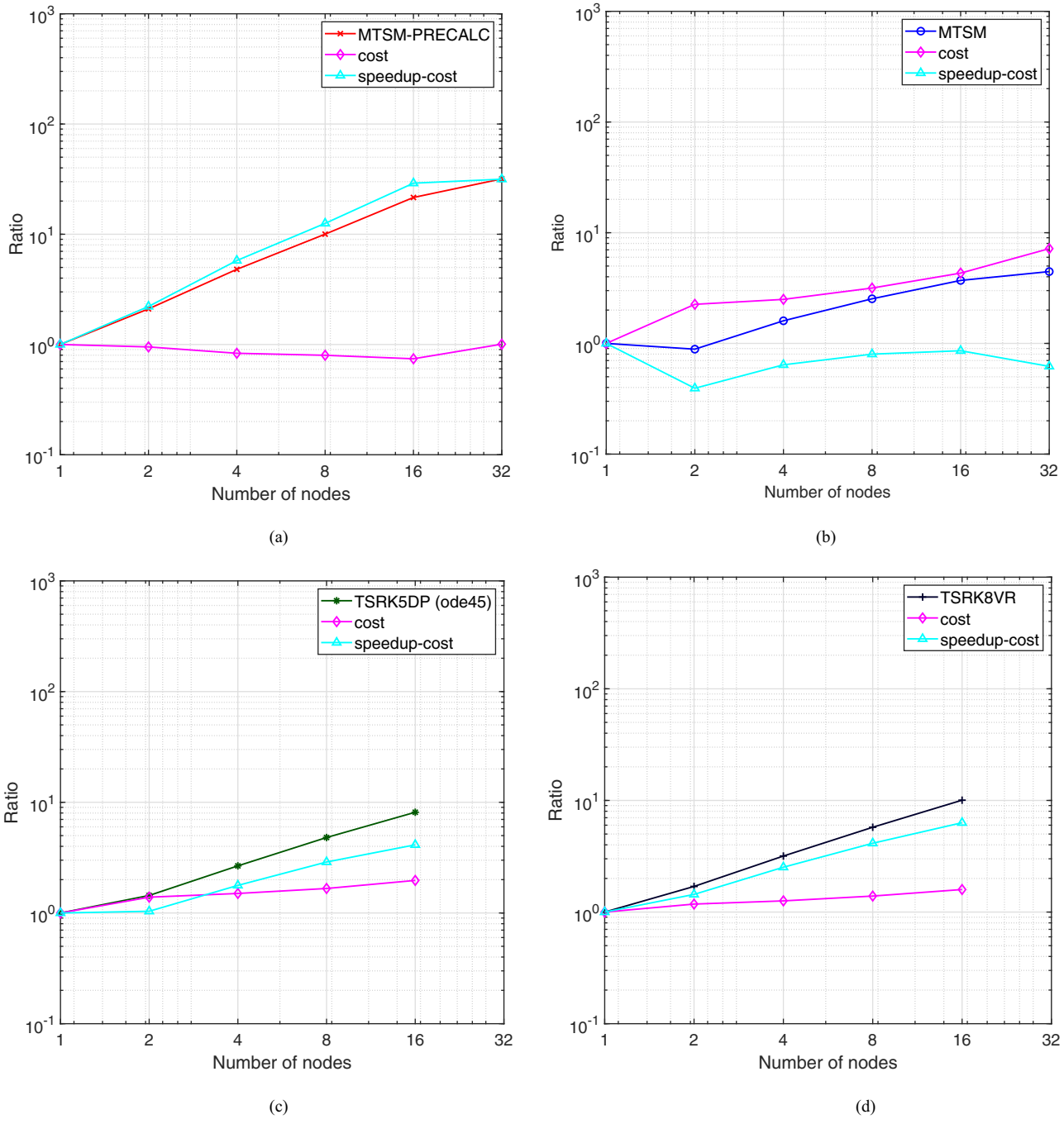

Figure 3: Parallel cost metrics: speedup, parallel cost and parallel speedup-cost ratio for each solver, $ne = 512{,}000$. (a) Parallel speedup-cost tradeoff, MTSM_PRECALC, (b) parallel speedup-cost tradeoff, MTSM, (c) parallel speedup-cost tradeoff, TSRK5DP, and (d) parallel speedup-cost tradeoff, TSRK8VR.

Table 4: Average time, $ne = 512{,}000$

Solver	# compute nodes (t_a) [s]								
	1	**4**	**8**	**12**	**16**	**20**	**24**	**28**	**32**
MTSM_PRECALC	16.78	3.49	1.67	0.88	0.78	0.77	0.62	0.52	0.53
MTSM	40.96	25.58	16.19	12.82	11.05	10.75	9.83	10.13	9.19
TSRK5DP	141.11	52.95	29.36	21.63	17.36	15.74	13.64	—	—
TSRK8VR	189.06	59.53	32.87	23.51	18.81	15.98	—	—	—

Table 5: Yes/No table

Solver	Problem size (ne)				
	128,000	**256,000**	**512,000**	**1,024,000**	**2,048,000**
MTSM_PRECALC	Yes	Yes	Yes	Yes	Yes
MTSM	Yes	Yes	Yes	Yes	No (6)
TSRK5DP	Yes	Yes	No (27)	No (4)	—
TSRK8VR	Yes	Yes	No (21)	No (2)	—

Table 6: Average efficiency (e_N) comparison for 1–32 nodes

Solver	Problem size (ne)				
	128,000	**256,000**	**512,000**	**1,024,000**	**2,048,000**
MTSM_PRECALC	**51.10**	**77.41**	**127.64**	**136.43**	**119.31**
MTSM	15.30	19.77	25.62	33.67	—
TSRK5DP	25.91	29.86	—	—	—
TSRK8VR	29.76	40.24	—	—	—

The underlined values indicate much faster calculation than the state-of-the-art numerical solvers.

