



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**

DEPARTMENT OF INTELLIGENT SYSTEMS

**EFEKTIVNÍ ŘEŠENÍ OBYČEJNÝCH DIFERENCIÁLNÍCH  
ROVNIC METODAMI VYŠŠÍCH ŘÁDŮ**

EFFECTIVE SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS USING HIGH ORDER ME-

THODS

**BAKALÁŘSKÁ PRÁCE**

BACHELOR'S THESIS

**AUTOR PRÁCE**

AUTHOR

**VOJTĚCH TEICHMANN**

**VEDOUcí PRÁCE**

SUPERVISOR

**Ing. PETR VEIGEND, Ph.D.**

BRNO 2024

## Zadání bakalářské práce



158229

Ústav: Ústav inteligentních systémů (UITS)  
Student: **Teichmann Vojtěch**  
Program: Informační technologie  
Název: **Efektivní řešení obyčejných diferenciálních rovnic metodami vyšších řádů**  
Kategorie: Modelování a simulace  
Akademický rok: 2024/25

### Zadání:

1. Prostudujte numerické metody využívané k řešení obyčejných diferenciálních rovnic (počátečních úloh).
2. Navrhněte metodu vyššího řádu založenou na Taylorových řadách pro výpočet lineárních a kvadratických soustav obyčejných diferenciálních rovnic.
3. Implementujte tuto metodu efektivně s využitím maticového-vektorového výpočtu v softwarovém prostředí MATLAB.
4. Porovnejte na sadě vhodných příkladů efektivitu vašeho řešení (zaměřte se také na jeho stabilitu, přesnost a rychlost) Vaší metody s ostatními běžně používanými metodami.
5. Zhodnoťte Vaše řešení a navrhněte další možná vylepšení.

### Literatura:

- M. Kubíček, M. Dubcová, D. Janovská. Numerické metody a algoritmy. Vydavatelství VŠCHT Praha, 2005. ISBN 80-7080-558-7.
- L. Čermák and R. Hlavička. Numerické metody. Akademické nakladatelství CERM, s.r.o. Brno, 2005. ISBN 80-214-3071-0.
- L. Čermák. Numerické metody II. Akademické nakladatelství CERM, s.r.o. Brno, 2004. ISBN 80-214-2722-1.
- ŠÁTEK Václav, KOCINA Filip, KUNOVSKÝ Jiří a SCHIRNER Alexander. Taylor Series Based Solution of Linear ODE Systems and MATLAB Solvers Comparison. In: MATHMOD VIENNA 2015 - 8th Vienna Conference on Mathematical Modelling. Vienna: ARGE Simulation News, 2015, s. 693-694. ISBN 978-3-901608-46-9.

Při obhajobě semestrální části projektu je požadováno:  
Body 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz <https://www.fit.vut.cz/study/theses/>

Vedoucí práce: **Veigend Petr, Ing., Ph.D.**  
Vedoucí ústavu: Kočí Radek, Ing., Ph.D.  
Datum zadání: 1.11.2024  
Termín pro odevzdání: 14.5.2025  
Datum schválení: 31.10.2024

## Abstrakt

Tato bakalářská práce se věnuje efektivnímu řešení obyčejných diferenciálních rovnic pomocí metod vyššího řádu založených na Taylorově řadě. Hlavní cíl je navrhnout a implementovat metodu, která bude schopná řešit lineární a kvadratické soustavy ODE s použitím maticového a vektorového výpočtu pomocí softwaru MATLAB. Dále se práce zaměřuje na srovnání této metody s běžnými používanými metodami jako jsou Eulerova nebo Runge-Kuttova metoda co do rychlosti, přesnosti a stability, a to na vybraných příkladech z praxe. Výsledky ukazují že metoda založená na Taylorově řadě má lepší výkon a přesnost než běžné používané metody.

## Abstract

This bachelor thesis is devoted to the efficient solution of ordinary differential equations using higher order methods based on Taylor series. The main objective is to design and make a method that will be able to solve linear and quadratic ODE well and correctly using matrix and vector calculations in MATLAB software. Furthermore, the thesis focuses on the comparison of this method with normally used techniques in terms of speed, accuracy, and stability, using selected examples from technical practice. The results show that the method based on Taylor series has better performance and accuracy than traditional methods such as Euler or Runge-Kutta method.

## Klíčová slova

obyčejné diferenciální rovnice, numerické metody, Taylorova řada, MATLAB, stabilita, přesnost

## Keywords

ordinary differential equations, numerical methods, Taylor series, MATLAB, stability, accuracy

## Citace

TEICHMANN, Vojtěch. *Efektivní řešení obyčejných diferenciálních rovnic metodami vyšších řádů*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Petr Veigend, Ph.D.

# Efektivní řešení obyčejných diferenciálních rovnic metodami vyšších řádů

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Veigenda, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal. Jako podpůrný nástroj při formulaci myšlenek a jazykové úpravě textu jsem využil ChatGPT.

.....

Vojtěch Teichmann

13. května 2025

## Poděkování

Chtěl bych velmi poděkovat svému vedoucímu, Ing. Petrovi Veigendovi, Ph.D., za jeho cenné rady a pomoc při řešení této práce. Dále bych chtěl poděkovat svým rodičům, kteří mě vždy podporovali, a to jak psychicky, tak finančně. Poděkování si zaslouží i moje sestra, která jako čerstvá absolventka Právnické fakulty Univerzity Palackého chápala mé pocity během zkouškového období a podporovala mě. Mockrát děkuji také své přítelkyni, která se mě vždy snažila podržet, když jsem měl špatnou náladu – například když se mi nepovedla nějaká zkouška – a velmi mě podporovala, i když zrovna nemohla být se mnou v Brně. V neposlední řadě děkuji i svým kamarádům, se kterými jsem mohl úspěšně spolupracovat v týmových projektech. Vždy jsme stáli při sobě, protože jsme všichni zažívali podobné pocity.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>6</b>
<b>2</b>	<b>Shrnutí dosavadního stavu</b>	<b>7</b>
2.1	Diferenciální rovnice . . . . .	7
2.1.1	Lineární a nelineární ODE . . . . .	7
2.1.2	Počáteční podmínky . . . . .	8
2.1.3	Analytické vs. numerické řešení . . . . .	8
2.1.4	Explicitní a implicitní metody . . . . .	9
2.2	Problematika přesnosti a stability . . . . .	9
2.2.1	Stabilita numerických metod . . . . .	10
2.2.2	Akumulace chyby . . . . .	11
2.2.3	Přesnost versus výpočetní náročnost . . . . .	12
2.2.4	Shrnutí . . . . .	12
2.3	Přehled numerických metod . . . . .	13
2.3.1	Eulerova metoda . . . . .	13
2.3.2	Metody Runge–Kutta . . . . .	14
2.3.3	Metoda Taylorova rozvoje . . . . .	15
<b>3</b>	<b>Návrh řešení</b>	<b>17</b>
3.1	Maticový přístup k Taylorově metodě . . . . .	17
3.1.1	Rozklad diferenciálních rovnic na soustavu diferenciálních rovnic prvního řádu . . . . .	17
3.1.2	Využití maticové reprezentace pro výpočet vyšších derivací . . . . .	19
3.2	Transformace vstupní úlohy na autonomní soustavu diferenciálních rovnic . . . . .	20
3.2.1	Kvadratické systémy a jejich řešení . . . . .	22
3.2.2	Ukázkový příklad transformace . . . . .	23
<b>4</b>	<b>Implementace a experimentální ověření</b>	<b>25</b>
4.1	Vstupní parametry a inicializace . . . . .	25
4.2	Implementace Taylorovy metody v MATLAB . . . . .	26
4.2.1	Lineární Taylorova metoda . . . . .	26
4.2.2	Nelineární Taylorova metoda . . . . .	27
4.3	Implementace Taylorovy metody v Python . . . . .	27
4.3.1	Původní implementace Taylorovy metody v Pythonu . . . . .	27
4.3.2	Optimalizace pomocí Just-In-Time kompilace . . . . .	28
4.4	Implementace metod pro porovnání nástrojů MATLAB a Python (Euler a Runge-Kutta) . . . . .	28
4.4.1	Eulerova metoda . . . . .	29

4.4.2	Metoda Runge-Kutta 4. řádu . . . . .	29
4.4.3	Implementace pro MATLAB a Python . . . . .	30
4.5	Přehled vestavěných adaptivních řešičů . . . . .	30
4.6	Zabudované řešiče v MATLABu . . . . .	30
4.7	Testovací příklady . . . . .	31
4.7.1	Výsledky lineárního problému (sinus a kosinus) v jazyce Python . . . . .	31
4.7.2	Automatické nastavení řádu (ORD) . . . . .	33
4.7.3	Kruhový test a porovnání metod . . . . .	34
4.7.4	Výpočet koeficientů Fourierovy řady . . . . .	41
4.7.5	Lorenzův systém . . . . .	45
<b>5</b>	<b>Závěr</b>	<b>52</b>
	<b>Literatura</b>	<b>54</b>
<b>A</b>	<b>Obsah paměťového média</b>	<b>57</b>
<b>B</b>	<b>Manuál</b>	<b>59</b>
B.1	Požadavky . . . . .	59
B.2	Spuštění . . . . .	59
B.3	Poznámka . . . . .	60
<b>C</b>	<b>Použité nástroje</b>	<b>61</b>

# Seznam obrázků

2.1	Závislost globální chyby na velikosti kroku pro Eulerovu metodu a RK4 při řešení ODE $y' = -2y, y(0) = 1$ do $t = 5$ . . . . .	10
4.1	Funkce chyby pro $h = 0.01$ s. . . . .	33
4.2	Funkce chyby pro $h = 1$ s. . . . .	34
4.4	ORD funkce pro $MTSM_h = 10$ s. . . . .	38
4.5	Kruhové testy pro adaptivní řešiče ode45, ode78, ode89 a ode113 při $\omega = 100$ rad·s <sup>-1</sup> , $TOL = 1 \times 10^{-3}$ , $MTSM_{TOL} = 1 \times 10^{-6}$ , $h = 0.01$ s. . . . .	39
4.6	Analytické řešení kruhového testu pro $\omega = 100$ rad·s <sup>-1</sup> . . . . .	40
4.7	Časový vývoj hodnot koeficientů Fourierovy řady $a_0$ a $a_2$ během numerické integrace pro funkci $f(t) = \sin^2(\omega t)$ . Graf ilustruje konvergenci koeficientů k jejich analytickým hodnotám $A_0 = 1$ a $A_2 = -0.5$ v čase $t = 2$ s. . . . .	42
4.8	Chování Lorenzova systému v rovině yz. . . . .	46
4.9	Chování Lorenzova systému v časové oblasti. . . . .	46
4.10	Grafy funkce ORD pro různé hodnoty $\rho$ . . . . .	46

# Seznam tabulek

4.1	Společné vstupní parametry a jejich využití v jednotlivých metodách . . . .	26
4.2	Seznam vestavěných adaptivních řešičů obyčejných diferenciálních rovnic, které jsou v práci využity pro porovnání s Taylorovou metodou. . . . .	30
4.3	První sada výsledků ( $TOL = 10^{-3}$ , $MTSM_{TOL} = 10^{-3}$ , $MTSM_h = 0.1$ s, $h = 0.01$ s). . . . .	32
4.4	Druhá sada výsledků ( $TOL = 10^{-7}$ , $MTSM_{TOL} = 10^{-3}$ , $MTSM_h = 0.1$ s, $h = 0.01$ s). . . . .	32
4.5	Třetí sada výsledků ( $TOL = 10^{-7}$ , $MTSM_{TOL} = 10^{-4}$ , $MTSM_h = 10$ s, $h = 0.01$ s). . . . .	32
4.6	První sada výsledků ( $TOL = 10^{-3}$ , $MTSM_{TOL} = 10^{-6}$ , $MTSM_h = 0.1$ s, $h = 0.01$ s). . . . .	36
4.7	Druhá sada výsledků ( $TOL = 10^{-7}$ , $MTSM_{TOL} = 10^{-6}$ , $MTSM_h = 0.1$ s, $h = 0.01$ s). . . . .	37
4.8	Třetí sada výsledků ( $TOL = 10^{-7}$ , $MTSM_{TOL} = 10^{-4}$ , $MTSM_h = 10$ s, $h = 0.01$ s). . . . .	37
4.9	Výsledky numerického řešení systému s $\omega = 100$ rad $\cdot$ s $^{-1}$ , $TOL = 1 \times 10^{-7}$ , $MTSM_{TOL} = 1 \times 10^{-6}$ , $h = 0.01$ s. . . . .	40
4.10	Výsledky numerického výpočtu koeficientů Fourierovy řady pro testovací příklad $f(t) = \sin^2(\omega t)$ pomocí lineárního systému 11 diferenciálních rovnic. . .	43
4.11	Výsledky numerického výpočtu koeficientů Fourierovy řady pomocí nelineárního systému 7 diferenciálních rovnic pro testovací příklad $f(t) = \sin^2(\omega t)$ . . .	44
4.12	Výsledky simulace pro Lorenzův systém, $\rho = 28$ . . . . .	47
4.13	Výsledky simulace pro Lorenzův systém, $\rho = 160$ . . . . .	47
4.14	Výsledky simulace pro Lorenzův systém, $\rho = 23.7$ . . . . .	47
4.15	Výsledky simulace Lorenzova systému pro $\rho = 28$ (neoptimalizovaná verze). . . . .	48
4.16	Výsledky simulace Lorenzova systému pro $\rho = 160$ (neoptimalizovaná verze). . . . .	48
4.17	Výsledky simulace Lorenzova systému pro $\rho = 23.7$ (neoptimalizovaná verze). . . . .	49
4.18	Výsledky simulace Lorenzova systému pro $\rho = 28$ (optimalizovaná verze). . . . .	49
4.19	Výsledky simulace Lorenzova systému pro $\rho = 160$ (optimalizovaná verze). . . . .	49
4.20	Výsledky simulace Lorenzova systému pro $\rho = 23.7$ (optimalizovaná verze). . . . .	50
4.21	Srovnání výpočetních časů Taylorovy metody v různých implementacích pro Lorenzův systém. . . . .	50

# Seznam zkratek

IVP	Počáteční úloha (Initial Value Problem). 8, 13
MTSM	Moderní metoda Taylorovy řady (Modern Taylor Series Method). 3, 4, 16, 32, 33, 36–40, 43, 44, 47–50
ODE	Obyčejná diferenciální rovnice (Ordinary Differential Equation). 3, 6–10, 12, 13, 15–17
ORD	Počet členů Taylorovy řady použitých při výpočtu (Order). 3, 6, 33–36, 38, 40, 45–47
RK4	Runge–Kuttova metoda 4. řádu. 3, 9–11, 14, 16, 29, 31, 32, 36, 37, 39, 40
TOL	Tolerance, maximální přípustná chyba při numerickém výpočtu. 3, 4, 32, 34, 36–40

# Kapitola 1

## Úvod

Numerické řešení obyčejných diferenciálních rovnic (anglicky ordinary differential equations, ODE) je základní metoda v moderním inženýrství, vědeckého výzkumu a průmyslu. Při řešení praktických úloh, pro které neexistuje analytické řešení, nebo je jeho odvození příliš obtížné, se pro získání výsledků používají numerické integrační metody. U numerických metod je důležitá přesnost výpočtu, jeho stabilita a v neposlední řadě i délka výpočtu. Metody vyššího řádu, mezi které patří například Taylorova metoda, umožňují dosáhnout vysoké přesnosti při menším počtu kroků, což velmi snižuje celkové výpočetní náklady.

Cílem této práce je navrhnout, implementovat a porovnat numerické metody pro řešení diferenciálních rovnic s důrazem na optimalizaci Taylorovy metody. V práci jsou porovnány metody, jako je Eulerova metoda, metoda Runge–Kutta a vestavené metody MATLABu s metodou vyššího řádu založenou na Taylorově polynomu s adaptivním nastavením řádu (ORD). Tato optimalizace umožňuje dynamicky upravit počet členů Taylorovy řady v závislosti na požadované přesnosti ( $\varepsilon$ ), čímž se snižuje počet integračních kroků a zrychlují se výpočty. Implementace metody a následné experimenty v jazyce Python byly realizovány nad rámec oficiálního zadání a slouží k rozšíření porovnání o další vývojové prostředí.

Práce je strukturována následovně. V první kapitole je shrnut dosavadní stav problematiky, kde jsou stručně představeny základní typy diferenciálních rovnic, význam počátečních podmínek a přehled numerických metod (včetně Eulerovy a metod Runge–Kutta). Následuje kapitola věnovaná návrhu řešení, ve které je popsán maticový přístup k Taylorově metodě, princip transformace nelineárních problémů na lineární soustavy a možnosti optimalizace výpočtu (např. efektivní výpočet mocnin kroku a optimalizace faktoriálů). V další kapitole jsou podrobně popsány implementační detaily a experimentální ověření metod v prostředí MATLABu, a to i automatizace testování a srovnání přesnosti, stability a výpočetní náročnosti jednotlivých přístupů. Závěrečná kapitola shrnuje dosažené výsledky, vyzdvihuje přínos optimalizované Taylorovy metody a navrhuje možné směry dalšího rozvoje.

# Kapitola 2

## Shrnutí dosavadního stavu

Tato kapitola se zabývá přehledem základních pojmů a metod souvisejících s řešením ODE. Cílem je shrnout klasické i moderní přístupy k numerickému řešení, zejména s ohledem na přesnost, stabilitu a efektivitu metod vyššího řádu. Tento přehled slouží jako teoretický základ pro následný návrh vlastního řešení pomocí Taylorovy metody.