Table 7: Average speedup against TSRK5DP comparison for 1–32 nodes

Solver	Problem size (ne)	
	128,000	**256,000**
MTSM_PRECALC	**16.04**	**20.90**
MTSM	**1.04**	**1.63**
TSRK8VR	0.97	**1.23**

The underlined values indicate much faster calculation than the state-of-the-art numerical solvers.

curve) also increases. MTSM, TSRK5DP, and TSRK8VR solvers are ideal to use 16 nodes (576 MPI processes), see Figures 3(b), (c), and (d), respectively.

Tables 4–7 summarise results for wave equation discretised in the spatial domain using the three-point central difference formula for each problem size ne, namely, 128,000, 256,000, 512,000, 1,024,000, and 2,048,000 ODEs. These results are averages of values for 1–32 compute nodes.

Table 4 shows the average computation times for a selected number of processes, that is, for 1, 4, 8, 12, 16, 20, 24, 28, and 32 compute nodes. The MTSM_PRECALC is the fastest of all solvers.

Table 5 indicates whether a solver calculates ("Yes") or does not calculate ("No") the result for 1–32 nodes for a given problem size. The notation "No (X)" implies that a given solver calculated the result for the maximum number of compute nodes denoted as X.

Notice that for the problem size greater than $ne = 256{,}000$, the TSRK5D and TSRK8VR solvers did not calculate the result for all 1–32 compute nodes because the maximum walltime (15 min) was exceeded. For example, for $ne = 1{,}024{,}000$, the TSRK5DP and TSRK8VR did not compute the results for all 32 compute nodes and were able to use four and two nodes

maximally, respectively. Similarly, the MTSM did not calculate results for problem sizes greater than ne = 1,024,000. Only the MTSM_PRECALC solver calculated results for all problem sizes for all 1–32 nodes.

The cells in Table 6 with average parallel efficiency (27) greater than or equal to 50% are in bold. The MTSM_PRECALC solver offers an efficiency greater than 50% for all problem sizes.

The cells in Table 7 showing the speedup ratio (26) with respect to the TSRK5DP solver where $s_a \gg 1$ is also marked in bold. The MTSM_PRECALC solver is always faster than the TSRK5DP solver.

Runge-Kutta solvers did not provide results for $ne >$ 256,000 ODEs for all 1–32 compute nodes (Table 5), for that reason, Table 7 has only two columns.

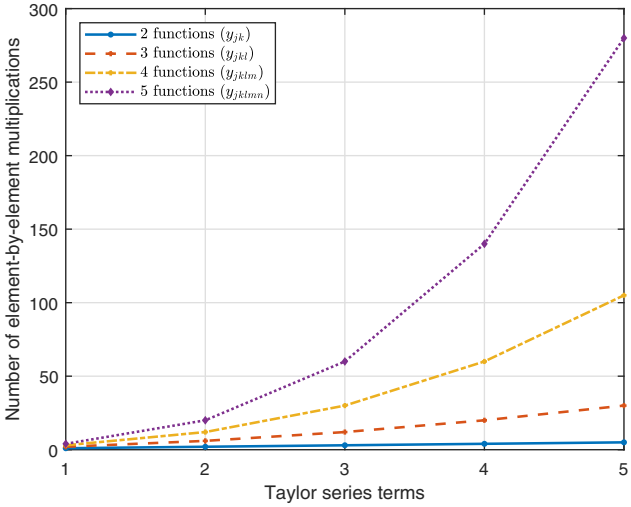

Figure 4: Number of multiplications.

5 Nonlinear problems and optimisations

For nonlinear problems, IVP (1) has the following form:

$$\boldsymbol{y}' = \mathbf{A}\boldsymbol{y} + \mathbf{B}_1\boldsymbol{y}_{jk} + \mathbf{B}_2\boldsymbol{y}_{jkl} + \ldots + \mathbf{b}, \quad \boldsymbol{y}(0) = \boldsymbol{y}_0, \tag{31}$$

where $\mathbf{A} \in \mathbb{R}^{ne\times ne}$ is the constant matrix for the linear part of the system (Section 3), matrices $\mathbf{B}_1 \in \mathbb{R}^{ne\times nm_{jk}}$, $\mathbf{B}_2 \in \mathbb{R}^{ne\times nm_{jkl}}$ are the constant matrices for nonlinear part of the system. The vector $\mathbf{b} \in \mathbb{R}^{ne}$ is the right-hand side for the forces incoming to the system, $\boldsymbol{y}_0$ is a vector of the initial conditions, and symbol ne stands for the number of equations of the system of ODEs. Symbols nm_{jk} and nm_{jkl} represent the number of two and three-functions multiplications, respectively.

The unknown function $\boldsymbol{y}_{jk} \in \mathbb{R}^{nm_{jk}}$ represents the vector of two-terms multiplications $\boldsymbol{y}_j \odot \boldsymbol{y}_k$ and similarly $\boldsymbol{y}_{jkl} \in \mathbb{R}^{nm_{jkl}}$ represents the vector of three-terms multiplications $\boldsymbol{y}_{jj} \odot \boldsymbol{y}_{kk} \odot \boldsymbol{y}_{ll}$, where indices $j,k,jj,kk,ll \in (1,\ldots,ne)$ come from multiplications terms in (31). The operation $\odot$ stands for *element-by-element* multiplication, i.e. $\boldsymbol{y}_j \odot \boldsymbol{y}_k$ is a vector $(y_{j_1}y_{k_1}, y_{j_2}y_{k_2}, \ldots, y_{j_{nm_{jk}}}y_{k_{nm_{jk}}})^T$. For simplification, the matrices $\mathbf{A}, \mathbf{B}_1, \mathbf{B}_2, \ldots$ and the vector $\mathbf{b}$ are constant. The higher derivatives of the terms $\mathbf{B}_1\boldsymbol{y}_{jk}$, $\mathbf{B}_2\boldsymbol{y}_{jkl}$ in (31) can be included in a recurrent calculation of the Taylor series terms $\boldsymbol{p}_{B1}$ and $\boldsymbol{p}_{B2}$

$$\begin{aligned}
&\boldsymbol{p}(1)_A = h(\mathbf{A}\boldsymbol{y}_i + \mathbf{b}), \quad \boldsymbol{p}(1)_{B1} = h(\mathbf{B}_1\boldsymbol{y}_{jk}),\\
&\boldsymbol{p}(1)_{B2} = h(\mathbf{B}_2\boldsymbol{y}_{jkl}), \quad \boldsymbol{p}(r)_A = \frac{h}{r}\mathbf{A}\boldsymbol{p}(r-1),\\
&\boldsymbol{p}(r)_{B1} = \frac{h}{r}\left(\mathbf{B}_1\sum_{a=1}^{r}\boldsymbol{p}(a-1)_j \odot \boldsymbol{p}(r-a)_k\right),\\
&\boldsymbol{p}(r)_{B2} = \frac{h}{r}\mathbf{B}_2\sum_{a=0}^{r-1}\boldsymbol{p}(a)_{jj} \odot \left(\sum_{b=1}^{r-a}\boldsymbol{p}(b-1)_{kk} \odot \boldsymbol{p}(r-a-b)_{ll}\right),
\end{aligned} \tag{32}$$

where $r = 2,\ldots,n$. Finally, the Taylor series terms are calculated as a sum of linear and nonlinear terms

$$\boldsymbol{p}(s) = \boldsymbol{p}(s)_A + \boldsymbol{p}(s)_{B1} + \boldsymbol{p}(s)_{B2}, \quad s = 1,\ldots,n, \tag{33}$$

where r and s are the current indexes of the Taylor series terms, a and b are the auxiliary indexes for the summation of two and three-terms multiplications in the nonlinear part of the Taylor series, and $\boldsymbol{p}(s)_A$ is the linear term computed using the recurrent calculation for linear systems. The next value of the function can be calculated using

$$\boldsymbol{y}_{i+1} = \boldsymbol{p}(0)_i + \boldsymbol{p}(1)_i + \boldsymbol{p}(2)_i + \ldots + \boldsymbol{p}(n_i), \tag{34}$$

where $\boldsymbol{p}(0)_i$ is the value of the function $\boldsymbol{y}_i$, $\boldsymbol{p}(1)_i$, ..., $\boldsymbol{p}(n)_i$ are the Taylor series terms calculated using (33). Multiplication terms of the Taylor series for more multiplications $\boldsymbol{p}_{B3}$, $\boldsymbol{p}_{B4}$, ... can be calculated recurrently in a similar way. More information can be found in the study by Veigend et al. [21].

The important fact to consider is the number of element-by-element multiplications to calculate the solution. The number of element-by-element multiplications used in (32) is visualised in Figure 4.

Due to the fact that the number of element-by-element multiplications increases rapidly when more functions are multiplied, the calculation of higher derivatives has to be optimised. Several optimisations are discussed in this article.

5.1 Auxiliary generating equations

This optimisation is used automatically when using the presented method. However, the equations that contain terms with many function multiplications (the example