### 2.1 Diferenciální rovnice

Obyčejná diferenciální rovnice je rovnice, která vyjadřuje vztah mezi neznámou funkcí a jejími derivacemi. Nejčastěji se uvažují ODE prvního řádu ve tvaru

$$y'(t) = f(t, y(t)), \quad (2.1)$$

kde  $y(t)$  je hledaná funkce (skalární nebo vektorová) a  $f$  je daná funkce dvou proměnných určující průběh změny  $y$  v čase  $t$ .

Pro vyšší řády lze ODE převést na soustavu prvního řádu zavedením nových proměnných představujících derivace. Například ODE druhého řádu

$$y''(t) = g(t, y, y') \quad (2.2)$$

lze převést substitucí  $v(t) = y'(t)$  na soustavu

$$\begin{aligned} y' &= v, \\ v' &= g(t, y, v). \end{aligned} \quad (2.3)$$

Obecně každou ODE  $n$ -tého řádu lze převést na ekvivalentní soustavu  $n$  rovnic prvního řádu [4]. V dalším textu proto pro jednoduchost vystačíme se soustavami prvního řádu  $y' = f(t, y)$ .

#### 2.1.1 Lineární a nelineární ODE

Důležitým pojmem je rozlišení **lineárních** a **nelineárních** ODE. Lineární ODE lze napsat v tvaru

$$y' = Ay + b, \quad (2.4)$$

případně obecněji v jednorozměrném případě, kde  $n$  je řád derivace,

$$y^{(n)} + a_{n-1}(t)y^{(n-1)} + \dots + a_1(t)y' + a_0(t)y = g(t), \quad (2.5)$$

kde neznámá funkce a její derivace vystupují pouze lineárně (tj. v první mocnině). Koeficienty mohou být funkcí nezávislé proměnné  $t$ , ale nesmí zde být součiny či obecné funkce neznámé  $y$  se sebou samou (ani jejími derivacemi) [15].

Naproti tomu **nelineární** rovnice obsahují členy, kde se neznámá funkce vyskytuje v nelineární formě (např.  $y \cdot y'$  nebo  $\sin(y)$  apod.). Typickým příkladem lineární ODE je homogenní lineární rovnice

$$y' = \lambda y \quad (2.6)$$

s konstantním koeficientem  $\lambda$ , jejímž řešením je exponenciála  $y(t) = Ce^{\lambda t}$  [4]. Příkladem nelineární ODE může být logistická rovnice

$$y' = \alpha y \left(1 - \frac{y}{K}\right) \quad (2.7)$$

modelující růst populace, nebo pohybová rovnice kyvadla

$$\varphi'' + \frac{g}{L} \sin(\varphi) = 0, \quad (2.8)$$

které obě obsahují násobky či obecné funkce neznámé veličiny.

### 2.1.2 Počáteční podmínky

Aby diferenciální rovnice měla jednoznačné řešení, je nutné specifikovat **počáteční podmínku** (resp. podmínky). Řešíme tzv. úlohu počáteční hodnoty (IVP): ke zadané ODE hledáme funkci  $y(t)$  splňující

$$y(t_0) = y_0 \quad (2.9)$$

v daném počátečním čase  $t_0$ . Pro  $n$ -tého řádu je třeba určit  $n$  počátečních podmínek:

$$\begin{aligned} y(t_0) &= y_0 \\ y'(t_0) &= v_0 \\ &\vdots \end{aligned} \quad (2.10)$$

Existence a jednoznačnost řešení za jistých předpokladů (např. spojitost a Lipschitzovská podmínka pravé strany) je zaručena klasickou větou Picardovou–Lindelöfovou [6]. Počáteční podmínka vybírá konkrétní trajektorii řešení ze všech možných. **Význam počátečních podmínek** je zásadní: i nepatrná změna  $y_0$  může vést k výrazně odlišnému průběhu  $y(t)$  [18]. Tato citlivost je známá zejména u chaotických dynamických systémů (např. Lorenzův atraktor [15]), kde malé odchylky v počátku rychle explodují v úplně jiné chování řešení („efekt motýlích křídel“) [14]. Obecně platí, že dobře formulovaná úloha vyžaduje zadání počátečních nebo okrajových podmínek tak, aby řešení bylo jednoznačné a pokud možno stabilní vůči malým změnám těchto podmínek [11].

### 2.1.3 Analytické vs. numerické řešení

Analytické řešení ODE je možné pouze pro některé typy rovnic. Pro lineární rovnice s konstantními koeficienty existují obecné metody (např. charakteristické polynomy, Laplaceova transformace [13]), avšak pro většinu reálných nelineárních problémů nelze nalézt řešení v uzavřeném tvaru nebo je takové řešení výpočetně neefektivní [25].

Z tohoto důvodu se v praxi přistupuje k **numerickému řešení** ODE, při němž získáváme přibližné hodnoty  $y(t)$  pomocí výpočetních metod namísto explicitního vzorce.

Numerické metody integrace ODE jsou základním nástrojem v inženýrských a přírodních vědách, kde se modelují dynamické systémy [17]. Moderní přístupy zahrnují také energeticky konzervativní metody [3] a metody založené na strojovém učení [16].

#### 2.1.4 Explicitní a implicitní metody

Při numerickém řešení obyčejných diferenciálních rovnic rozlišujeme dvě základní třídy metod: **explicitní** a **implicitní**. Tyto metody se liší zejména v tom, jakým způsobem využívají informace z předchozích kroků k výpočtu hodnoty v dalším bodě.

- **Explicitní metody** určují novou hodnotu řešení přímo na základě známých hodnot z předchozího kroku. Tyto algoritmy jsou obvykle výpočetně jednodušší, ale mohou mít omezení v oblasti stability, zejména při řešení tuhých problémů. Při výpočtu nové hodnoty  $y_{n+1}$  využívají pouze předcházející hodnotu a případně starší známé hodnoty.
- **Implicitní metody** naopak určují novou hodnotu řešení nepřímo – rovnice pro výpočet  $y_{n+1}$  obsahuje samotné  $y_{n+1}$ , které ještě neznáme. Pro výpočet je obvykle nutné řešit soustavu (často nelineárních) rovnic, což zvyšuje výpočetní náročnost na jeden krok, ale umožňuje to použít větší integrační krok.

Obecně platí, že explicitní metody jsou jednodušší na implementaci a rychlejší, ale mohou být nestabilní, zejména pro úlohy s vysokou tuhostí. Implicitní metody jsou výpočetně náročnější, ale bývají stabilnější a vhodnější pro tuhé problémy [4].

Tato odlišnost bude dále důležitá při porovnávání výhod a nevýhod různých numerických metod.

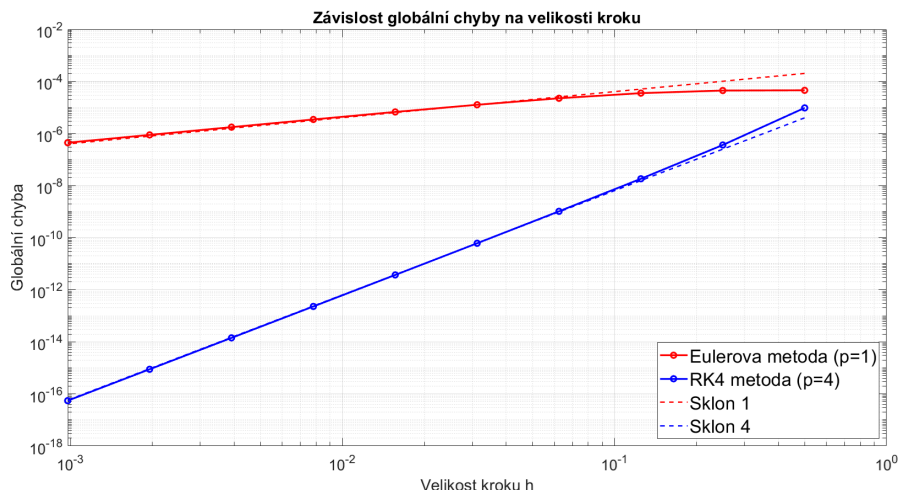
## 2.2 Problematika přesnosti a stability

Numerické řešení ODE je vždy zatíženo jistou chybou oproti skutečnému (analytickému) řešení. Při analýze přesnosti rozlišujeme **lokální chybu** jednoho kroku (chybu, které metoda dosáhne na intervalu  $[t_n, t_{n+1}]$ , pokud na začátku kroku zná přesnou hodnotu) a **globální chybu** řešení v čase  $t_n$  (chybu, která se akumuluje v průběhu všech kroků od počátku). Pro metodu řádu  $p$  platí, že lokální diskretizační chyba je úměrná  $h^{p+1}$ , zatímco globální chyba roste zhruba úměrně  $h^p$  [1, 4]. To znamená, že zmenšení kroku  $h$  například na polovinu zmenší globální chybu přibližně  $2^p$ -krát (pro dostatečně malý  $h$  v asymptotické oblasti konvergence).

Na obrázku 2.1 je ilustrován pokles globální chyby v závislosti na kroku pro Eulerovu metodu (řád  $p = 1$ ) a RK4 (řád  $p = 4$ ). Graf v logaritmickém měřítku jasně ukazuje, že chyba RK4 klesá se čtvrtou mocninou kroku (sklon křivky  $\approx 4$ ) oproti lineárnímu poklesu u Eulerovy metody (sklon  $\approx 1$ ). Modrá a červená čárkovaná přímkou slouží jako orientační reference pro sklon 4, resp. 1. Z grafu je zřejmé, že metoda vyššího řádu dosahuje výrazně menší chyby při stejném kroku  $h$ , což umožňuje použít i větší kroky k dosažení dané přesnosti.

Z hlediska řádu přesnosti lze základní metody charakterizovat následovně:

- Explicitní Eulerova metoda má 1. řád přesnosti
- Metody typu Runge-Kutta 2. řádu (RK2) mají 2. řád přesnosti



Obrázek 2.1: Závislost globální chyby na velikosti kroku pro Eulerovu metodu a RK4 při řešení ODE  $y' = -2y, y(0) = 1$  do  $t = 5$ .

- Metoda Runge-Kutta 4. řádu (RK4) dosahuje 4. řádu přesnosti
- Taylorova metoda  $n$ -tého řádu má  $n$ -tý řád přesnosti

Volba velikosti kroku  $h$  zásadně ovlivňuje přesnost i stabilitu metody. Pokud zvolíme krok příliš velký, nemusí polynomiální aproximace mezi body  $t_n$  a  $t_{n+1}$  dostatečně zachytit změnu řešení – výsledkem je velká lokální chyba, která se propaguje dále. Naopak velmi malý krok sice sníží diskretizační chybu, ale znamená více kroků a tím více příležitostí pro akumulaci numerických nepřesností (včetně zaokrouhlovacích chyb při aritmetice s plovoucí řádovou čárkou). V extrémním případě může při příliš malém  $h$  začít dominovat zaokrouhlovací chyba a globální chyba se přestane zlepšovat. To znamená, že existuje optimální velikost kroku, za níž už celková chyba neklesá [25]. Z hlediska efektivity je tedy žádoucí volit krok co největší, ale ještě takový, aby splňoval požadavky na přesnost.

### 2.2.1 Stabilita numerických metod

Důležitou vlastností numerických metod je jejich **stabilita**. Ta určuje, zda se chyby v průběhu numerického výpočtu (zaokrouhlovací, diskretizační) tlumí nebo naopak zesilují při postupném provádění iterací. Numerická metoda je považována za stabilní, pokud hodnota řešení v následujícím integračním kroku nepřekročí (v absolutní hodnotě) hodnotu v aktuálním kroku [26]:

$$|y_{i+1}| \leq |y_i|. \quad (2.11)$$

Pro analýzu stability se zavádí funkce stability  $R(z)$ , definovaná jako:

$$R(z) = \frac{y_{i+1}}{y_i} \quad (2.12)$$

kde  $z = h\lambda$ , přičemž  $h$  je velikost integračního kroku a  $\lambda$  je parametr v testovací rovnici. Metoda je stabilní v oblasti absolutní stability  $\mathcal{D}$ :

$$\mathcal{D} = \{z \in \mathbb{C} \mid |R(z)| \leq 1\}. \quad (2.13)$$

Pokud oblast stability  $\mathcal{D}$  zahrnuje celou levou komplexní polorovinu:

$$\mathcal{D}(z) = \{z \in \mathbb{C}, \operatorname{Re}(z) \leq 0\} \quad (2.14)$$

pak mluvíme o **absolutní stabilitě** (A-stabilitě) dané metody.

Pro zkoumání stability metod se často používá Dahlquistův testovací problém:

$$y' = \lambda y, \quad y(0) = y_0, \quad \operatorname{Re}(\lambda) < 0. \quad (2.15)$$

Různé numerické metody mají odlišné oblasti stability, což je důležité zejména při řešení tuhých problémů, kde je volba vhodné metody z hlediska stability zásadní.

Například u **tuhých systémů**, které mají některé vlastní čísla s velmi zápornou reálnou částí, vyžadují explicitní metody extrémně malý krok k udržení stability (viz podmínka  $|1 + \lambda h| < 1$  u Eulerovy metody) [11]. V takových případech se používají speciální A-stabilní metody (typicky implicitní, např. rodina implicitních Runge–Kutta metod nebo vícekrokové BDF metody) [20], které umožňují bezpečně integrovat i tuhé rovnice s velkým krokem.

**Tuhost** (angl. *stiffness*) je vlastnost diferenciálních rovnic, kdy řešení obsahuje složky, které se mění velmi rychle (např. exponenciálně klesají) vedle složek, které se mění pomalu. Tuhé systémy (stiff systems) jsou charakteristické tím, že explicitní metody pro ně vyžadují extrémně malý integrační krok z důvodu stability, přestože přesnost by byla dostatečná i s mnohem větším krokem. Pro efektivní řešení tuhých soustav jsou proto vhodnější implicitní metody, které mohou pracovat s větším integračním krokem [4].

Pro netuhé úlohy, na něž cílí tato práce, není stabilita většinou limitujícím faktorem – pokud volíme krok rozumně vzhledem k charakteristické časové škále problému, moderní metody (RK s adaptací, Taylorova metoda) si stabilitu udrží. Například metoda RK4 má poměrně širokou oblast absolutní stability (pro testovací rovnici  $y' = \lambda y$  zůstává stabilní pro  $\lambda h$  zhruba v oblasti  $\operatorname{Re}(\lambda h) < 0$  o šířce přibližně 2.8 na reálné ose), což je výrazně lepší než u Eulerovy metody [4]. Metody vyšších řádů mívají komplexnější stabilitní charakteristiky; obecně platí, že žádná explicitní metoda s  $p > 1$  není A-stabilní [13], avšak prakticky to u hladkých netuhých úloh nevádí.

### 2.2.2 Akumulace chyby

**Akumulace chyby** je dalším důležitým aspektem při numerickém řešení diferenciálních rovnic. Globální chyba  $e_n = y(t_n) - y_n$  vzniká z chyb v každém kroku (lokálních chyb) a také z propagace předchozí chyby numerickým procesem. Lze ukázat, že pokud je metoda stabilní ve smyslu nulových perturbací (tzv. 0-stabilita), pak globální chyba roste lineárně s počtem kroků a je řádu  $O(h^p)$  [4, 1].

Při praktickém použití numerických metod je třeba věnovat pozornost několika faktorům:

- **Zaokrouhlovací chyby** mohou významně ovlivnit výsledek, zvláště při použití velmi malých kroků
- **Tuhé problémy** mohou vyžadovat speciální metody bez ohledu na teoretický řád přesnosti
- **Nespojitosti** v řešené funkci mohou způsobit degradaci řádu metody
- **Nestabilita** může způsobit, že teoreticky přesnější metoda dává v praxi horší výsledky

Nicméně pokud by metoda nebyla dostatečně stabilní, mohou se chyby z předchozích kroků zesilovat. To byl případ explicitních Adamsových metod vyšších řádů zmíněný výše – jejich 0-stabilita selhává od 5. řádu, takže libovolně malé  $h$  negarantuje konvergenci. V praxi běžně používané metody (RK, Adams do 4. řádu, Taylorova metoda atd.) jsou navrženy tak, aby byly minimálně 0-stabilní a většinou i A-stabilní (pokud jsou implicitní) nebo dostatečně stabilní pro netuhé systémy (pokud jsou explicitní). Akumulace chyby se tedy řídí především velikostí lokální chyby a počtem kroků.

### 2.2.3 Přesnost versus výpočetní náročnost

Z hlediska **přesnosti versus výpočetní náročnosti** existuje kompromis při volbě řádu metody. Metody **vyšších řádů** (např. RK8, Taylorova metoda s  $m$  kolem 10 a více) dokážou při hladkém průběhu řešení dosáhnout mimořádně malých chyb i pro relativně velké kroky  $h$  [25]. To je výhodné například u:

- dlouhodobých simulací, kde by akumulace malé chyby na každém kroku jinak přerostla v nepoužitelný výsledek – vysoký řád zajistí, že chyba zůstane v požadovaných mezích i po mnoha krocích (příkladem mohou být astronomické výpočty drah planet na stovky let, kde se používají metody řádů 10–12),
- citlivých (chaotických) systémů, kde je často nutné použít velmi malý krok pro stabilitu a přesnost; metody vyšších řádů v kombinaci s adaptivním řízením kroku tam mohou být efektivnější než opakované zjemňování kroku u nižších řádů [15].

Studie rovněž ukazují, že pro požadovanou vysokou přesnost mohou moderní Taylorovy metody překonat klasické numerické metody – ušetří mnoho kroků a tím i výpočetního času [15, 1]. Na druhou stranu, metody vysokých řádů vyžadují více výpočtů v každém kroku (např. RK8 provádí 13 vyhodnocení  $f$  na krok, Taylorova metoda musí spočítat několik řadu derivací). Pro méně náročné požadavky na přesnost tak může jednodušší metoda nižšího řádu dosáhnout výsledku rychleji.