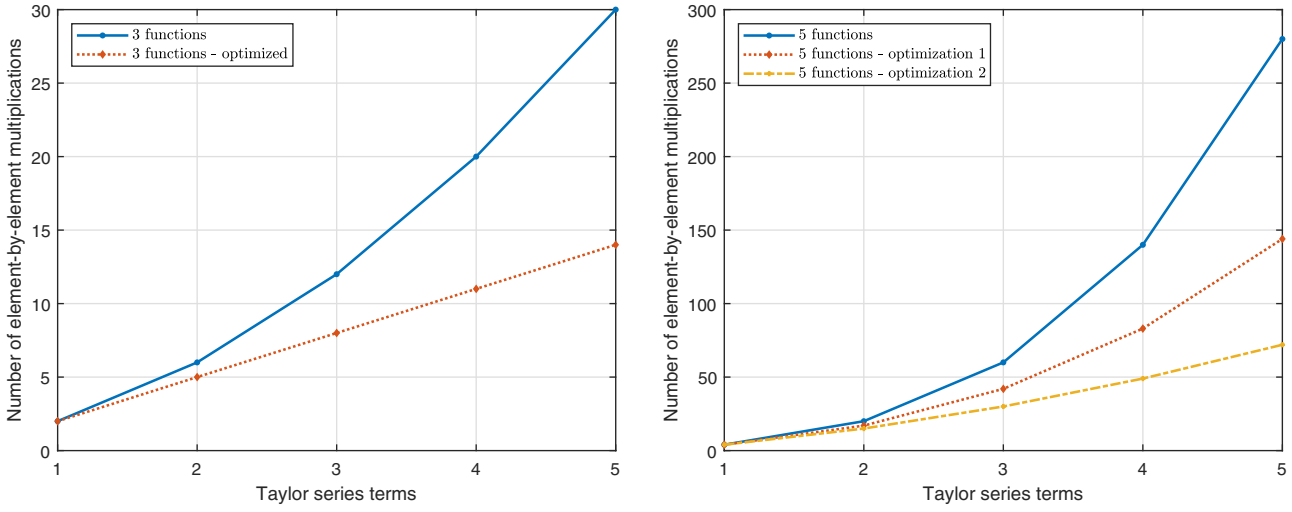

Figure 5: Comparison of a number of element-by-element multiplications ($\odot$) with and without optimisations. (a) Three function multiplications and (b) Five function multiplications.

behaviour for five multiplication is in Figure 5(b)) can be simplified by applying the generating process again. The newly generated equation contains several terms and therefore decreases the number of function multiplications in the original term.

5.2 Partly calculated results of the previous term

This optimisation is universal and can be used for any system. Consider Table 8, which contains the Taylor series terms used when calculating higher derivatives. The coloured areas when calculating the Taylor series terms can be saved and used when calculating further terms.

The optimisation becomes more pronounced for more function multiplications, and it can be performed multiple times. This is shown in Table 9, which shows partial results being saved and used. Both partial (darker colours) and full results (lighter colours) are saved and used.

The number of performed function multiplications with and without optimisations is shown in Figure 5(a) and (b).

The figures shows that the number of operations decreases quite dramatically compared to the case without any optimisations (Figure 4), and it is almost linear.

Table 8: Three function multiplications

n	Indexes of Taylor series terms														
1	0														
	0														
	0														
2	1	0	0												
	0	1	0												
	0	0	1												
3	2	1	1	0	0	0									
	0	1	0	2	1	0									
	0	0	1	0	1	2									
4	3	2	2	1	1	1	0	0	0	0					
	0	1	0	2	1	0	3	2	1	0					
	0	0	1	0	1	2	0	1	2	3					
5	4	3	3	2	2	2	1	1	1	1	0	0	0	0	0
	0	1	0	2	1	0	3	2	1	0	4	3	2	1	0
	0	0	1	0	1	2	0	1	2	3	0	1	2	3	4

Table 9: Five function multiplications – partial and full results

n	Indexes of Taylor series terms														
1	0														
	0														
	0														
	0														
	0														
2	1	0	0	0	0										
	0	0	1	0	0										
	0	0	0	1	0										
	0	0	0	0	1										
	0	1	0	0	0										
3	2	1	1	1	1	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	1	0	0	2	1	1	0	0	0
	0	0	1	0	0	0	0	1	0	0	1	0	2	1	0
	0	0	0	1	0	0	0	0	1	0	0	1	0	1	2
	0	0	0	0	1	2	1	1	1	0	0	0	0	0	0

5.3 Variable step size

When solving nonlinear problems, the P function can oscillate quite rapidly. For some nonlinear problems, however, it rapidly changes near the beginning of the calculation and then stabilises. When this happens, the method often calculates with a relatively small integration step. Due to the fact that the method can automatically adjust the value of P in the current step based on the size of the step, the size can be dynamically increased to decrease the overall number of operations the method has to perform.

To work with this optimisation, the constant h_{scale} is defined as the *scaling factor* for the size of integration step h

$$h_{\text{new}} = h_{\text{scale}} \cdot h.$$

The new value for the size of the integration step h_{new} is used until the calculation ends. The scaling factor is only applied when

$$\sum_{a=i-3}^{i} P(a) \leq P_{\min} \cdot 3, \quad i = 4 \ldots n_T,$$

where n_T is the number of time steps and $P_{\min}$ is the value of the P function that has to be kept for *three* integration steps. This approach is useful for problems where the previous approach cannot be used (i.e. for systems that only contain two function multiplications). These optimisations and their advantages and disadvantages are discussed on a set of benchmarks in Section 6.

5.4 Further optimisations

Additional optimisations are possible and are being actively studied. One of the approaches that is currently being tested limits the number of multiplications to two by expressing the Taylor series terms using systems of differential-algebraic equations (DAEs). The results are promising and are going to be published at a later date.

6 Nonlinear problem examples

The previously mentioned optimisations are tested using the selected set of nonlinear nonstiff problems presented in [36]. The selected benchmark problems are a subset of a full benchmark collection, which is intended to show the potential strengths and weaknesses of newly developed numerical methods.

The selected problems are going to be analysed and then solved using the nonlinear MTSM solver with or without the optimisations.

Note that the systems of ODEs have to be always transformed into the autonomous system of homogeneous ODEs without division and with just basic arithmetic operations [21].

Experiments were repeated 100 times. The **median computation time** (t_m) was calculated from the computation times during the individual runs of the solvers. The **speedup-against** (s_a) ratio of the solver is defined as follows:

$$s_a = \frac{t_{m1}}{t_{m2}}, \tag{35}$$

where t_{m1} denotes the median calculation time using one of the MATLAB ode solvers from Table 10 against t_{m2} which denotes the median calculation time using of the MTSM solvers listed in the same table.

The $s_a \gg 1$ indicates a significantly faster computation time using the MTSM solvers.

The error (err) is calculated as norm of the difference between numerical solution MTSM solvers and MATLAB ode solvers at the time T

$$\text{err} = \|\mathbf{y}_{M(T)} - \mathbf{y}_{O(T)}\|,$$

where $\mathbf{y}_{M(T)}$ is the value calculated by the MTSM solver at the time T and $\mathbf{y}_{O(T)}$ is the value calculated by the ode solver at the time T. Table 10 lists the state-of-the-art solvers that are part of MATLAB and implemented MTSM solvers for MATLAB used in this section.

The parameters for all used solvers are summarised in Table 11.

Table 10: MATLAB numerical methods

Solver	Method
ode23	Bogacki-Shampine [37]
ode45	Dormand-Prince [26]
ode113	Adams-Bashfort method with predictor-corrector PECE scheme
MTSM orig	Nonlinear MTSM solver without optimisations
MTSM opt	Nonlinear MTSM solver with optimisations from Section 5

Table 11: Parameters for nonlinear simulation experiments

Parameter	Value
MTSM accuracy	$\varepsilon = 1 \times 10^{-9}$
Relative, absolute tolerances	$\text{rel}_{\text{TOL}} = \text{abs}_{\text{TOL}} = 1 \times 10^{-9}$
Maximum simulation time	$T = 20$ (s)
Maximum order for MTSM solvers	$P_{\max} = 64$

Table 12: Results for problem A2, h = 0.5 a, h_{scale} = 5, $P_{\min}$ = 9

			MTSM orig		MTSM opt	
Solver	**# of steps**	**Time of calculation [s]**	**err**	**Ratio (s_a)**	**err**	**Ratio (s_a)**
MTSM orig	40	0.00152015	—	—	—	—
MTSM opt	16	0.0007533	—	—	—	—
ode23	28,550	0.168798	1.67406×10^{-7}	**114.25**	2.19986×10^{-7}	**230.1**
ode45	2,177	0.0040288	1.67407×10^{-7}	**2.73**	2.19987×10^{-7}	**5.49**
ode113	232	0.0031412	1.67406×10^{-7}	**2.13**	2.19986×10^{-7}	**4.28**

6.1 Problem A2

Problem A2 is defined as follows:

$$y' = \frac{-y^3}{2} = -0.5y^3 \quad y(0) = 1.$$

To solve this nonlinear system using MTSM, the term y^3 has to be replaced by a set of auxiliary ODEs

$$\begin{aligned}
y_1' &= -0.5y_1^3 && y_1(0) = 1\\
y_2 &= y_1^3 &&\\
y_2' &= 3y_1^2 y_1' = 3y_1^2(-0.5y_1^3) = -1.5y_1^2 y_2 && y_2(0) = y_1(0)^3\\
y_3 &= y_1^2 &&\\
y_3' &= 2y_1 y_1' = 2y_1(-0.5y_1^3) = -y_1 y_2 && y_3(0) = y_1(0)^2,
\end{aligned}$$

so the final system becomes

$$\begin{aligned}
y_1' &= -0.5y_2 && y_1(0) = 1\\
y_2' &= -1.5y_2 y_3 && y_2(0) = y_1(0)^3\\
y_3' &= -y_1 y_2 && y_3(0) = y_1(0)^2,
\end{aligned}$$

which can be transformed into a matrix-vector representation (31)

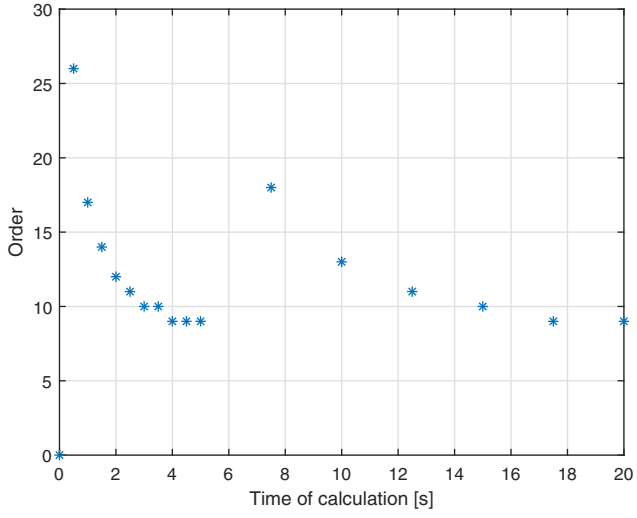

Figure 6: Order function P for problem A2, h = 0.5 s, h_{scale} = 5, $P_{\min}$ = 9.

$$\mathbf{A} = \begin{pmatrix} 0 & -0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \mathbf{B}_1 = \begin{pmatrix} 0 & 0 \\ -1.5 & 0 \\ 0 & -1 \end{pmatrix} \quad \boldsymbol{y}_{jk} = \begin{pmatrix} y_2 y_3 \\ y_1 y_2 \end{pmatrix}.$$

The numerical results are presented in Table 12. MTSM solvers outperform the state-of-the-art solvers.