Existuje také praktický limit daný numerickou aritmetikou – při extrémně vysokém řádu může docházet k větším zaokrouhlovacím chybám a ztrátě signifikantních cifer, takže přínos vyššího řádu se nakonec vytrácí [9]. V literatuře se uvádí, že optimální řád explicitní metody v dvojité přesnosti bývá okolo 8–12; nad tuto hranici už dodatečný zisk přesnosti většinou nevyvažuje nárůst náročnosti a rizika numerických chyb.

### 2.2.4 Shrnutí

Závěrem lze říci, že volba metody a velikosti integračního kroku závisí na charakteru úlohy. Pro běžné hladké netuhé problémy dostačují klasické metody (např. RK 4. řádu, popř. adaptivní RK) a střední kroky. Pokud je ale požadována vysoká přesnost, dlouhý integrační interval nebo řešení velmi citlivého systému, metody vyšších řádů (Taylorova metoda, více-krokové vysokého řádu či extrapolační metody) mohou nabídnout výrazně lepší výsledky.

Důležité je přitom vždy sledovat jak **přesnost**, tak **stabilitu** – spolehlivá metoda by měla automaticky řídit krok tak, aby udržela oboje v přijatelných mezích [11]. Problémy tuhosti řeší implicitní A-stabilní metody, které však nejsou předmětem této práce. Celkově jsou metody pro numerické řešení ODE dobře prozkoumané a výběr vhodné metody (včetně jejího řádu) a kroku  $h$  je klíčem k úspěšnému a efektivnímu řešení konkrétní úlohy [4, 13].

## 2.3 Přehled numerických metod

Numerické metody pro řešení ODE aproximují skutečné řešení diskrétní posloupností hodnot  $y_n \approx y(t_n)$ , kde  $t_n$  jsou uzlové body (typicky  $t_n = t_0 + nh$  pro zvolený integrační krok  $h$ ). Z této aproximace lze získat spojitou interpolační funkci, primární je však právě výpočet diskrétní trajektorie  $\{y_n\}$ .

Metody lze rozdělit na **jednokrokové** a **víceprokové** [4, 11]. Jednokrokové metody počítají nový bod  $y_{n+1}$  pouze z bodu předchozího  $y_n$  (případně s pomocnými mezivýpočty uvnitř intervalu  $[t_n, t_{n+1}]$ ). Naproti tomu víceprokové metody využívají několik minulých bodů  $y_n, y_{n-1}, \dots, y_{n-k}$ ; typickým příkladem je Adams-Bashforthova explicitní metoda, která kombinuje hodnoty derivace v několika předchozích krocích [25].

V této práci se zaměříme především na jednokrokové metody, konkrétně na metodu založenou na **Taylorově rozvoji** [15], kterou budeme srovnávat s dalšími běžnými metodami: **Eulerovou metodou**, která představuje nejjednodušší přístup, a metodami z rodiny **Rungeho–Kuttových metod**, jež jsou v současnosti široce používané pro svou robustnost a efektivitu [22, 25].

### 2.3.1 Eulerova metoda

Jedním z nejjednodušších přístupů k numerické integraci ODE je **explicitní Eulerova metoda**, pojmenovaná po Leonhardu Eulerovi. Vychází z definice derivace: na malém časovém úseku  $h$  se řešení posune přibližně o  $y'(t) \cdot h$ . Eulerův krok tedy používá směr (sklon) řešení v počátku intervalu a lineárně extrapoluje hodnotu. Diskrétní iterační vzorec pro IVP  $y' = f(t, y)$  je:

$$y_{n+1} = y_n + h f(t_n, y_n) \quad (2.16)$$

Ze známého  $y_n$  v čase  $t_n$  tak získáme  $y_{n+1}$  v čase

$$t_{n+1} = t_n + h \quad (2.17)$$

přičtením přírůstku určeného směrem  $f(t_n, y_n)$ . Eulerova metoda je velmi jednoduchá na implementaci a výpočetně nenáročná (vyžaduje jen jedno vyhodnocení funkce  $f$  na krok). Jedná se však o metodu **1. řádu přesnosti**, neboť lokální diskretizační chyba je řádu  $O(h^2)$  a globální chyba narůstá lineárně  $O(h)$  [4, 25]. To znamená, že pro dosažení vysoké přesnosti je třeba volit velmi malý krok  $h$ .

Metoda je navíc pouze **podmíněně stabilní**: aplikujeme-li ji na Dahlquistův testovací problém

$$y' = \lambda y \quad (2.18)$$

dostaneme iterační vztah

$$y_{n+1} = (1 + \lambda h) \cdot y_n. \quad (2.19)$$

Numerické řešení tak bude stabilní jen pokud platí

$$|1 + \lambda h| < 1 \quad (2.20)$$

[11]. Pro záporné reálné  $\lambda$  (stabilní spojité systémy) to představuje omezení

$$h < \frac{2}{|\lambda|}. \quad (2.21)$$

Toto omezení definuje oblast stability Eulerovy metody v komplexní rovině kolem bodu  $(-1, 0)$  s poloměrem 1. Pokud je krok větší než tato mez, metoda může numerické řešení

zcela znehodnotit (oscilace nebo divergence místo správného poklesu). Eulerova metoda tedy vyžaduje dostatečně malý krok nejen kvůli přesnosti, ale i kvůli numerické stabilitě.

Existuje také **implicitní Eulerova metoda** s iteračním vzorcem

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1}) \quad (2.22)$$

kteřá používá sklon v koncovém bodě kroku. Ta je A-stabilní (nepodmíněně absolutně stabilní na testovací rovnici pro libovolně velké  $h$ ) [13], avšak vyžaduje řešit v každém kroku obecně nelineární rovnici (např. Newtonovou iterací), což ji činí výpočetně náročnější.

### 2.3.2 Metody Runge–Kutta

Eulerova metoda využívá pouze informaci o derivaci na začátku intervalu. Přesnost lze významně zvýšit, pokud dokážeme zohlednit průběh funkce  $f(t, y)$  i uvnitř intervalu  $[t_n, t_{n+1}]$ . Tímto principem se řídí **Rungeho–Kuttovy metody** (RK metody), pojmenované po Carlu Rungeovi a Wilhelmu Kuttovi, kteří je rozvinuli na počátku 20. století. RK metody vyšších řádů vzorkují směr  $y'$  (pravou stranu) ve více bodech a vhodnou lineární kombinací těchto „mezisklonů“ dosahují vyšší přesnosti integrace [4].

Nejznámější je **klasická RK4**, tj. explicitní Runge–Kuttova metoda 4. řádu. Její algoritmus na každém kroku vypočítá čtyři pomocné hodnoty (odhady směrů), označované tradičně  $k_1, k_2, k_3, k_4$ :

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \\ k_4 &= f(t_n + h, y_n + hk_3). \end{aligned} \quad (2.23)$$

Tyto hodnoty odpovídají odhadům derivace  $y'$  v několika bodech kroku (na začátku, dvakrát uprostřed a na konci intervalu). Nový bod se pak získá sloučením těchto informací:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (2.24)$$

Tato vážená suma (odpovídající Simpsonovu integračnímu pravidlu) zajistí korekci celkového směru – hrubě řečeno  $k_1$  a  $k_4$  (počáteční a koncový sklon) dostanou menší váhu, zatímco  $k_2$  a  $k_3$  (sklon v polovině intervalu) dvojnásobnou [11]. RK4 metoda dosahuje 4. řádu globální přesnosti, což je výrazné zlepšení oproti Eulerovi při stále poměrně nízké výpočetní náročnosti. Cena za vyšší přesnost je čtyřnásobné volání funkce  $f$  v každém kroku namísto jednoho. Pro mnoho praktických úloh je však RK4 velmi efektivním kompromisem mezi přesností a rychlostí a patří k nejpoužívanějším pevnokrokovým metodám.

Moderní softwarové implementace často využívají adaptivní  **vkládané Runge–Kuttovy metody**, které odhadují lokální chybu a podle ní upravují krok  $h$  [25]. Například populární řešič `ode45` v MATLABu využívá dvojici formulek 4. a 5. řádu (metoda Dormand–Prince) [22]. V každém kroku se spočítá nejen výsledné  $y_{n+1}$  pátého řádu, ale i doplňkové řešení čtvrtého řádu; jejich rozdíl aproximuje lokální chybu. Pokud je chyba větší než zadaná tolerance, krok se zmenší; naopak při velmi malém odhadu chyby se krok zvětší. Tímto způsobem adaptivní RK řešiče (jako `ode45`) dosahují požadované přesnosti s optimálně velkým krokem v každém úseku řešení a jsou proto velmi efektivní pro hladká netuhá řešení.

Pro extrémně přesné požadavky existují i explicitní RK metody vyšších řádů (v MATLABu např. `ode78` a `ode89`, kde první číslo označuje řád metody a druhé číslo řád metody použité pro kontrolu chyby) [23, 24], avšak jejich praktické využití je omezené na speciální úlohy s velmi hladkým průběhem na dlouhém intervalu [9]. RK metody obecně představují univerzální a dobře prozkoumaný nástroj; detailní teoretický rozbor těchto metod (řád, stabilita, Butcherovy tabulky apod.) lze nalézt například v [4].

Kromě jednokrokových metod Runge-Kutta existují také **více krokové metody**, například Adams–Bashforthovy a Adams–Moultonovy metody, nebo metody založené na schématu **prediktor-korektor** kombinujícím explicitní a implicitní kroky [17]. Tyto metody využívají interpolaci či extrapolaci derivace z několika předchozích bodů a mohou dosáhnout vysokého řádu. Například MATLAB řešič `ode113` (označení značí adaptivně volený řád až 13) kombinuje prediktor Adams–Bashforth a korektor Adams–Moulton a dynamicky volí řád 1–13 podle potřeby [19].

Jedná se o PECE algoritmus (Predict–Evaluate–Correct–Evaluate), kde explicitní prediktor odhaduje nový bod a poté implicitní korektor zpřesní řešení s využitím predikce, čímž se eliminuje nutnost iterativního řešení nelineární rovnice od nuly. Více krokové metody ovšem trpí některými omezeními stability. V praxi se vyšších řádů dosahuje spíše implicitními více krokovými metodami nebo jinými technikami (extrapolace, kombinace s RK apod.).

### 2.3.3 Metoda Taylorova rozvoje

Všechny doposud zmíněné metody (Euler, Runge–Kutta, Adams) ve své podstatě aproximují řešení polynomiálně pomocí omezeného počtu členů jeho lokálního Taylorova rozvoje, aniž by však tyto derivace počítaly přímo. Například Eulerův krok lze interpretovat jako uříznutí Taylorovy řady po lineárním členu:

$$y(t_n + h) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(\xi), \quad (2.25)$$

kde  $y''(\xi)$  je zbytek (druhá derivace na nějakém  $\xi \in (t_n, t_n + h)$ ) – Eulerova metoda tak ignoruje všechny vyšší derivace a nese chybu  $O(h^2)$ . Runge–Kutta metody se dají považovat za promyšlený způsob, jak získat informaci ekvivalentní vyšším derivacím z opakovaného vyhodnocení původní funkce  $f$  [4, 11].

Naproti tomu **Taylorova metoda** přistupuje přímo k použití rozvoje řešení do Taylorovy řady. Pokud má funkce na intervalu vyšší derivaci, můžeme použít Taylorovou řadu  $m$ -tého řádu:

$$y(t_n + h) \approx y(t_n) + hy'(t_n) + \frac{h^2}{2!}y''(t_n) + \dots + \frac{h^m}{m!}y^{(m)}(t_n). \quad (2.26)$$

Tuto ideu lze přímo převést na vzorec pro výpočet následující hodnoty numerické metody:

$$y_{n+1} = y_n + hy'(t_n) + \frac{h^2}{2!}y''(t_n) + \dots + \frac{h^m}{m!}y^{(m)}(t_n). \quad (2.27)$$

Takový algoritmus by byl velmi účinný – jedním krokem by spočítal  $y_{n+1}$  s chybou řádu  $O(h^{m+1})$ , tedy lze dosáhnout libovolně vysoké přesnosti volbou dostatečného  $m$ . Problém ovšem je, že obecně neznáme hodnoty vyšších derivací  $y', y'', \dots, y^{(m)}$  v čase  $t_n$ . Musíme je získat z původní ODE. Pro jednoduchost uvažujme ODE ve tvaru  $y' = f(t, y)$ . Potom

$$y''(t) = \frac{d}{dt}y'(t) = \frac{d}{dt}f(t, y(t)) = \frac{\partial f}{\partial t}(t, y) + \frac{\partial f}{\partial y}(t, y)y'(t), \quad (2.28)$$

kde  $\frac{\partial f}{\partial y}$  značí derivaci  $f$  podle  $y$  a člen  $y'$  lze opět nahradit  $f(t, y)$ . Tím dostáváme  $y''$  vyjádřeno pomocí  $f$  a jeho parciálních derivací. Podobně můžeme pokračovat k výrazu pro  $y'''$  (zderivovat  $y''$ ) atd. Obecně dostaneme pro  $k$ -tou derivaci vztah  $y^{(k)}(t) = f_k(t, y(t))$ , kde  $f_k$  je  $k$ -tá funkce vzniklá opakovanou diferenciací původního výrazu  $f$  [4].

Aplikace Taylorova polynomu  $m$ -tého řádu tak vyžaduje mít k dispozici postupně  $f, f_2, f_3, \dots, f_m$ , přičemž  $f_1 \equiv f$ . Metoda založená na takovém vzorci se nazývá **Taylorova (složená) metoda řádu  $m$** . Jde vlastně o obecnou formu explicitní jednokrokové metody, do níž ostatní výše zmíněné metody spadají jako speciální případy (pro konkrétní  $m$  a způsob, jakým získávají informaci o vyšších derivacích – např. RK4 odpovídá použití efektivních 4 derivací pomocí vícenásobného volání  $f$  namísto explicitního výpočtu  $f_2, f_3, f_4$ ).

Historicky byla Taylorova metoda jedním z prvních uvažovaných přístupů k numerickému řešení ODE, ale dříve nebyla příliš praktická. Hlavní překážkou byla složitost odvození výrazů pro vysoké derivace pro každý konkrétní problém. Pro každou novou ODE bylo nutné ručně (nebo pomocí symbolického algebraického systému) vypočítat a implementovat výrazy pro  $f_2, f_3, \dots$ , což je pracné a náchylné k chybám [18].

Již v 60. letech však vznikly první programy, které se pokusily generovat potřebné derivace ze zadané funkce  $f$  automaticky – rané práce tehdy využívaly rekurzivní techniky a intervalovou aritmetiku k sestavení Taylorova polynomu se zárukou přesnosti [6]. Později byly vyvinuty sofistikovanější algoritmy, které dokáží generovat Taylorovy koeficienty rekurentně přímo během výpočtu, čímž odpadá ruční derivování [25]. Tento přístup dnes spadá do oblasti **automatické diferenciaci** (Automatic Differentiation, AD) a **symbolického výpočtu**, které umožňují získat libovolný počet derivací programově.

V posledních dvou dekadách zažívá Taylorova metoda obrození díky konceptu tzv. **moderní Taylorovy metody** (Modern Taylor Series Method, MTSM) [15]. Jedná se o plně automatizovaný postup, který řeší ODE s proměnným krokem i řádem: na začátku každého kroku rekurentně vygeneruje tolik členů Taylorova rozvoje, kolik je potřeba k dosažení požadované tolerance chyby, a podle toho zvolí i délku kroku [5]. Metoda tak dynamicky zvyšuje řád v úsecích, kde je řešení hladké, a naopak zmenšuje krok tam, kde hladké není.

Výzkum na VUT v Brně navázal na tento směr a vyvinul efektivní vektorový a maticový přístup k implementaci Taylorovy metody [15]. Podrobnosti o tomto maticovém přístupu budou rozvedeny v následující kapitole.

# Kapitola 3

## Návrh řešení

Tato kapitola se zabývá návrhem efektivního algoritmu pro numerické řešení ODE pomocí Taylorovy metody vyššího řádu. Na základě poznatků z předchozí kapitoly práce se zaměřím na konkrétní implementační kroky vedoucí k optimalizaci výpočtu. Nejdříve se zaměříme na maticový přístup k výpočtu derivací, který umožňuje systematickou a efektivní konstrukci členů Taylorovy řady. Poté se bude kapitola věnovat transformaci problému na autonomní tvar a následně optimalizacím, které snižují výpočetní náročnost a umožňují rychlejší výpočty při zachování stejné přesnosti. V následujícím textu budou vektory značeny tučným písmem, zatímco skaláry a matice normálním písmem.

### 3.1 Maticový přístup k Taylorově metodě

Numerická Taylorova metoda využívá rozvoj řešení do Taylorovy řady a umožňuje tak dosáhnout libovolného řádu přesnosti jedním krokem (viz rovnice 2.26). Klíčovým problémem je však získání hodnot vyšších derivací  $y'(t)$ ,  $y''(t)$ ,  $y^{(3)}(t)$ , ... v čase  $t_n$  pro každou iteraci. Maticový přístup tento problém řeší systematicky pomocí lineární algebry – přepisuje diferenciální úlohu do maticové podoby, v níž lze vyšší derivace spočítat opakovaným použitím maticových operací místo ručního odvozování výrazů [15, 1].

Tento přístup vychází z myšlenky, že pro lineární soustavy obyčejných diferenciálních rovnic je určování derivací triviální a lze je zapsat pomocí mocnin matice popisující systém [11].