Optimisations in Sections 5.1 and 5.3 are used. The automatic change in step size at t = 5 s is shown in Figure 6.

6.2 Problem B4

This problem is interesting because it contains nontrivial mathematical operations that have to be replaced using auxiliary equations. The problem is defined as follows:

$$\begin{aligned}
y_1' &= -y_2 - \frac{y_1 y_3}{\sqrt{y_1^2 + y_2^2}} && y_1(0) = 3\\
y_2' &= y_1 - \frac{y_2 y_3}{\sqrt{y_1^2 + y_2^2}} && y_2(0) = 0\\
y_3' &= \frac{y_1}{\sqrt{y_1^2 + y_2^2}} && y_3(0) = 0.
\end{aligned}$$

After generating auxiliary equations and decreasing the number of function multiplications (optimisation from Section 5.1), the system of ODEs can be written as follows:

$$\begin{aligned}
y_1' &= -y_2 - y_{11}\\
y_2' &= y_1 - y_2 y_3 y_5\\
y_3' &= y_1 y_5\\
y_4' &= -2y_3 y_8 y_9 - 2y_3 y_8 y_{10}\\
y_5' &= 2y_3 y_6 y_9 + 2y_3 y_6 y_{10}\\
y_6' &= 8y_3 y_6 y_7 y_9 + 8y_3 y_6 y_7 y_{10}\\
y_7' &= 6y_3 y_6 y_8 y_9 + 6y_3 y_6 y_8 y_{10}\\
y_8' &= 4y_3 y_5 y_6 y_9 + 4y_3 y_5 y_6 y_{10}\\
y_9' &= -2y_9 - 2y_9 y_{11}\\
y_{10}' &= 2y_1 y_2 - 2y_3 y_5 y_{10}\\
y_{11}' &= -y_2 y_3 y_5 - y_3 y_5 y_{11} + y_8 y_9 + 2y_1 y_6 y_9 y_{12} + 2y_1 y_6 y_{10} y_{12}\\
y_{12}' &= 2y_{11}
\end{aligned} \tag{36}$$

Table 13: Results for problem B4, h = 0.5 s, h_{scale} = 2.5, and P_{min} = 12

			MTSM orig		MTSM opt	
Solver	**# of steps**	**Time [s]**	**err**	**Ratio (s_a)**	**err**	**Ratio (s_a)**
MTSM orig	40	9.8158×10^{-3}	—	—	—	—
MTSM opt	19	6.6631×10^{-3}	—	—	—	—
ode23	282,708	1.76957	7.45931×10^{-7}	**180.28**	8.39871×10^{-7}	**265.58**
ode45	11,773	2.31782×10^{-2}	7.45942×10^{-7}	**2.36**	8.39869×10^{-7}	**3.48**
ode113	468	6.847×10^{-3}	7.45943×10^{-7}	0.7	8.39869×10^{-7}	**1.03**

with initial conditions $\boldsymbol{y}(0) = (3, 0, 0, y_1(0)^2 + y_2(0)^2, \frac{1}{y_4(0)}, y_5(0)^4, y_5(0)^3, y_5(0)^2, y_1(0)^2, y_2(0)^2, y_1(0)y_3(0)y_5(0), y_3(0)^2)^T$
The matrix-vector representation of system of ODEs (36) using the notation from (32) follows:

- the 12 × 12 matrix $\mathbf{A}$ has five nonzero elements $\mathbf{A}(1, 2) = -1$, $\mathbf{A}(1, 11) = -1$, $\mathbf{A}(2, 1) = 1$, $\mathbf{A}(9, 9) = -2$ and $\mathbf{A}(12, 11) = 2$,
- the 12 × 4 matrix $\mathbf{B}_1$ has four nonzero elements $\mathbf{B}_1(3, 1) = 1$, $\mathbf{B}_1(9, 2) = -2$, $\mathbf{B}_1(10, 3) = 2$, $\mathbf{B}_1(11, 4) = 1$,
- the 12 × 8 matrix $\mathbf{B}_2$ has eight nonzero elements $\mathbf{B}_2(2, 1) = -1$, $\mathbf{B}_2(4, 2) = -2$, $\mathbf{B}_2(4, 3) = -2$, $\mathbf{B}_2(5, 4) = 2$, $\mathbf{B}_2(6, 5) = 2$, $\mathbf{B}_2(10, 6) = -2$, $\mathbf{B}_2(11, 7) = -1$ and $\mathbf{B}_2(11, 8) = -1$,
- the 12 × 8 matrix $\mathbf{B}_3$ has eight nonzero elements $\mathbf{B}_3(6, 1) = 8$, $\mathbf{B}_3(6, 2) = 8$, $\mathbf{B}_3(7, 3) = 6$, $\mathbf{B}_3(7, 4) = 6$, $\mathbf{B}_3(8, 5) = 4$, $\mathbf{B}_3(8, 6) = 4$, $\mathbf{B}_3(11, 7) = 2$ and $\mathbf{B}_3(11, 8) = 2$,
- two-term multiplications $\boldsymbol{y}_{jk} = (y_1y_5, y_9y_{11}, y_1y_2, y_8y_9)^T$,
- three-term multiplications $\boldsymbol{y}_{jkl} = (y_2y_3y_5, y_3y_8y_9, y_3y_8y_{10}, y_3y_6y_9, y_3y_6y_{10}, y_3y_5y_{10}, y_2y_3y_5, y_3y_5y_{11})^T$,
- four-term multiplications $\boldsymbol{y}_{jklm} = (y_3y_6y_7y_9, y_3y_6y_7y_{10}, y_3y_6y_8y_9, y_3y_6y_8y_{10}, y_3y_5y_6y_9, y_3y_5y_6y_{10}, y_1y_6y_9y_{12}, y_1y_6y_{10}y_{12})^T$,
- vector representing the right-hand side of the system $\mathbf{b} = \mathbf{0}$.

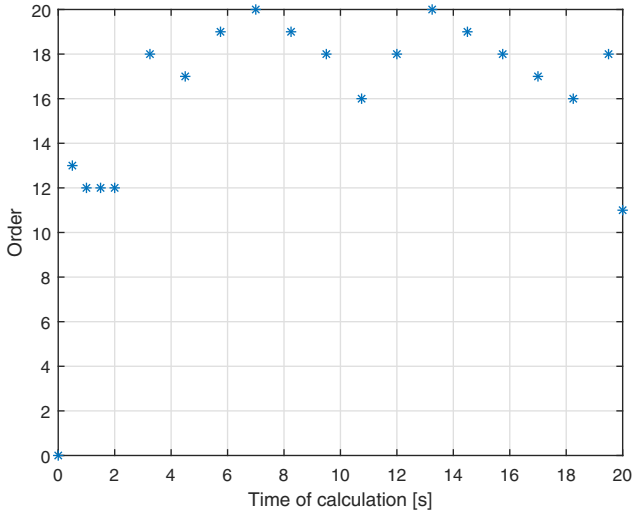

Figure 7: Order function P for benchmark problem B4, h = 0.5 s, h_{scale} = 2.5, P_{min} = 12.

The numerical results are in Table 13. Note that optimisations 5.2 and 5.3 are also used.

The plot of the order function P with step size scaling factor set to h_{scale} = 2.5 and P_{min} = 12 is in Figure 7. The automatic change in step size is visible at t = 2 s.

The results show that the optimisations have a profound impact on the performance of the method and are beneficial. Presented optimisations can be freely combined to improve performance without any loss in accuracy.

7 Conclusion

The article dealt with the numerical solution of both linear and nonlinear IVPs represented by a system of ODEs. The MTSM outperforms the state-of-the-art MATLAB and PETSc ODE solvers in all cases. For linear problems, the wave equation, represented by large systems of ODEs, was solved in parallel using the method of lines. For nonlinear problems, optimisations were presented and demonstrated on the set of benchmark problems.

Our research is going to focus on effective solution of nonlinear systems of DAEs using the presented method.

Acknowledgement: The article includes the results of the internal BUT FIT project FIT-S-23-8151. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254). The work extends the conference article [23] (DOI: 10.1109/Informatics57926.2022.10083462).

Funding information: The authors state that no funding is involved.

Author contributions: All authors have accepted responsibility for the entire content of this manuscript and approved its submission.

Conflict of interest: Authors state no conflict of interest.

Data availability statement: Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

References

[1] J. Kunovský, *Modern Taylor Series Method*, Habilitation work, Faculty of Electrical Engineering and Computer Science, Brno University of Technology, 1994.

[2] J. Kunovský, Tksl. online, https://www.fit.vut.cz/research/product/51/.en.

[3] Matlab and simulink software, 2017, http://www.mathworks.com.

[4] F. Kocina, Fos. online, http://www.fit.vutbr.cz/iveigend/fos.

[5] V. Šátek, *High performance computing research group, online*, https://www.fit.vut.cz/research/group/hpc/.en.

[6] F. Kocina, J. Kunovský, G. Nečasová, V. Šátek, and P. Veigend, "Parallel solution of higher order differential equations," In *Proceedings of the 2016 International Conference on High Performance Computing & Simulation (HPCS 2016)*, *Institute of Electrical and Electronics Engineers*, pp. 302–309, 2016.

[7] R. Barrio, M. Rodríguez, A. Abad, and F. Blesa, "TIDES: A free software based on the Taylor series method," *Monografías de la Real Academia de Ciencias de Zaragoza*, vol. 35, pp. 83–95, 2011.

[8] A. Jorba and M. Zou, "A software package for the numerical integration of ODE by means of high-order Taylor methods," *Exp. Math.*, vol. 14, pp. 99–117, 2005.

[9] Y. F. Chang and G. Corliss, "ATOMF: Solving ODEs and DAEs using Taylor series," *Computers & Mathematics with Applications*, vol. 28, pp. 209–233, 1994.

[10] M. Berz, *COSY infinity version 8 Reference Manual*, Technical Report MSUCL-1088, National Superconducting Cyclotron Lab., Michigan State University, East Lansing, Mich., 1997.

[11] N. S. Nedialkov and J. Pryce, "Solving differential algebraic equations by Taylor series III. the DAETS code," *JNAIAM J. Numer. Anal. Ind. Appl. Math*, vol. 3, pp. 61–80, 2008.