#### 3.1.1 Rozklad diferenciálních rovnic na soustavu diferenciálních rovnic prvního řádu

Nejprve je třeba převést danou úlohu na soustavu prvního řádu. Vyšší řád ODE lze standardně rozložit zavedením nových proměnných představujících derivace nižšího řádu [4]. Například druhého řádu ODE  $y'' + a_1y' + a_0y = g(t)$  lze převést na ekvivalentní soustavu dvou prvních derivací zavedením substituce:

$$\begin{aligned}y_1 &= y, \\ y_2 &= y',\end{aligned}\tag{3.1}$$

takže

$$\begin{aligned}y_1' &= y_2, \\ y_2' &= -a_1y_2 - a_0y_1 + g(t).\end{aligned}\tag{3.2}$$

Obecně lze každou diferenciální rovnici  $n$ -tého řádu převést na soustavu  $n$  diferenciálních rovnic prvního řádu. Je však třeba poznamenat, že substituce, jak je výše uvedena, je jen jedním ze způsobů řešení tohoto problému. Další přístupy zahrnují metody snižování řádu derivate nebo metody postupné integrace, které mohou být v některých případech výhodnější.

Tato soustava může být zapsána ve vektorové formě

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) \quad (3.3)$$

kde  $\mathbf{y}$  je stavový vektor proměnných. Pro diferenciální rovnici (soustavu) navíc platí, že funkce  $\mathbf{f}$  má tvar lineární kombinace stavových proměnných plus případný nehomogenní člen nezávislý na  $\mathbf{y}$ . Typicky uvažujeme lineární soustavu s konstantními koeficienty ve tvaru

$$\mathbf{y}'(t) = A\mathbf{y}(t) + \mathbf{b} \quad (3.4)$$

kde  $A$  je matice o velikosti  $n \times n$  a  $\mathbf{b}$  je  $n$ -rozměrný sloupcový vektor.

Pro lineární soustavu (3.4) je určování vyšších derivací přímočaré. Pokud jsou  $A$  a  $\mathbf{b}$  časově nezávislé (konstantní), pak derivací obou stran rovnice (3.4) podle  $t$  získáme druhou derivaci:

$$\mathbf{y}''(t) = A\mathbf{y}'(t) + \mathbf{b}'(t). \quad (3.5)$$

Protože  $\mathbf{b}$  je konstantní vektor (tj.  $\mathbf{b}' = \mathbf{0}$ ), platí

$$\mathbf{y}''(t) = A\mathbf{y}'(t) = A(A\mathbf{y} + \mathbf{b}) = A^2\mathbf{y}(t) + A\mathbf{b}. \quad (3.6)$$

Analogicky třetí derivace bude

$$\mathbf{y}'''(t) = A\mathbf{y}''(t) = A^3\mathbf{y}(t) + A^2\mathbf{b} \quad (3.7)$$

a obecně lze indukci odvodit vztah [1]:

$$\mathbf{y}^{(k)}(t) = A^k\mathbf{y}(t) + A^{k-1}\mathbf{b} \quad \text{pro každé } k \geq 1. \quad (3.8)$$

Tato formule jednoznačně vyjadřuje  $k$ -tou derivaci řešení lineární soustavy pomocí mocniny matice  $A$ . Je-li navíc  $\mathbf{b} = \mathbf{0}$  (homogenní soustava), zjednoduší se na

$$\mathbf{y}^{(k)}(t) = A^k\mathbf{y}(t). \quad (3.9)$$

V praxi není nutné skutečně počítat  $A^k$  pro velká  $k$  pomocí náročných maticových operací, protože výpočet derivací lze zřetězit rekurentně [5]. Z rovnice (3.5) vidíme, že známe-li hodnotu první derivace  $\mathbf{y}'(t_n)$ , pak  $\mathbf{y}''(t_n)$  získáme vynásobením  $\mathbf{y}'(t_n)$  maticí  $A$ . Obecně ze vztahu (3.8) plyne rekurence:

$$\mathbf{y}^{(k+1)}(t_n) = A\mathbf{y}^{(k)}(t_n) \quad \text{pro } k \geq 1 \quad (3.10)$$

protože v  $k$ -té derivaci (3.8) lze  $A^{k-1}\mathbf{b}$  považovat za konstantu (je to konkrétní vektor při pevně daném  $t_n$ ) a jeho derivace je nulová. První derivace

$$\mathbf{y}'(t_n) = A\mathbf{y}(t_n) + \mathbf{b} \quad (3.11)$$

se vypočítá přímo dosazením do (3.4). Vyšší derivace se vypočítají aplikováním matice  $A$  na předchozí derivaci (tj.  $\mathbf{y}^{(2)}(t_n) = A\mathbf{y}'(t_n)$ ,  $\mathbf{y}^{(3)}(t_n) = A\mathbf{y}''(t_n)$  atd.). Tímto postupem získáme systematicky  $\mathbf{y}^{(k)}(t_n)$  pro libovolné  $k$  bez potřeby odvozovat individuálně výraz pro  $k$ -tou derivaci [1].

### 3.1.2 Využití maticové reprezentace pro výpočet vyšších derivací

Jakmile máme mechanismus pro získání derivací, můžeme sestavit Taylorův polynom řešení. Podle vzorce (3.8) platí

$$\mathbf{y}(t_n + h) = \mathbf{y}(t_n) + h \mathbf{y}'(t_n) + \frac{h^2}{2!} \mathbf{y}''(t_n) + \dots \quad (3.12)$$

V konečné implementaci nelze sčítat nekonečně mnoho členů, takže výpočet dalších členů řady ve chvíli, kdy jsou další členy zanedbatelně malé vůči definované přesnosti [1]. Pro lineární soustavy lze teoreticky určit poloměr konvergence Taylorovy řady (například pro homogenní systém  $y' = \lambda y$  je poloměr konvergence nekonečný, protože jde o expanzi exponenciály). V numerickém výpočtu se ovšem přistupuje k adaptivní volbě počtu členů: v každém kroku generujeme členy tak dlouho, dokud poslední člen nepřekročí zadanou toleranci  $\varepsilon$  [5].

Algoritmus pro jeden integrační krok  $t_n \rightarrow t_{n+1} = t_n + h$  tedy vypadá následovně:

#### Výpočet prvního členu

Z aktuálního stavu  $\mathbf{y}(t_n)$  vypočítáme první derivaci  $\mathbf{y}'(t_n) = A\mathbf{y}(t_n) + \mathbf{b}$  (podle (3.4)). Následně sestavíme první příspěvek Taylorova rozvoje:

$$T_1 = h\mathbf{y}'(t_n), \quad (3.13)$$

což odpovídá členu  $\frac{h^1}{1!}\mathbf{y}'(t_n)$ . Tento člen přičteme k průběžné sumě:  $\mathbf{Y} = \mathbf{y}(t_n) + T_1$  (na začátku je  $\mathbf{Y}$  inicializováno počáteční podmínkou  $\mathbf{y}(t_n)$ , což odpovídá nultému členu).

#### Iterativní generování dalších členů

Pro  $k = 1, 2, 3, \dots$  opakujeme:

1. Z předchozího  $k$ -tého členu  $T_k = \frac{h^k}{k!}\mathbf{y}^{(k)}(t_n)$  vygenerujeme  $(k+1)$ -ní člen. Díky rekurenci (3.10) platí  $\mathbf{y}^{(k+1)}(t_n) = A\mathbf{y}^{(k)}(t_n)$ . Vzhledem k tomu, že  $\mathbf{y}^{(k)}(t_n)$  je již obsažena v  $T_k$  (jako  $T_k \cdot \frac{k!}{h^k}$ ), lze nový člen vypočítat jako

$$T_{k+1} = \frac{h}{k+1}AT_k. \quad (3.14)$$

Tento vztah plyne z:

$$T_k = \frac{h^k}{k!}\mathbf{y}^{(k)}(t_n) \implies \mathbf{y}^{(k)}(t_n) = \frac{k!}{h^k}T_k, \quad (3.15)$$

a dosazením do definice  $T_{k+1} = \frac{h^{k+1}}{(k+1)!}\mathbf{y}^{(k+1)}(t_n)$  a využitím  $\mathbf{y}^{(k+1)}(t_n) = A\mathbf{y}^{(k)}(t_n)$  dostaneme

$$T_{k+1} = \frac{h^{k+1}}{(k+1)!}A\mathbf{y}^{(k)}(t_n) = \frac{h^{k+1}}{(k+1)!}A\frac{k!}{h^k}T_k = \frac{h}{k+1}AT_k, \quad (3.16)$$

což je právě vzorec (3.14).

2. Přičteme nový člen k sumě:  $\mathbf{Y} := \mathbf{Y} + T_{k+1}$ .

3. Ověříme podmínku ukončení – pokud  $\|T_{k+1}\|_\infty < \varepsilon$  (největší složka vektoru  $T_{k+1}$  je menší než stanovená tolerance), máme dostatečný počet členů a generování se ukončí [15]. V opačném případě pokračujeme dalším krokem.
4. Zabezpečíme, aby se algoritmus nezacyklil, i když řada konverguje příliš pomalu: máme-li apriori odhad maximálního počtu členů (např.  $m_{\max} = 60$ ), ukončíme cyklus i při jeho dosažení a ohlásíme varování, že daný krok vyžaduje příliš mnoho členů.

### Aktualizace stavu

Po ukončení sčítání nastavíme  $\mathbf{y}(t_{n+1}) = \mathbf{Y}$  jako výsledek pro další krok. Dále je vhodné uložit si počet použitých členů  $m$  v tomto kroku (tzv. aktuální řád metody), abychom měli přehled o konvergenci řady a tuto informaci mohli využít pro adaptivní volbu kroku (viz dále).

Tento algoritmus implementuje Taylorovu metodu  $m$ -tého řádu s proměnným  $m$  – v každém kroku automaticky generuje tolik členů, kolik je potřeba k dosažení požadované přesnosti  $\varepsilon$  [5, 25].

### Efektivita maticového přístupu

Výhodou maticového přístupu je, že výpočet  $AT_k$  lze provést efektivně využitím vektorových a maticových operací (např. v prostředí MATLAB či v jazyce Python s knihovnou NumPy jsou tyto operace optimalizovány a využívají vektorové instrukce CPU) [22]. Všechny složky vektoru  $\mathbf{y}$  a jejich derivace se tak počítají najednou. Například pro jednoduchou lineární soustavu

$$\begin{aligned} y_1' &= \alpha y_1 + \beta y_2 \\ y_2' &= \gamma y_1 + \delta y_2 \end{aligned} \tag{3.17}$$

s maticí  $A = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}$  se následující člen Taylorovy řady  $T_{k+1}$ , podle vztahu (3.14), vypočítá jako

$$T_{k+1} = \frac{h}{k+1} \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \begin{pmatrix} T_{k,1} \\ T_{k,2} \end{pmatrix} = \frac{h}{k+1} \begin{pmatrix} \alpha T_{k,1} + \beta T_{k,2} \\ \gamma T_{k,1} + \delta T_{k,2} \end{pmatrix}, \tag{3.18}$$

tedy obě složky nového členu se získají jedním vynásobením  $2 \times 2$  matice a dvousložkového vektoru.

Pro větší soustavy se efekt ještě více projeví: maticový zápis umožňuje využít efektivní algoritmy pro násobení matic a vektorů a také tzv. SIMD paralelismus (Single Instruction, Multiple Data) na úrovni hardware, kde více násobení/sčítání probíhá souběžně [9].

## 3.2 Transformace vstupní úlohy na autonomní soustavu diferenciálních rovnic

Pro obecné nelineární diferenciální rovnice přímé použití Taylorovy metody naráží na obtížnost výpočtu derivací funkce  $\mathbf{f}(t, \mathbf{y})$  vyšších řádů. Pokud  $\mathbf{f}$  obsahuje nelinearity (např. součiny neznámých, mocniny, goniometrické či jiné funkce), derivováním se výraz rychle komplikuje. Řešením je transformovat původní problém do takové podoby, aby se vyšší derivace daly generovat pomocí algebraických operací obdobně jako v předchozí části [1, 15].

Principiálně jde o to rozložit složité nelineární vztahy na jednodušší prvky – typicky zavedením dodatečných proměnných, které reprezentují dílčí výrazy, a odvozením jejich vlastních diferenciálních rovnic. Cílem je získat ekvivalentní systém ODE, který neobsahuje „složité“ nelineární operace (např. násobení proměnných navzájem, složené funkce jako  $\sin(y)$ ,  $\ln(y)$  apod., nebo explicitní závislost na čase). Takový systém lze pak řešit Taylorovou metodou stejně snadno jako lineární soustavu: jeho pravá strana se bude skládat jen ze součtů a součinů známých veličin, což jsou algebraické operace, pro které umíme derivace generovat automaticky [5].

Nejprve je vhodné odstranit explicitní závislost na čase, aby byla soustava **autonomní**. Pro funkce jako  $\sin(t)$ ,  $\exp(t)$  či obecně jakékoliv známé časové funkce  $g(t)$ , které vystupují v pravé straně, zavedeme nové pomocné proměnné, jejichž vývoj popisuje jednoduchá (typicky lineární) ODE. Například pokud  $f(t, y)$  obsahuje člen  $\cos t$ , definujeme pomocnou proměnnou  $u(t) = \cos(t)$ . Její derivace je  $u'(t) = -\sin(t)$ . Abychom odstranili člen  $\sin t$ , zavedeme druhou proměnnou  $v(t) = \sin(t)$  s rovnicí  $v'(t) = \cos(t)$ , neboli  $v' = u$ . Tak získáme soustavu diferenciálních rovnic

$$\begin{aligned} u' &= -v, & u(0) &= u_0, \\ v' &= u, & v(0) &= v_0, \end{aligned} \quad (3.19)$$

což je homogenní lineární systém (harmonický oscilátor), který generuje hodnoty  $\cos(t)$  a  $\sin(t)$  pro příslušný čas. V původní rovnici pak  $\cos(t)$  nahradíme  $u$  (resp.  $\sin(t)$  nahradíme  $v$ , pokud by tam byla). Tímto způsobem zmizí explicitní  $t$  z pravé strany a místo něj přibudou nové rovnice pro  $u, v$ .

Dále se zaměříme na zpracování nelineárních členů, jako jsou součiny neznámých nebo jejich mocniny. Uvažujme obecnou nelineární soustavu

$$\mathbf{y}' = f(\mathbf{y}), \quad (3.20)$$

kde  $f(\mathbf{y})$  může obsahovat polynomiální členy (např. součiny složek  $\mathbf{y}$ ) a jiné nelinearity. Předpokládejme, že jsme schopni identifikovat **kvadratické** nelineární členy, tj. součiny dvou stavových veličin [25]. (Vyšší stupně by šly analogicky rozložit zavedením více pomocných proměnných, avšak kvadratické členy jsou nejčastější a bohatě demonstrují princip transformace.)

Označme  $y_i(t)$  a  $y_j(t)$  dvě složky stavu, jejichž součin se vyskytuje v  $f$ . Zavedeme novou proměnnou  $z_{ij}(t)$  definovanou jako

$$z_{ij}(t) = y_i(t) \cdot y_j(t). \quad (3.21)$$

Nyní odvodíme diferenciální rovnici pro  $z_{ij}(t)$ : derivací definice dostaneme

$$z'_{ij}(t) = \frac{d}{dt}[y_i(t) \cdot y_j(t)] = y'_i(t) \cdot y_j(t) + y_i(t) \cdot y'_j(t), \quad (3.22)$$

což je dle známého součinnového pravidla [27]. Nyní využijeme původní rovnice (3.20):  $y'_i = f_i(\mathbf{y})$  a  $y'_j = f_j(\mathbf{y})$ . Po dosazení do (3.22) získáme

$$z'_{ij}(t) = f_i(\mathbf{y}(t)) \cdot y_j(t) + y_i(t) \cdot f_j(\mathbf{y}(t)). \quad (3.23)$$

Tato rovnice obecně stále může obsahovat součin proměnných (pokud  $f_i$  či  $f_j$  obsahovaly jiné součiny). Avšak důležité je, že  $z'_{ij}$  je **vyjádřena pomocí nižších derivací**

**původních veličin** – konkrétně  $f_i$  a  $f_j$  představují právě pravé strany původních rovnic, které již známe (resp. budeme během výpočtu znát jejich hodnoty a derivace) [4].

Tímto způsobem tedy převádíme nelineární interakci  $y_i \cdot y_j$  na novou proměnnou  $z_{ij}$ , jejíž změna v čase je dána „lineární kombinací“ členů  $f_i \cdot y_j$  a  $y_i \cdot f_j$ . Pokud  $f_i$  a  $f_j$  byly čistě lineární výrazy, byla by i rovnice pro  $z_{ij}$  lineární; v opačném případě může obsahovat další součiny jiných stavových proměnných. Ty bychom opět mohli definovat jako nové proměnné a postup zopakovat.

Ve výsledku tak získáme rozšířený stavový vektor, který kromě původních proměnných  $\mathbf{y}$  obsahuje i nové proměnné pro každou identifikovanou nelineární kombinaci (součin, podíl, složenou funkci) [9]. Pokud původní systém obsahoval pouze polynomiální nelinearity omezeného stupně, tento proces může uzavřít soustavu bez potřeby nekonečně mnoha proměnných – typicky však počet nových proměnných výrazně narůstá s každým stupněm nelinearity.

V krajním případě (např.  $y' = y^2$ ) dochází ke zvýšení stupně polynomu po zavedení nové proměnné a proces by generoval stále složitější vztahy (tzv. Carlemanova linearizace vede na nekonečnou soustavu lineárních rovnic). V praxi se proto omezuje na nízké stupně nebo speciální struktury rovnic, kde je transformace zvládnutelná.

### 3.2.1 Kvadratické systémy a jejich řešení

Můj návrh řešení se zaměřuje zejména na kvadratické systémy, tj. ODE, jejichž pravá strana je nejvýše kvadratická funkce stavových proměnných. To zahrnuje mnoho praktických modelů (např. Lotka-Volterrův model predátor-kořist, logistická rovnice je také kvadratická  $y' = ay - by^2$ , atd.) [18].