[12] R. Barrio, F. Blesa, and M. Lara, "VSVO formulation of the Taylor method for the numerical solution of ODEs," *Computers and Mathematics with Applications*, vol. 50, pp. 93–111, 2005.

[13] R. Barrio, "Performance of the Taylor series method for ODEs/DAEs," In: *Applied Mathematics and Computation*, vol. 163, pp. 525–545, 2005. ISSN 00963003.

[14] P. Mohazzabi and J. L. Becker, "Numerical solution of differential equations by Direct Taylor expansion," *Journal of Applied Mathematics and Physics*, vol. 5, no. 3, 623–630, 2017.

[15] A. Baeza, S. Boscarino, P. Mulet, G. Russo, and D. Zorío, "Approximate Taylor methods for ODEs," *Computers & Fluids*, vol. 159, pp. 156–166, 2017.

[16] P. Amodio, F. Iavernaro, F. Mazzia, M. S. Mukhametzhanov, and Y. D. Sergeyev, "A generalized Taylor method of order three for the solution of initial value problems in standard and infinity floating-point arithmeticm," *Mathematics and Computers in Simulation*, vol. 141, pp. 24–39, 2017.

[17] S. Dimova, I. Hristov, R. Hristova, I. Puzynin, T. Puzynina, Z. Sharipov, et al., *OpenMP parallelization of multiple precision Taylor series method*, 2019. https://arxiv.org/abs/1908.09301.

[18] S. Liao, "On the reliability of computed chaotic solutions of non-linear differential equations," *Tellus A: Dynamic Meteorology and Oceanography*, vol. 61, no. 4, pp. 550–564, 2008.

[19] I. Hristov, R. Hristova, S. Dimova, P. Armyanov, N. Hegunov, I. Puzynin, et al., "Parallelizing multiple precision Taylor series method forï£¡integrating the Lorenz system," In: I. Georgiev, H. Kostadinov, and E. Lilkova, editors, *Advanced Computing in Industrial Mathematics*, Springer International Publishing, Cham, pp. 56–66, 2023.

[20] I. Hristov, R. Hristova, S. Dimova, P. Armyanov, N. Shegunov, I. Puzynin, et al., "On the efficient parallel computing of long term reliable trajectories for the Lorenz system," arXiv:2101.06682v1, 2021.

[21] P. Veigend, G. Nečasová, and V. Šátek, "Taylor series based numerical integration method," *Open Computer Science*, vol. 11, no. 1, pp. 60–69, 2021.

[22] P. Veigend, G. Nečasová, and V. Šátek, "High order numerical integration method and its applications - the first 36 years of MTSM," In: *2019 IEEE 15th International Scientific Conference on Informatics*, Institute of Electrical and Electronics Engineers, pp. 25–30, 2019.

[23] P. Veigend, G. Nečasová, and V. Šátek, "Taylor series method in numerical integration: Linear and nonlinear problems," In: *2022 IEEE 16th International Scientific Conference on Informatics, Informatics 2022 - Proceedings*, IEEE, pp. 239–244. Communications Society, 2023.

[24] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I*, Nonstiff Problems. Springer-Verlag, Berlin Heidelberg, 1987. ISBN 3-540-56670-8.

[25] G. D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, Oxford University Press, UK, 1985.

[26] J. R. Dormand and P. J. Prince, "A family of embedded runge-kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

[27] J. H. Verner, "Explicit Runge-Kutta methods with estimates of the local truncation error," *Society for Industrial and Applied Mathematics*, vol. 15, pp. 772–790, 1978.

[28] IT4Innovations Documentation. Introduction. online, 2022, https://docs.it4i.cz/barbora/introduction/.

[29] S. Abhyankar, J. Brown, E. M. Constantinescu, D. Ghosh, B. F. Smith, and H. Zhang. *PETSc/TS: A Modern Scalable ODE/DAE Solver Library*, 2018.

[30] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, et al. *PETSc Web page*, 2024. https://www.mcs.anl.gov/petsc.

[31] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, et al. *PETSc Users Manual, Technical Report ANL-95/11 - Revision 3.12*, *Argonne National Laboratory*, 2024, https://www.mcs.anl.gov/petsc.

[32] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object-oriented numerical software libraries," In: E. Arge, A. M. Bruaset, and H. P. Langtangen, editors,

Modern Software Tools in Scientific Computing, Birkhauser Press, Basel, pp. 163–202, 1997.

[33] S. Hamdi, W. E. Schiesser, and G. W. Griffiths, "Method of lines," *Scholarpedia*, vol. 2, no. 7, pp. 1–11, 2007.

[34] R. Vichnevetsky, *Numerical stability of methods of lines for partial differential equations*, Rutgers University, New Jersey, 1971.

[35] A. D. Malony, *Metrics*, Springer US, Boston, MA, pp. 1124–1130, 2011.

[36] W. H. Enright and J. D. Pryce, "Two fortran packages for assessing initial value methods," In: *ACM Transactions on Mathematical Software*, ACM, Mar, vol. 13, pp. 1–27. 1987.

[37] L. F. Shampine and M. W. Reichelt, "The matlab ode suite," *Journal of Numerical Analysis and Applied Mathematics*, vol. 18, no. 1, pp. 1–22, 1997.