Po zahrnutí pomocných proměnných pro případné vnější funkce ( $\sin t$ ,  $\exp t$ , atd.) můžeme kvadratický systém obecně zapsat jako

$$\mathbf{y}' = A\mathbf{y} + B_1\mathbf{q} + \mathbf{b}, \quad (3.24)$$

kde  $A$  je matice odpovídající čistě lineární části systému,  $\mathbf{b}$  vektor konstantních (nebo již transformací zavedených) členů a  $B_1\mathbf{q}$  značí kvadratické členy. Zde  $\mathbf{q}$  představuje vektor všech nezávislých kvadratických kombinací stavových proměnných a  $B_1$  je vhodná matice, která lineárně kombinuje tyto kvadratické členy do jednotlivých rovnic.

Například pokud máme jediný kvadratický člen  $y_1y_2$  v první rovnici, pak  $\mathbf{q} = [y_1y_2]$  a vektor  $B_1\mathbf{q}$  bude mít první složku  $b_{11}(y_1y_2)$  a ostatní složky nulové (tj.  $B_1$  má nenulový prvek  $b_{11}$  odpovídající koeficientu u  $y_1y_2$  v  $f_1$ ). Ve složitějších systémech  $\mathbf{q}$  zahrnuje všechny kombinace  $y_iy_j$  (pro  $i \leq j$ , aby se nezdvjovaly) a matice  $B_1$  obsahuje v každém řádku koeficienty, které u příslušných kombinací v dané rovnici vystupují.

Podstatné je, že takto rozšířený systém (3.24) má podobnou strukturu jako lineární soustava: pravá strana je součtem lineárního členu  $A\mathbf{y}$ , kvadratického členu  $B_1\mathbf{q}$  a případně konstanty  $\mathbf{b}$ . Derivování takovéto soustavy lze provádět algoritmicky [15, 1].

Postupujeme podobně jako u lineární soustavy, avšak musíme aplikovat pravidlo pro derivaci součinu. Z derivace (3.24) dostaneme

$$\mathbf{y}'' = A\mathbf{y}' + B_1\mathbf{q}'. \quad (3.25)$$

Pro výpočet  $\mathbf{q}'$  využijeme vztah (3.22), aplikovaný na každý součin  $y_iy_j$  v  $\mathbf{q}$ . Tím získáme  $\mathbf{q}'$  vyjádřené pomocí členů  $y_iy'_j$  a  $y'_iy_j$ , což jsou všechny výrazy známé z původní rovnice (resp. z  $\mathbf{y}'$ ). Výsledek lze opět zapsat ve tvaru podobném (3.24):

$$\mathbf{y}'' = A(A\mathbf{y} + B_1\mathbf{q} + \mathbf{b}) + B_1(\tilde{A}\mathbf{z} + \tilde{B}_1\mathbf{w}). \quad (3.26)$$

Zde  $\tilde{A}$  a  $\tilde{B}_1$  reprezentují lineární kombinace, které vzniknou při derivaci kvadratických členů – vektor  $\mathbf{z}$  může zahrnovat již známé součiny a  $\mathbf{w}$  případně nové třetího řádu (pokud by bylo třeba).

Není nutné zacházet do detailů této rovnice; důležité je, že jednotlivé členy  $\mathbf{y}''$  lze spočítat z hodnot  $\mathbf{y}$  a  $\mathbf{y}'$  pomocí operací s maticemi  $A$ ,  $B_1$  a kombinováním součinů. Obecně každý  $k$ -tý derivovaný člen  $\mathbf{q}^{(k)}$  vzniká jako součet členů typu  $y_i^{(r)} y_j^{(s)}$  pro  $r + s = k$  (což je aplikace Leibnizova pravidla pro  $k$ -tou derivaci součinu) [6].

Ve výsledku dostáváme pro  $k$ -tou derivaci  $\mathbf{y}^{(k)}$  výraz, který je složen lineárně z již vypočtených derivací nižších řádů. To znamená, že existuje rekurentní algoritmus, který z hodnot  $\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(k)}$  vypočte  $\mathbf{y}^{(k+1)}$  podobně snadno, jako tomu bylo v lineárním případě [5].

V moji implementaci tuto logiku zajišťuje právě maticové vyjádření (3.24) – pole indexů pro  $y_i y_j$  a odpovídající matice  $B_1$  nám umožňují programově (bez ručního derivování) generovat nové členy Taylorovy řady.

### 3.2.2 Ukázkový příklad transformace

Ačkoli se předchozí část zaměřila především na kvadratické nelinearity, následující ukázka demonstruje, že stejný postup lze aplikovat i na jiné typy členů, například goniometrické funkce.

Pro ilustraci si ukážeme transformaci nelineární diferenciální rovnice na soustavu autonomních obyčejných diferenciálních rovnic

$$y' = \sin(\sqrt{\cos t}), \quad y(0) = y_0. \quad (3.27)$$

Pravá strana obsahuje vnořené funkce  $\sqrt{\cos t}$ . Transformace začíná od nevnitřnější funkce. Nejprve tedy nahradíme funkci  $\cos t$  pomocnými rovnicemi

$$\begin{aligned} y_1 &= \cos(t), \\ y_1' &= -\sin(t) = -y_2, & y_1(0) &= \cos(0) = 1, \\ y_2 &= \sin(t), \\ y_2' &= \cos(t) = y_1, & y_2(0) &= \sin(0) = 0, \end{aligned} \quad (3.28)$$

Pro  $\sqrt{y_1}$  zavedeme novou proměnnou

$$\begin{aligned} y_3 &= \sqrt{y_1}, \\ y_3' &= \frac{1}{2\sqrt{y_1}} y_1' = \frac{1}{2y_3} y_1' = -\frac{1}{2y_3} y_2. \end{aligned} \quad (3.29)$$

Abychom se zbavili zlomků, definujeme  $y_4 = y_3^{-1}$ :

$$\begin{aligned} y_4 &= y_3^{-1}, \\ y_4' &= -y_3^{-2} y_3' = -y_3^{-2} \left(-\frac{1}{2} y_3^{-1} y_2\right) = \frac{1}{2} y_3^{-3} y_2 = \frac{1}{2} y_4^3 y_2. \end{aligned} \quad (3.30)$$

Z (3.29) a (3.30) snadno dostaneme

$$y_3' = -\frac{1}{2} y_4 y_2, \quad y_3(0) = y_1(0)^{\frac{1}{2}}. \quad (3.31)$$

Původní rovnice tak přechází na tvar

$$y' = \sin y_3. \quad (3.32)$$

Zavedeme další pomocné proměnné

$$\begin{aligned} y_5 &= \sin y_3, & y_6 &= \cos y_3, \\ y'_5 &= \cos y_3 y_3 = y_6 y_3, & y'_6 &= -\sin y_3 y'_3 = -y_5 y'_3. \end{aligned} \quad (3.33)$$

Dosazením všech předchozích vztahů vznikne výsledný systém sedmi rovnic prvního řádu

$$\begin{aligned} y' &= y_5, & y(0) &= y_0, \\ y'_1 &= -y_2, & y_1(0) &= \cos 0, \\ y'_2 &= y_1, & y_2(0) &= \sin 0, \\ y'_3 &= -\frac{1}{2}y_4 y_2, & y_3(0) &= \sqrt{\cos 0}, \\ y'_4 &= \frac{1}{2}y_4^3 y_2, & y_4(0) &= \frac{1}{\sqrt{\cos 0}}, \\ y'_5 &= y_6 y_3, & y_5(0) &= \sin(\sqrt{\cos 0}), \\ y'_6 &= -y_5 y'_3, & y_6(0) &= \cos(\sqrt{\cos 0}). \end{aligned} \quad (3.34)$$

Tento systém lze řešit jako autonomní soustavu ODE pomocí Taylorovy metody s maticovým přístupem.

## Kapitola 4

# Implementace a experimentální ověření

V této kapitole se práce zaměřuje na implementaci a srovnání několika numerických metod pro řešení ODE. Konkrétně se jedná o Taylorovou metodu, Eulerovou metodu, Runge-Kutta 4. řádu a o několik dalších metod. Implementace těchto metod byly provedeny v prostředí MATLAB a Python, přičemž hlavní důraz je kladen na efektivitu výpočtu, stabilitu a přesnost řešení.

### 4.1 Vstupní parametry a inicializace

Všechny implementované metody (Taylorova, Eulerova, RK4) používají vstupní parametry shrnuté v tabulce 4.1.

Pro porovnání jsou využity také adaptivní vestavěné ODE-řešiče uvedené v tabulce 4.2; u těchto metod se délka integračního kroku neurčuje explicitně parametrem  $h$ , ale algoritmus ji průběžně přizpůsobuje tak, aby splnil zadané relativní a absolutní tolerance (`RelTol/AbsTol` v MATLABu, resp. `rtol/atol` ve SciPy).

Parametr	Význam	Jednotky / typ	Metoda			
			Taylor (L)	Taylor (NL)	Euler	RK4
$A$	Matice lineární části systému	$\mathbb{R}^{n \times n}$	✓	✓	✓	✓
$b$	Vektor konstantních členů	$\mathbb{R}^n$	✓	✓	✓	✓
$B_1$	Kvadratické koeficienty	$\mathbb{R}^{n \times n \times n}$		✓		
$(i, j)$	Indexy proměnných v kvadratických členech	–		✓		
$y_0$	Počáteční podmínky	$\mathbb{R}^n$	✓	✓	✓	✓
$h$	Krok integrace	s	✓	✓	✓	✓
$t_{\text{final}}$	Konečný čas simulace	s	✓	✓	✓	✓
$\varepsilon$	Tolerance konvergence Taylorovy řady	–	✓	✓		
<b>maxTerms</b>	Max. počet členů Taylorovy řady	–	✓	✓		

Tabulka 4.1: Společné vstupní parametry a jejich využití v jednotlivých metodách

## 4.2 Implementace Taylorovy metody v MATLAB

Taylorova metoda je implementována pro lineární i nelineární systémy. Implementace se skládá z několika částí: výpočet derivací, sčítání členů Taylorovy řady a kontrola konvergence.

### 4.2.1 Lineární Taylorova metoda

Pro lineární diferenciální rovnice je metoda implementována ve skriptu `taylor_method.m`. Tento skript je zaměřen na výpočet diferenciálních rovnic ve tvaru:

$$\frac{dy}{dt} = A\mathbf{y} + \mathbf{b} \quad (4.1)$$

#### Algoritmus

- Inicializace vstupních parametrů:** Podrobný popis v tabulce 4.1
- Výpočet derivací:** V každém kroku výpočtu se počítají derivace dle následujícího vztahu:

$$\frac{d^k \mathbf{y}}{dt^k} = A^k \mathbf{y} + \sum_{j=0}^{k-1} A^j \mathbf{b}. \quad (4.2)$$

Pro každý krok  $n$  se iterativně přičítají jednotlivé členy Taylorovy řady, dokud není dosaženo stanovené přesnosti  $\varepsilon$ .

3. **Kontrola konvergence:** Při každém přičítání nového členu řady se kontroluje, zda velikost přičítaného členu je menší než stanovená tolerance. Pokud ano, výpočet se ukončí.
4. **Uložení výsledku:** Po ukončení výpočtu se výsledky ukládají do vektoru  $y_{\text{taylor}}$  pro každý časový bod.

#### 4.2.2 Nelineární Taylorova metoda

Nelineární Taylorova metoda byla implementována ve skriptu `taylor_method_nonlinear.m`. Tento skript zahrnuje kvadratické nelineární členy, což umožňuje řešit diferenciální rovnice následujícího tvaru:

$$\frac{d\mathbf{y}}{dt} = A\mathbf{y} + B_1(y_i y_j) + \mathbf{b} \quad (4.3)$$

Kde  $B_1$  je matice popisující nelineární interakce mezi proměnnými.

#### Algoritmus pro nelineární metodu

1. **Inicializace vstupních parametrů:** Podrobný popis v tabulce 4.1
2. **Výpočet derivací:** Výpočet zahrnuje nejen lineární členy, ale také kvadratické členy, které se počítají následovně:

$$\frac{d^k \mathbf{y}}{dt^k} = A^k \mathbf{y} + \sum_{j=0}^{k-1} A^j \mathbf{b} + B_1 \left( \sum_{l=0}^{k-1} \frac{d^l y_i}{dt^l} \frac{d^{k-1-l} y_j}{dt^{k-1-l}} \right). \quad (4.4)$$

3. **Rekurzivní výpočet:** Každý další člen Taylorovy řady se přičítá do celkového součtu, dokud není dosaženo požadované přesnosti.
4. **Optimalizace výpočtu:** Aby se předešlo zbytečnému výpočtu, kontroluje se, zda se přičítané členy postupně zmenšují. Pokud se dostatečně zmenší, výpočet je ukončen dříve.

### 4.3 Implementace Taylorovy metody v Python

Implementace Taylorovy metody v Pythonu byla provedena primárně za účelem porovnání její efektivity s implementací v MATLABu. Tato implementace byla následně optimalizována pomocí JIT kompilace, aby bylo možné zhodnotit vliv optimalizace na rychlost výpočtu a porovnat ji s ostatními metodami.

#### 4.3.1 Původní implementace Taylorovy metody v Pythonu

Původní implementace Taylorovy metody v Pythonu byla navržena tak, aby co nejdříve odpovídala MATLABovské verzi metody. Pro výpočet byly využity základní knihovny Pythonu jako `numpy` pro efektivní práci s maticemi a vektory. Algoritmus byl následující:

1. **Inicializace vstupních parametrů:** Podrobný popis v tabulce 4.1
2. **Výpočet derivací:** Každý další člen Taylorovy řady byl počítán pomocí vztahu jako v MATLABu 4.4.
3. **Rekurzivní výpočet:** Výpočet probíhá pomocí cyklu, kde se každý nový člen Taylorovy řady přičítá k předchozímu výsledku. Výpočet pokračuje až do dosažení požadované přesnosti  $\varepsilon$  nebo maximálního počtu členů `maxTerms`.
4. **Uložení výsledků:** Výsledky byly ukládány pro každý časový krok do vektorů a následně vykresleny pomocí knihovny `matplotlib`.

### 4.3.2 Optimalizace pomocí Just-In-Time kompilace

Po původní implementaci v Pythonu byla metoda optimalizována pomocí Just-In-Time (JIT) kompilace za použití knihovny `Numba`. Cílem bylo výrazně urychlit výpočet především u složitějších nelineárních systémů. Použití této optimalizace má několik výhod:

- **Zrychlení výpočtu:** JIT kompilace umožňuje přeložit části kódu přímo do strojového kódu, což vede k výraznému zrychlení.
- **Paralelní výpočet:** Díky možnosti použití příkazu `prange` lze paralelizovat výpočet, což vede ke zvýšení efektivity.
- **Efektivní práce s pamětí:** Díky optimalizaci knihovny `Numba` dochází ke zlepšení práce s maticemi a vektory.

Optimalizovaná implementace pomocí JIT kompilace probíhá následovně:

1. **Překlad funkce pomocí dekorátoru `@jit`.** Funkce, která provádí výpočet jednotlivých členů Taylorovy řady, je označena dekorátorem `@jit`, což umožňuje její přeložení přímo do strojového kódu.
2. **Využití paralelizace.** Pomocí `prange` je umožněno rozdělení výpočtu do více vláken, což umožňuje urychlit výpočet při použití vícejádrového procesoru.
3. **Optimalizace maticových operací.** Veškeré operace s maticemi jsou prováděny pomocí knihovny `numpy`, což zajišťuje efektivní práci s pamětí a rychlé operace.

Tento postup umožnil značné urychlení výpočtu v porovnání s původní implementací.

## 4.4 Implementace metod pro porovnání nástrojů MATLAB a Python (Euler a Runge-Kutta)

V této části jsou popsány implementace základních numerických metod pro řešení obyčejných diferenciálních rovnic, které slouží jako referenční metody pro porovnání s Taylorovou metodou. Konkrétně se jedná o Eulerovu metodu a metodu Runge-Kutta 4. řádu. Implementace těchto metod jsou provedeny jak v prostředí MATLAB, tak v Pythonu. Jelikož se jedná o běžně používané metody, jsou jejich algoritmy velmi podobné v obou prostředích.

#### 4.4.1 Eulerova metoda

Eulerova metoda je základní numerická metoda prvního řádu pro řešení obyčejných diferenciálních rovnic. Její jednoduchost spočívá v přibližném výpočtu řešení pomocí lineárního rozvoje funkce.

##### Algoritmus

1. **Inicializace vstupních parametrů:** Podrobný popis v tabulce 4.1. Velikost integračního kroku  $h$  zároveň určuje přesnost metody – globální chyba explicitní Eulerovy metody je řádu  $O(h)$ , takže zmenšením  $h$  lze dosáhnout vyšší přesnosti (na rozdíl od Taylorovy metody, kde se konvergence řídí tolerancí  $\varepsilon$ ).

2. **Výpočet diferenciálních rovnic pomocí Eulerovy metody:**

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h(\mathbf{A}\mathbf{y}_n + \mathbf{b}). \quad (4.5)$$

3. **Uložení výsledků:** Výsledky jsou ukládány pro každý časový krok do matice  $Y$ , kde každý sloupec odpovídá hodnotě v daném čase.

4. **Iterace přes všechny časové kroky:** Algoritmus se opakuje až do dosažení konečného času  $t_{\text{final}}$ .

#### 4.4.2 Metoda Runge-Kutta 4. řádu

Metoda Runge-Kutta 4. řádu (RK4) patří mezi klasické metody pro řešení ODE. Jedná se o metodu čtvrtého řádu s dobrou přesností a stabilitou, což ji činí vhodnou pro porovnání s Taylorovou metodou.

##### Algoritmus

1. **Inicializace vstupních parametrů:** Podrobný popis v tabulce 4.1. Velikost kroku  $h$  současně určuje i přesnost metody: globální chyba klasické metody Runge-Kutta čtvrtého řádu klesá jako  $O(h^4)$ , takže zmenšením  $h$  lze dosáhnout výrazně rychlejšího zpřesnění než u Eulerovy metody (řád  $O(h)$ ).

2. **Výpočet diferenciálních rovnic pomocí metody RK4:**

- (a) Výpočet mezikroků:

$$\begin{aligned} k_1 &= \mathbf{A}\mathbf{y}_n + \mathbf{b}, \\ k_2 &= \mathbf{A}\left(\mathbf{y}_n + \frac{h}{2}k_1\right) + \mathbf{b}, \\ k_3 &= \mathbf{A}\left(\mathbf{y}_n + \frac{h}{2}k_2\right) + \mathbf{b}, \\ k_4 &= \mathbf{A}(\mathbf{y}_n + hk_3) + \mathbf{b}. \end{aligned} \quad (4.6)$$

- (b) Aktualizace hodnoty  $y$ :

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \quad (4.7)$$

3. **Uložení výsledků:** Výsledky jsou ukládány pro každý časový krok do matice  $Y$ .

4. **Iterace přes všechny časové kroky:** Výpočet pokračuje až do dosažení konečného času  $t_{\text{final}}$ .

### 4.4.3 Implementace pro MATLAB a Python

Implementace v MATLABu i Pythonu probíhá obdobným způsobem. Rozdíly spočívají především v syntaxi obou jazyků a v použití knihoven:

- **MATLAB:** Výpočty jsou prováděny pomocí maticových operací, což je výhodné zejména pro lineární systémy.
- **Python:** Implementace používá knihovnu `numpy` pro efektivní práci s maticemi a vektory. Pro následnou optimalizaci byly použity knihovny `Numba`.

Obě metody (Eulerova a Runge-Kutta) jsou implementovány v MATLABu a Pythonu. Jelikož algoritmus je totožný, výsledky jsou vhodné pro porovnání s Taylorovou metodou.

## 4.5 Přehled vestavěných adaptivních řešičů

V následující tabulce se nachází stručný popis jednotlivých metod, které byly použity pro porovnání.

Řešič	Stručný popis metody
<i>MATLAB</i>	
ode23	Bogacki–Shampine, explicitní RK řádu 2/3 [2]
ode45	Dormand–Prince, explicitní RK řádu 4/5 [7]
ode78	Explicitní Runge–Kutta řádu 7/8 [9]
ode89	Explicitní Runge–Kutta řádu 8/9 [9]
ode113	Adams–Bashforth–Moulton, PECE schéma (řád 1–13) [19]
<i>Python (SciPy)</i>	
RK23	Bogacki–Shampine, explicitní RK řádu 2/3 [2]
RK45	Dormand–Prince, explicitní RK řádu 4/5 [7]
DOP853	Dormand–Prince 8(5,3), explicitní RK řádu 8 [7]
LSODA	LSODA – Adams / BDF s automatickou detekcí tuhosti [10]

Tabulka 4.2: Seznam vestavěných adaptivních řešičů obyčejných diferenciálních rovnic, které jsou v práci využity pro porovnání s Taylorovou metodou.

## 4.6 Zabudované řešiče v MATLABu

Prostředí MATLAB nabízí vestavěné adaptivní řešiče pro netuhé (nonstiff) ODE. Nejpopulárnější jsou `ode23` a `ode45`, které využívají explicitní Runge-Kutta metody s vloženými vzorci 3. a 5. řádu pro odhad lokální chyby. Konkrétně řešič `ode23` je založen na páru formulí 2. a 3. řádu (Bogacki-Shampine) [21, 2], zatímco `ode45` využívá složitější pár 4. a 5. řádu (Dormand-Prince) [22, 7]. V každém kroku se vypočítají dvě aproximace řešení různého řádu a z jejich rozdílu se stanoví lokální chyba, podle které řešič automaticky upravuje délku kroku integrace tak, aby dodržel danou toleranci.

V novějších verzích MATLABu přibýly metody vyššího řádu `ode78` a `ode89`, které využívají explicitní metody RK řádu 7/8 a 8/9. Tyto metody dokážou pro hladká řešení

s vysokými nároky na přesnost dosáhnout vyšší efektivity než `ode45` [9]. Pro velmi přesné integrace hladkých netuhých systému může být efektivnější `ode78` a `ode89` je pak výhodnější u extrémně hladkých systému na dlouhých intervalech nebo při velmi přísné toleranci [23, 24].

Odlisný přístup má víceukrokový řešič `ode113`, který implementuje Adams-Bashforth-Moultonův prediktor-korektor s proměnným krokem a řádem. `Ode113` adaptivně volí řád metody od 1 do 13 podle potřeby [19]. Jedná se o PECE algoritmus (Predict-Evaluate-Correct-Evaluate), kde explicitní Adams-Bashforth prediktor odhaduje nový bod a poté implicitní Adams-moulton korektor zpřesní řešení s využitím předchozí predikce. Díky tomu není nutné při korekci řešit nelineární rovnice iterativně od nuly – korekce využije známe derivace z prediktoru. Řešič `ode113` efektivně spojuje výhody explicitních a implicitních metod: využívá informace z několika předchozích kroků a současně zachovává automatickou kontrolu chyby a kroku integrace jako jednokrokové RK metody [19]. Na začátku výpočtu provede `ode113` potřebné zaváděcí kroky nižšího řádu (dokud nemá k dispozici dostatek bodů pro vícečlenný vzorec) a poté dynamicky upravuje jak velikost kroku, tak řád metody.

Všechny uvedené MATLAB řešiče (`ode23`, `ode45`, `ode78`, `ode89` i `ode113`) jsou navrženy pro netuhé úlohy – v případě výskytu tuhosti (stiffness) mohou selhávat nebo být neúměrně pomalé, a je proto vhodné použít alternativní metody pro tuhé ODE, například `ode15s` [20].

## 4.7 Testovací příklady

V této části jsou popsány výsledky testování implementovaných metod na vybraných experimentech.

### 4.7.1 Výsledky lineárního problému (sinus a kosinus) v jazyce Python

Pro testování byl zvolen jednoduchý lineární oscilátor  $y' = Ay$ , kde

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad y(0) = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

tedy harmonický oscilátor s analytickým řešením

$$y_{\text{analyt}}(t) = \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}.$$

Metody byly testovány ve třech různých konfiguracích, které se lišily nastavením přesnosti a kroků. Výsledky jsou shrnuty v tabulkách 4.3–4.5.

Z tabulky 4.3 vyplývá, že Taylorova metoda slouží jako referenční. I když není nejpřesnější, má solidní kompromis mezi časem a přesností. Naproti tomu metoda RK4 dosáhla extrémně nízké chyby, ale za cenu více než šestnásobného času. Adaptivní řešič `DOP853` dosáhl velmi nízkého času a chyby pod  $10^{-2}$ , a byl tedy v tomto nastavení nejrychlejší.

Při zpřísnění toleranci (tabulka 4.4) se výrazně zlepšila přesnost adaptivních řešičů, zejména u `DOP853` a `LSODA`. Taylorova metoda ale zůstala nejrychlejší, i když její chyba se nezměnila. Z toho plyne, že pro požadovanou přesnost kolem  $10^{-7}$  je Taylorova metoda stále výhodná, pokud chyba v řádu  $10^{-3}$  postačuje.

Metoda	Počet kroků	Chyba	Čas [s]	Poměr
MTSM	500	2.02809e-03	0.0148015	1
Euler	5000	2.84016e-01	0.0285306	1.92755
RK4	5000	4.16667e-09	0.0923352	6.23823
RK23	120	1.46297e-01	0.00516272	0.348797
RK45	46	3.50462e-02	0.00304627	0.205808
DOP853	21	8.20996e-03	0.00220966	0.196579
LSODA	167	8.76407e-02	0.00316978	0.214152

Tabulka 4.3: První sada výsledků ( $TOL = 10^{-3}$ ,  $MTSM_{TOL} = 10^{-3}$ ,  $MTSM_h = 0.1$  s,  $h = 0.01$  s).

Metoda	Počet kroků	Chyba	Čas [s]	Poměr
MTSM	500	2.02809e-03	0.0130312	1
Euler	5000	2.84016e-01	0.0250142	1.91955
RK4	5000	4.16666e-09	0.100543	7.71557
RK23	2628	1.46339e-05	0.109131	8.37457
RK45	292	2.16023e-06	0.019382	1.48735
DOP853	61	2.79079e-07	0.00703979	0.540224
LSODA	322	3.91432e-06	0.00506163	0.388422

Tabulka 4.4: Druhá sada výsledků ( $TOL = 10^{-7}$ ,  $MTSM_{TOL} = 10^{-3}$ ,  $MTSM_h = 0.1$  s,  $h = 0.01$  s).

Metoda	Počet kroků	Chyba	Čas [s]	Poměr
MTSM	5	7.12078e-05	0.00146699	1
Euler	5000	2.84016e-01	0.0245295	16.7209
RK4	5000	4.16666e-09	0.093372	63.8859
RK23	120	1.46297e-01	0.00455332	3.10385
RK45	46	3.50647e-02	0.00268173	1.82805
DOP853	21	8.00147e-03	0.00221133	1.50739
LSODA	167	8.76407e-02	0.00227547	1.55111

Tabulka 4.5: Třetí sada výsledků ( $TOL = 10^{-7}$ ,  $MTSM_{TOL} = 10^{-4}$ ,  $MTSM_h = 10$  s,  $h = 0.01$  s).

Ve třetím testu (tabulka 4.5) Taylorova metoda dosáhla pouze 5 kroků a stále velmi malé chyby ( $\approx 7 \times 10^{-5}$ ). Výpočetní čas byl zároveň nejnižší ze všech metod. Ostatní metody vykazovaly stabilní časy, ale byly výrazně pomalejší než MTSM.

Výsledky testů lineárního oscilátoru ukazují, že pro tento typ úloh je Taylorova metoda velmi efektivní z hlediska poměru přesnosti a výpočetního času, zejména při vhodném nastavení kroku a tolerance. V případě lineárních problémů Taylorova metoda dosahuje dobrých výsledků i bez optimalizace JIT kompilací, na rozdíl od nelineárních systémů, kde je optimalizace zásadní pro dosažení konkurenceschopných výpočetních časů, jak bylo ukázáno v sekci o Lorenzově systému.

#### 4.7.2 Automatické nastavení řádu (ORD)

V této části je prezentováno experimentální ověření efektivity automatického nastavení řádu (ORD) pomocí Taylorovy metody. Pro experiment byl zvolen testovací příklad, který řeší soustavu diferenciálních rovnic:

$$\begin{aligned} y' &= z, & y(0) &= 0, \\ z' &= -y, & z(0) &= 5, \end{aligned}$$

s analytickým řešením:

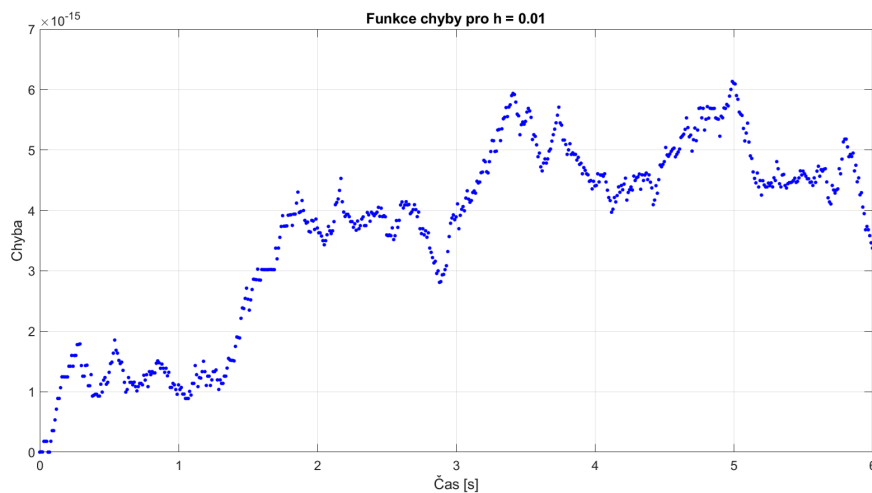
$$y = 5 \sin(t), \quad z = 5 \cos(t).$$

Při experimentu byly použity hodnoty:

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

#### Výsledky experimentu

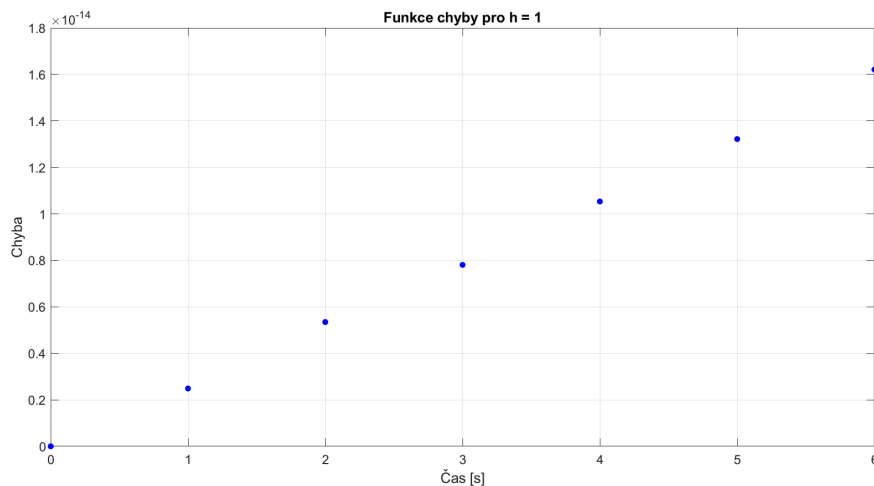
Pro numerické řešení byla použita přesnost  $\varepsilon = 10^{-12}$  a byly zkoumány různé hodnoty kroku  $h$ . Byla analyzována především funkce chyby, která ukazuje absolutní chybu oproti analytickému řešení.



Obrázek 4.1: Funkce chyby pro  $h = 0.01$  s.

## Diskuse výsledků

Z grafu na obrázku 4.1 je patrné, že absolutní chyba je velmi malá (řádově  $10^{-14}$ ), což odpovídá přesnosti blízké strojové přesnosti (tzv. numerické nule) a dokazuje vysokou přesnost metody. Při výpočtu s krokem  $h = 0.01$  s byla hodnota ORD konstantní a rovna 6 pro všechny kroky výpočtu.



Obrázek 4.2: Funkce chyby pro  $h = 1$  s.

Při kroku  $h = 1$  s (viz obrázek 4.2) je chyba mírně vyšší (řádově  $10^{-13}$ ). Pro tento větší krok byla hodnota ORD také konstantní a rovna 8 pro všechny kroky výpočtu. Toto zvýšení počtu členů Taylorovy řady při větším kroku je logické, protože s rostoucí velikostí kroku je potřeba více členů rozvoje pro zachování požadované přesnosti výpočtu.

### 4.7.3 Kruhový test a porovnání metod

V následujícím experimentu byly srovnány různé numerické metody na příkladu kruhového testu. Sledována byla přesnost jednotlivých metod a počet členů Taylorova rozvoje (ORD) při použití Taylorovy metody. Řešená soustava diferenciálních rovnic byla definována jako:

$$\begin{aligned}y' &= \omega z, & y(0) &= 0, \\z' &= -\omega y, & z(0) &= 1,\end{aligned}$$

s analytickým řešením:

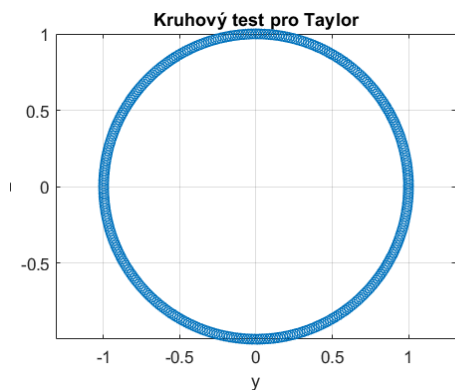
$$y = \sin(\omega t), \quad z = \cos(\omega t).$$

Parametry experimentu byly následující:

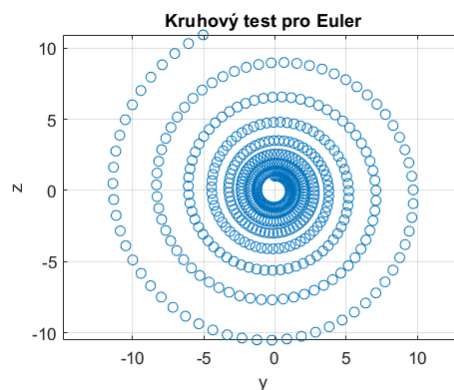
- Frekvence  $\omega = 1 \text{ rad} \cdot \text{s}^{-1}$ ,
- časový interval  $t \in \langle 0, 50 \rangle \text{ s}$ ,
- krok  $h = 0.1 \text{ s}$ ,
- přesnost  $\text{TOL} = 10^{-6}$ .

## Výsledky experimentu

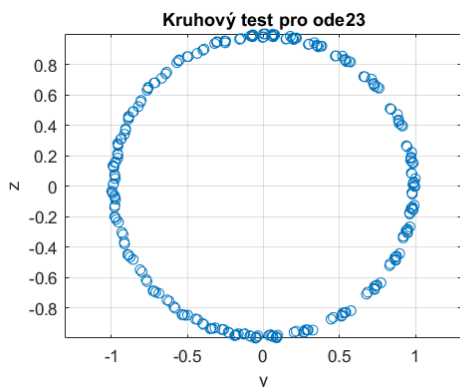
Na následujících obrázcích jsou prezentovány výsledky kruhového testu a hodnoty ORD. Pro každý řešič byly vykresleny trajektorie  $y(t)$  proti  $z(t)$ , které by při ideálním řešení měly tvořit kruh. Analytické řešení bylo vykresleno samostatně pro porovnání.



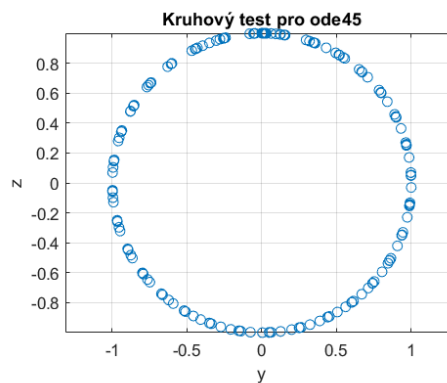
(a) Kruhový test pro Taylorovu metodu.



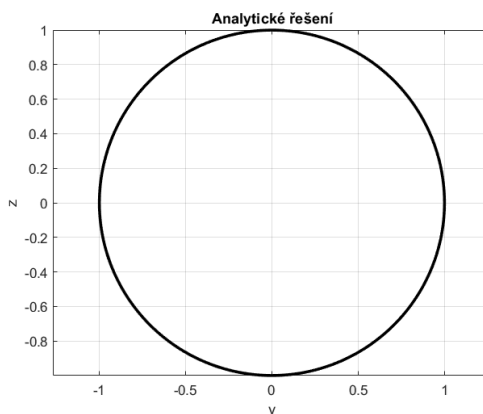
(b) Kruhový test pro Eulerovu metodu.



(c) Kruhový test pro metodu ode23.



(d) Kruhový test pro metodu ode45.



(e) Analytické řešení kruhového testu.

## Diskuse výsledků

Na základě výsledků kruhového testu (viz Obrázky 4.3a–4.3d) je zřejmé, že Taylorova metoda, metoda Runge-Kutta a adaptivní metody MATLAB ode45, ode78, ode89 a ode113

vykazují vysokou přesnost. Jejich trajektorie se velmi blíží analytickému řešení (viz Obrázek 4.3e). Naopak Eulerova metoda vykazuje značnou nepřesnost, což je patrné z deformace trajektorie do spirály.

Při výpočtu pomocí Taylorovy metody se ukázalo, že metoda používá konstantní počet členů  $\text{ORD} = 5$  po celou dobu simulace. Tato vlastnost potvrzuje stabilitu metody při řešení tohoto lineárního problému.

### Výsledky kruhového testu: Porovnání řešičů

V následujících tabulkách jsou shrnuty výsledky kruhového testu pro  $\omega = 1 \text{ rad} \cdot \text{s}^{-1}$  a časový interval  $t \in \langle 0, 50 \rangle$ . Byla porovnávána Taylorova metoda, Eulerova metoda metoda Runge-Kutta a vybrané `odeXX` řešiče z prostředí MATLAB.

Sledovanými kritérii jsou:

- **Počet kroků** – počet kroků, které řešič použil,
- **Chyba** – maximální chyba oproti analytickému řešení,

$$\text{Chyba} = \max_t \|\mathbf{y}(t) - \mathbf{y}_{\text{analytical}}(t)\|_2 \quad (4.8)$$

kde  $\mathbf{y}(t)$  je numerické řešení a  $\mathbf{y}_{\text{analytical}}(t)$  je analytické řešení v čase  $t$ . Pro kruhový test je analytickým řešením kombinace funkcí sinus a kosinus:

$$\mathbf{y}_{\text{analytical}}(t) = \begin{bmatrix} \sin(\omega t) \\ \cos(\omega t) \end{bmatrix} \quad (4.9)$$

- **Čas [s]** – doba výpočtu,
- **Poměr** – poměr času výpočtu vůči času Taylorovy metody,

$$\text{Poměr} = \frac{\text{čas výpočtu řešiče}}{\text{čas výpočtu Taylorovy metody}} \quad (4.10)$$

Pokud má Poměr hodnotu větší než 1, znamená to, že daný řešič je pomalejší než Taylorova metoda.

Metoda	Počet kroků	Chyba	Čas [s]	Poměr
MTSM	501	6.9439e-07	0.0008679	1
Euler	5001	0.28402	0.0010828	1.2476
RK4	5001	4.1667e-09	0.0038001	4.3785
ode23	169	0.054813	0.0011718	1.3502
ode45	241	0.0074474	0.0007982	0.91969
ode78	201	0.0074212	0.0010434	1.2022
ode89	241	0.00023394	0.0011876	1.3684
ode113	106	0.030287	0.0012326	1.4202

Tabulka 4.6: První sada výsledků ( $\text{TOL} = 10^{-3}$ ,  $\text{MTSM}_{\text{TOL}} = 10^{-6}$ ,  $\text{MTSM}_h = 0.1 \text{ s}$ ,  $h = 0.01 \text{ s}$ ).

Metoda RK4 dosahuje nejvyšší přesnosti (chyba řádově  $10^{-9}$ ), ale za cenu nejdelšího výpočetního času, což odpovídá poměru 4.3785 vůči Taylorově metodě. Metoda `ode45` je

jediná, která běží rychleji než Taylorova metoda, ale není zdaleka tak přesná. Eulerova metoda je sice relativně rychlá, ale má poměrně vysokou chybu (řádově  $10^{-1}$ ).

Metoda	Počet kroků	Chyba	Čas [s]	Poměr
MTSM	501	6.9439e-07	0.00108300	1
Euler	5001	0.28402	0.00116790	1.0784
RK4	5001	4.1667e-09	0.00385770	3.562
ode23	3575	5.7474e-06	0.00825680	7.624
ode45	1485	6.3518e-07	0.00191050	1.7641
ode78	593	8.6392e-08	0.00186660	1.7235
ode89	577	1.9062e-08	0.00216070	1.9951
ode113	260	1.836e-06	0.00251830	2.3253

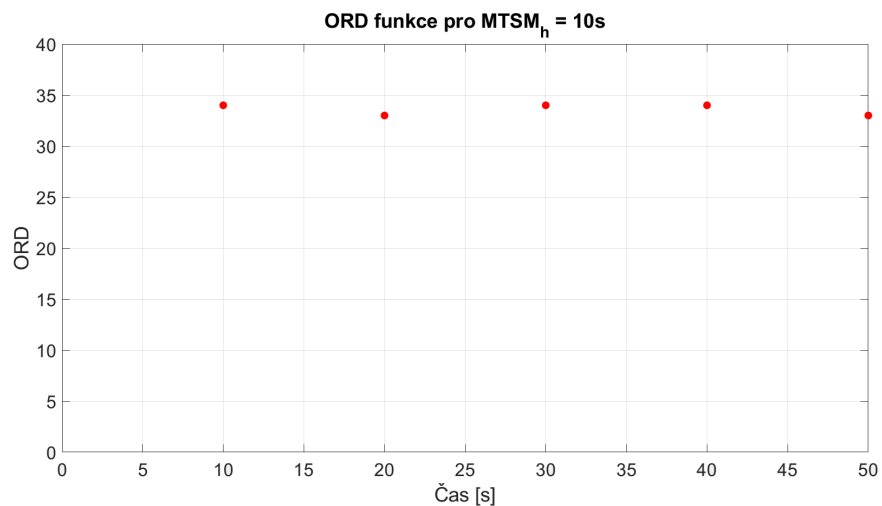
Tabulka 4.7: Druhá sada výsledků ( $TOL = 10^{-7}$ ,  $MTSM_{TOL} = 10^{-6}$ ,  $MTSM_h = 0.1$  s,  $h = 0.01$  s).

Tabulka 4.7 ukazuje srovnání metod při vyšší požadované přesnosti u vestavěných metod ( $TOL = 10^{-7}$ ). Je vidět, že všechny adaptivní řešiče (ode23, ode45, ode78, ode89, ode113) výrazně zvýšily počet kroků oproti předchozímu testu, což vedlo k lepší přesnosti, ale za cenu delšího výpočetního času. Metoda ode89 dosahuje nejlepší přesnosti, zatímco RK4 si udržuje vysokou přesnost při pevném počtu kroků, ale s třikrát delším časem výpočtu než Taylorova metoda. Metoda ode23 vykazuje nejvyšší poměr času, což naznačuje nižší efektivitu při požadavcích na vyšší přesnost.

Metoda	Počet kroků	Chyba	Čas [s]	Poměr
MTSM	6	7.1208e-05	9.9e-05	1
Euler	5001	0.28402	0.0016047	16.209
RK4	5001	4.1667e-09	0.0037523	37.902
ode23	3575	5.7474e-06	0.0082153	82.983
ode45	1485	6.3518e-07	0.0019629	19.827
ode78	593	8.6392e-08	0.0016759	16.928
ode89	577	1.9062e-08	0.0018621	18.809
ode113	260	1.836e-06	0.0024002	24.244

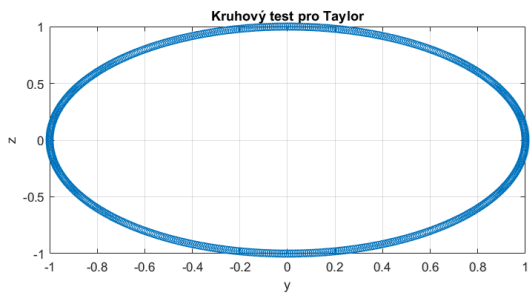
Tabulka 4.8: Třetí sada výsledků ( $TOL = 10^{-7}$ ,  $MTSM_{TOL} = 10^{-4}$ ,  $MTSM_h = 10$  s,  $h = 0.01$  s).

V tabulce 4.8 jsou patrné extrémní odchylky v počtu kroků a chybě – Taylorova metoda při velmi velkém kroku ( $MTSM_h = 10$  s) provede jen 6 kroků, ale chyba je větší než v předchozích experimentech ( $7.12 \times 10^{-5}$ ). Metoda RK4 dosahuje stále nejlepší přesnosti ( $4.17 \times 10^{-9}$ ), ale za cenu výrazně vyššího času výpočtu ( $37.9 \times$  delší než Taylorova metoda). Zajímavé je, že ode23 má extrémně vysoký poměr času v porovnání s ostatními metodami. Adaptivní řešiče drží chybu o několik řádů níže než Taylorova metoda, ale cenou je výrazně delší výpočetní čas u všech metod (Poměr  $> 16$  pro všechny). Při tomto nastavení se počet použitých členů Taylorovy řady zvyšuje, což je vidět na obrázku 4.4.

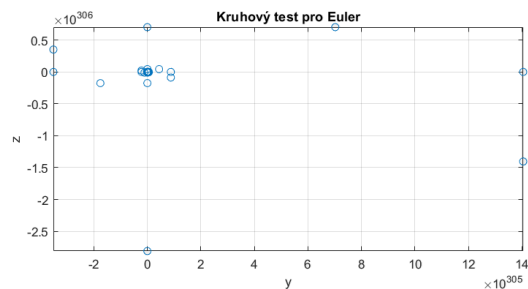


Obrázek 4.4: ORD funkce pro  $MTSM_h = 10s$ .

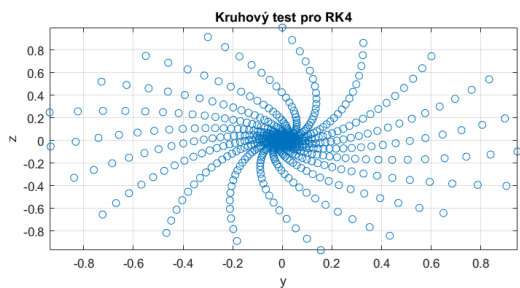
Další sada experimentů ukazuje chování systému pro  $\omega = 100 \text{ rad} \cdot \text{s}^{-1}$ . Experiment je proveden s parametry  $TOL = 1 \times 10^{-3}$ ,  $MTSM_{TOL} = 1 \times 10^{-6}$  a krokem  $h = 0.01 \text{ s}$ . Výsledky kruhových testů pro všechny numerické metody jsou zobrazeny na obrázcích 4.5.



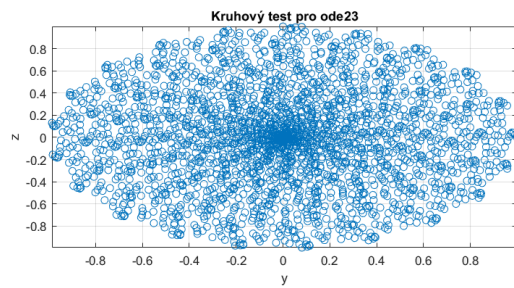
(a) Kruhový test pro Taylorovu metodu.



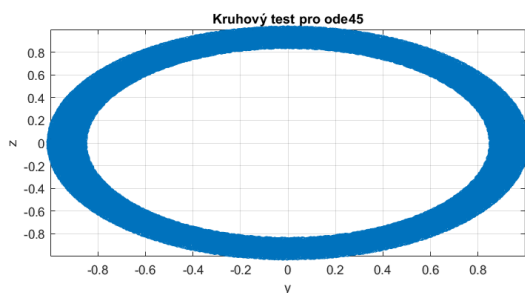
(b) Kruhový test pro Eulerovu metodu.



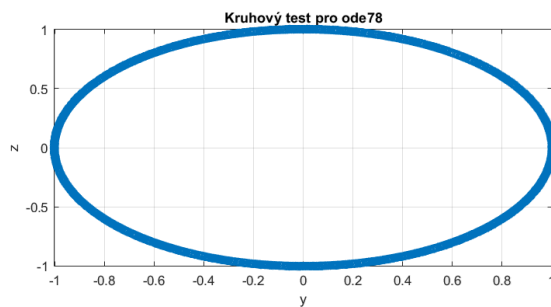
(c) Kruhový test pro metodu RK4.



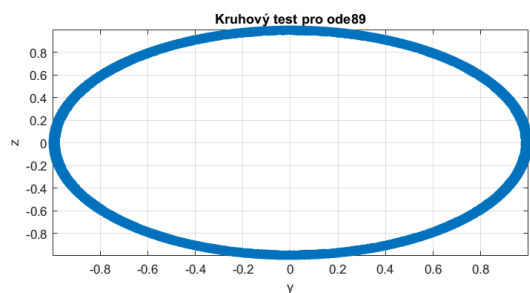
(d) Kruhový test pro řešič ode23.



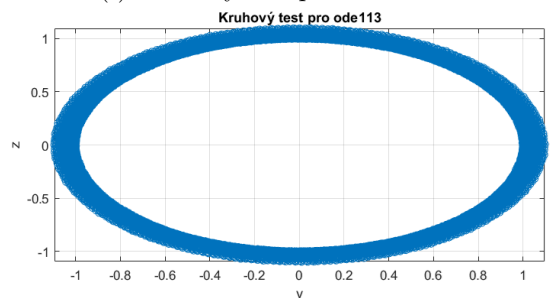
(e) Kruhový test pro řešič ode45.



(f) Kruhový test pro řešič ode78.

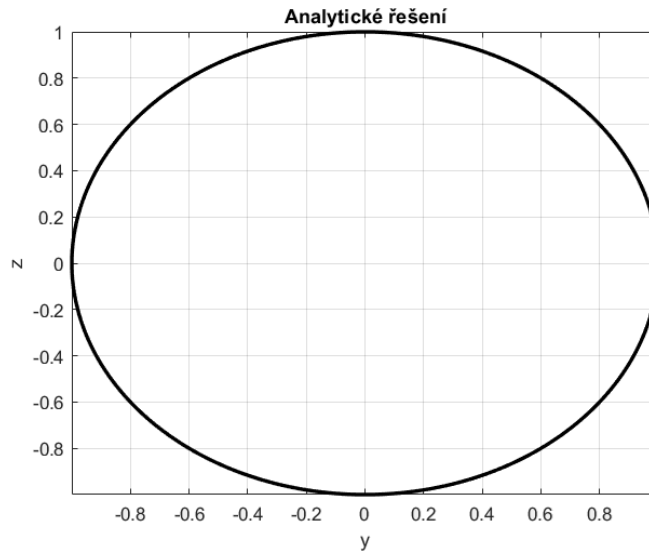


(g) Kruhový test pro řešič ode89.



(h) Kruhový test pro řešič ode113.

Obrázek 4.5: Kruhové testy pro adaptivní řešiče ode45, ode78, ode89 a ode113 při  $\omega = 100 \text{ rad} \cdot \text{s}^{-1}$ ,  $TOL = 1 \times 10^{-3}$ ,  $MTSM_{TOL} = 1 \times 10^{-6}$ ,  $h = 0.01 \text{ s}$ .



Obrázek 4.6: Analytické řešení kruhového testu pro  $\omega = 100 \text{ rad} \cdot \text{s}^{-1}$ .

Metoda	Počet kroků	Chyba	Čas [s]	Poměr
MTSM	5001	0.0001249	0.0099037	1
Euler	5001	Inf	0.0010456	0.10558
RK4	5001	1.0554	0.0037301	0.37301
ode23	4855	1.011	0.010931	1.1037
ode45	23201	0.73986	0.024981	2.5224
ode78	18777	0.0044686	0.043699	4.4124
ode89	23113	0.020748	0.066189	6.6832
ode113	9889	2.0869	0.061666	6.2266

Tabulka 4.9: Výsledky numerického řešení systému s  $\omega = 100 \text{ rad} \cdot \text{s}^{-1}$ ,  $TOL = 1 \times 10^{-7}$ ,  $MTSM_{TOL} = 1 \times 10^{-6}$ ,  $h = 0.01 \text{ s}$ .

Tabulka 4.9 ukazuje výsledky numerického řešení systému s vysokou frekvencí. Z výsledků je patrné, že Taylorova metoda dosahuje nejlepší přesnosti ze všech testovaných metod. Pro toto řešení Taylorova metoda využívá konstantně 10 členů řady (ORD=10) v každém kroku, což zajišťuje dostatečnou přesnost i při takto vysoké frekvenci. Eulerova metoda při této frekvenci zcela selhává, což je indikováno nekonečnou chybou (Inf). Metoda RK4, přestože používá stejný počet kroků jako Taylorova metoda, vykazuje výrazně vyšší chybu.

Zajímavé je také chování adaptivních metod. Přestože tyto metody automaticky přizpůsobují velikost kroku, všechny kromě metody ode78 mají poměrně vysokou chybu. Metoda ode45 použila nejvíce kroků, ale její chyba je stále relativně vysoká. Metoda ode78 dosahuje druhé nejlepší přesnosti, ale za cenu více než čtyřnásobného času výpočtu oproti Taylorově metodě.

Z časového hlediska je Eulerova metoda nejrychlejší, ale její výsledky jsou nepoužitelné. Taylorova metoda představuje optimální kompromis mezi přesností a výpočetní náročností. Adaptivní metody vysokého řádu nám dávají dobrou přesnost, ale za cenu delšího času potřebnou na výpočet.

#### 4.7.4 Výpočet koeficientů Fourierovy řady

Následující příklad ukazuje výpočet koeficientů Fourierovy řady pomocí soustavy diferenciálních rovnic. Fourierova řada představuje významný matematický nástroj, který umožňuje reprezentovat periodickou funkci jako nekonečnou sumu harmonických funkcí. Tento rozklad má uplatnění ve velké škále aplikací, například analýza signálů nebo řešení parciálních diferenciálních rovnic.

Libovolnou periodickou funkci  $f(t)$  s periodou  $T$  lze vyjádřit pomocí Fourierovy řady ve tvaru:

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\omega t) + \sum_{k=1}^{\infty} b_k \sin(k\omega t), \quad (4.11)$$

kde koeficienty Fourierovy řady jsou definovány následujícími integračními vztahy:

$$a_0 = \frac{2}{T} \int_0^T f(t) dt, \quad (4.12)$$

$$a_k = \frac{2}{T} \int_0^T f(t) \cos(k\omega t) dt, \quad k = 1, 2, 3, \dots, \quad (4.13)$$

$$b_k = \frac{2}{T} \int_0^T f(t) \sin(k\omega t) dt, \quad k = 1, 2, 3, \dots, \quad (4.14)$$

přičemž  $\omega = \frac{2\pi}{T}$  je úhlová frekvence.

#### Transformace na počáteční úlohu

Výpočet definičního integrálu

$$Y = \int_0^T f(x) dx \quad (4.15)$$

lze převést na řešení počáteční úlohy:

$$y' = f(x), \quad y(0) = 0, \quad (4.16)$$

kde hodnota řešení  $y(T)$  v čase  $T$  odpovídá přímo hledané hodnotě integrálu  $Y$ . Analogicky lze postupovat i při výpočtu koeficientů Fourierovy řady, které lze formulovat jako počáteční úlohy:

$$a'_k = \frac{2}{T} f(t) \cos(k\omega t), \quad a_k(0) = 0, \quad (4.17)$$

kde řešení  $a_k(T)$  reprezentuje hodnotu koeficientu  $A_k$ .

#### Testovací příklad

Pro verifikaci efektivity numerických metod byl zvolen jednoduchý modelový příklad funkce

$$f(t) = \sin^2(\omega t) \quad (4.18)$$

s parametry  $\omega = \frac{2\pi}{T} = \pi \text{ rad}\cdot\text{s}^{-1}$  pro periodu  $T = 2 \text{ s}$ .

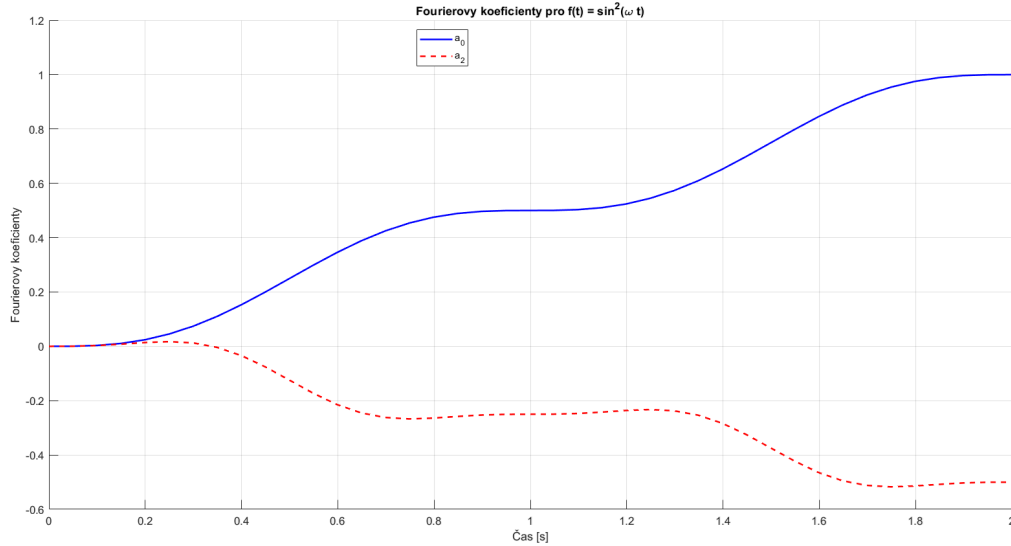
Pro tuto specifickou funkci má Fourierova řada pouze dva nenulové koeficienty, a lze ji zapsat ve tvaru:

$$\sin^2(\omega t) = \frac{a_0}{2} + a_2 \cos(2\omega t) \quad (4.19)$$

Analytické řešení dává:

$$\sin^2(\omega t) = \frac{1}{2} - \frac{1}{2} \cos(2\omega t) \quad (4.20)$$

kde  $A_0 = 1$  a  $A_2 = -\frac{1}{2}$ .



Obrázek 4.7: Časový vývoj hodnot koeficientů Fourierovy řady  $a_0$  a  $a_2$  během numerické integrace pro funkci  $f(t) = \sin^2(\omega t)$ . Graf ilustruje konvergenci koeficientů k jejich analytickým hodnotám  $A_0 = 1$  a  $A_2 = -0.5$  v čase  $t = 2$  s.

## Implementace systému ODE

Pro efektivní numerické řešení byl problém formulován jako soustava 11 lineárních diferenciálních rovnic prvního řádu:

$$\begin{aligned} y_1' &= y_3, & y_1(0) &= 0, \\ y_2' &= \frac{2}{T}y_6, & y_2(0) &= 0, \\ y_3' &= 2\omega y_4, & y_3(0) &= 0, \\ y_4' &= \omega(y_5 - y_3), & y_4(0) &= 0, \\ y_5' &= -2\omega y_4, & y_5(0) &= 1, \\ y_6' &= 2\omega(y_8 - y_7), & y_6(0) &= 0, \\ y_7' &= 2\omega(y_9 + y_6), & y_7(0) &= 0, \\ y_8' &= \omega(y_{10} - y_6 - 2y_9), & y_8(0) &= 0, \\ y_9' &= \omega(y_{11} - y_7 + 2y_8), & y_9(0) &= 0, \\ y_{10}' &= -2\omega(y_8 + y_{11}), & y_{10}(0) &= 1, \\ y_{11}' &= -2\omega(y_9 - y_{10}), & y_{11}(0) &= 0. \end{aligned} \quad (4.21)$$

Jak je patrné z obrázku 4.7, hodnoty koeficientů  $a_0$  a  $a_2$  postupně konvergují k jejich analytickým hodnotám. V čase  $t = T = 2$  s dosahuje koeficient  $a_0$  hodnoty 1 a koeficient  $a_2$  hodnoty  $-0.5$ , což přesně odpovídá analytickému řešení. Modrá křivka reprezentuje časový vývoj koeficientu  $a_0$ , zatímco červená přerušovaná křivka znázorňuje vývoj koeficientu  $a_2$ .

Metoda	Chyba	Čas [s]	Poměr
MTSM	3.55e-13	1.7920e-02	1.0
Euler	2.42e-01	7.9817e-03	0.4
RK4	9.04e-07	4.5127e-03	0.3
ode23	3.06e-10	9.0165e-02	5.0
ode45	9.65e-12	1.8481e-02	1.0
ode78	6.81e-13	1.5261e-02	0.9
ode89	7.53e-14	1.3109e-02	0.7
ode113	1.60e-10	4.9153e-02	2.7

Tabulka 4.10: Výsledky numerického výpočtu koeficientů Fourierovy řady pro testovací příklad  $f(t) = \sin^2(\omega t)$  pomocí lineárního systému 11 diferenciálních rovnic.

### Analýza výsledků

Provedené numerické experimenty demonstrují vysokou efektivitu Taylorovy metody, která dosahuje vynikající přesnosti při minimálním výpočetním čase. Z tabulky 4.10 je patrné, že MTSM dosahuje velmi vysoké přesnosti s chybou řádu  $10^{-13}$ .

Ve srovnání s ostatními metodami vykazuje Eulerova metoda nejnižší přesnost (chyba řádu  $10^{-1}$ ), ačkoliv její výpočetní čas je poměrně nízký. Metoda RK4 představuje dobrý kompromis mezi přesností a efektivitou, s chybou řádu  $10^{-7}$  a nejnižším výpočetním časem ze všech testovaných metod.

Mezi testovanými vestavěnými řešiči vykazuje nejlepší přesnost metoda ode89 s chybou řádu  $10^{-14}$ , která předčí i Taylorovu metodu, a to při 0.7násobku jejího výpočetního času. Metoda ode78 dosahuje podobné přesnosti jako Taylorova metoda (řádu  $10^{-13}$ ) při mírně nižším výpočetním čase.

Řešiče ode23 a ode113 poskytují přesnost řádu  $10^{-10}$ , avšak za cenu vyššího výpočetního času. Zejména řešič ode23 vykazuje 5krát delší výpočetní čas než Taylorova metoda, zatímco ode113 je přibližně 2.7krát pomalejší.

Celkově výsledky ukazují, že pro tento konkrétní problém výpočtu Fourierových koeficientů nabízejí metody ode89 a ode78 nejlepší poměr přesnosti a výpočetní náročnosti, následované Taylorovou metodou.

### Alternativní nelineární formulace

V rámci výzkumu efektivity různých formulací problému byl testován také alternativní přístup s nelineárním systémem sedmi diferenciálních rovnic prvního řádu:

$$\begin{aligned}
 y_1' &= y_3, & y_1(0) &= 0, \\
 y_2' &= \frac{2}{T}y_3y_6, & y_2(0) &= 0, \\
 y_3' &= 2\omega y_4, & y_3(0) &= 0, \\
 y_4' &= \omega(y_5 - y_3), & y_4(0) &= 0, \\
 y_5' &= -2\omega y_4, & y_5(0) &= 1, \\
 y_6' &= -2\omega y_7, & y_6(0) &= 1, \\
 y_7' &= 2\omega y_6, & y_7(0) &= 0.
 \end{aligned} \tag{4.22}$$

Tento systém je charakteristický přítomností nelineárního členu v druhé rovnici, kde se vyskytuje součin proměnných  $y_3$  a  $y_6$ . Takové systémy představují dodatečnou výzvu pro numerické řešiče, protože nelinearita může vést k nestabilitě řešení a vyžaduje speciální přístup.

Pro účinné řešení tohoto nelineárního systému byla použita rozšířená verze Taylorovy metody (MTSM) s automatickým výpočtem vyšších derivací. Výsledky numerických experimentů pro nelineární formulaci systému jsou shrnuty v tabulce 4.11.

Metoda	Čas [s]	Poměr	chyba
MTSM	5.23100e-04	1.0	1.98e-07
ode23	1.81302e-02	34.7	1.02e-09
ode45	1.67600e-03	3.2	1.15e-10
ode78	1.03610e-03	2.0	1.08e-12
ode89	9.63800e-04	1.8	2.78e-12
ode113	1.46760e-03	2.8	2.32e-11

Tabulka 4.11: Výsledky numerického výpočtu koeficientů Fourierovy řady pomocí nelineárního systému 7 diferenciálních rovnic pro testovací příklad  $f(t) = \sin^2(\omega t)$ .

### Srovnání lineární a nelineární formulace

Srovnání výsledků z tabulek 4.10 a 4.11 poskytuje zajímavý pohled na efektivitu různých formulací problému. Zatímco lineární systém vyžaduje 11 diferenciálních rovnic, nelineární formulace obsahuje pouze 7 rovnic, což teoreticky může vést k nižší výpočetní náročnosti.

Přestože nelineární formulace s menším počtem rovnic nabízí potenciální výhodu v menším počtu výpočetních operací, z výsledků je patrné, že lineární systém s 11 rovnicemi umožňuje Taylorově metodě dosáhnout podstatně vyšší přesnosti (chyba řádu  $10^{-12}$  oproti  $10^{-7}$  u nelineárního systému). Rozdíl v přesnosti naznačuje, že lineární formulace je pro Taylorovu metodu numericky stabilnější, přestože vyžaduje více rovnic.

Je však zajímavé, že v případě nelineárního systému dosahují vestavěné metody s adaptivním krokem (zejména ode78 a ode89) vynikající přesnosti, srovnatelné nebo dokonce lepší než Taylorova metoda u lineárního systému, a to při výrazně nižších výpočetních časech.

Zejména metoda ode78, která u lineárního systému vykazovala neúměrně vysoký výpočetní čas (394,7krát vyšší než MTSM), je u nelineárního systému pouze 2krát pomalejší než Taylorova metoda a přitom dosahuje přesnosti řádu  $10^{-12}$ .

Toto pozorování naznačuje, že volba vhodné formulace problému a odpovídající numerické metody může významně ovlivnit efektivitu výpočtu. Pro Taylorovu metodu je v tomto případě výhodnější lineární formulace, zatímco vestavěné adaptivní řešiče si překvapivě lépe poradí s nelineární formulací s menším počtem rovnic.

#### 4.7.5 Lorenzův systém

Prezentovaný příklad demonstruje chování nelineárního systému – Lorenzova systému, který jako první popsal Edward Lorenz v [14]. Tento matematický model objasňuje nepředvídatelnou dynamiku atmosférických jevů. Dle Lorenzovy teorie lze atmosféru planety modelovat jako dvourozměrnou tekutinovou buňku s vertikálním teplotním gradientem – ohřivanou zespodu a ochlazovanou shora, jak uvádí [11]. Dynamika této tekutiny je charakterizována následujícím trojrozměrným systémem obyčejných diferenciálních rovnic (4.23)

$$\begin{aligned} x' &= \sigma(y - x) & x(0) &= 1 \\ y' &= \rho x - y - xz & y(0) &= 1 \\ z' &= xy - \beta z & z(0) &= 1, \end{aligned} \quad (4.23)$$

kde  $\sigma$  je Prandtlovo číslo,  $\rho$  je Rayleighovo číslo a  $\beta$  je parametr související s fyzikální velikostí systému. Chování systému závisí na hodnotách parametrů a počátečních podmínkách. Typickou vlastností tohoto modelu je vysoká citlivost na počátečních podmínkách. Nepatrné změny počátečního stavu mohou vést k odlišným trajektoriím, což vysvětluje zásadní omezení dlouhodobé předpovědi počasí. Soustavu rovnic (4.23) lze přepsat do maticovo-vektorové reprezentace

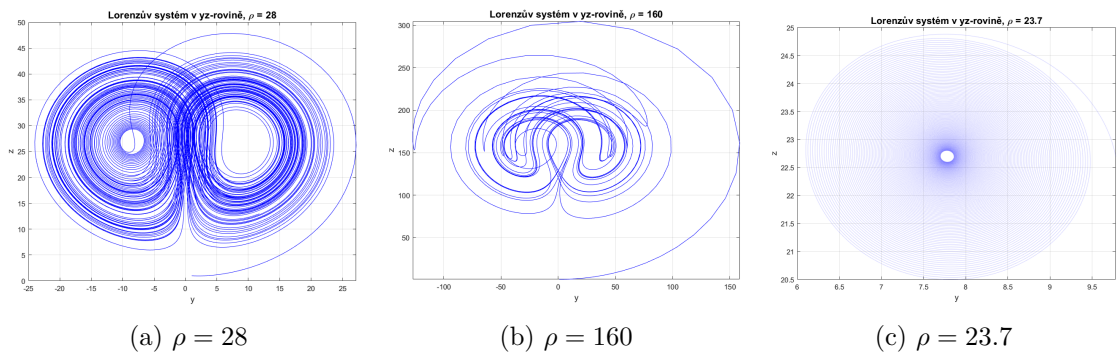
$$\mathbf{y} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \mathbf{A} = \begin{pmatrix} -\sigma & \sigma & 0 \\ \rho & -1 & 0 \\ 0 & 0 & -\beta \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \mathbf{B}_1 = \begin{pmatrix} 0 & 0 \\ -1 & 0 \\ 0 & 1 \end{pmatrix}, \mathbf{y}_{ij} = \begin{pmatrix} 1 & 3 \\ 1 & 2 \end{pmatrix}. \quad (4.24)$$

Pro experimenty byly pevně stanoveny parametry  $\sigma = 10$  a  $\beta = 8/3$ . Měněn byl pouze parametr  $\rho$ , aby bylo dosaženo různého chování systému (4.23). Pro  $\rho = 28$ , hodnotu původně použitou Lorenzem [14], je pozorováno chaotické chování. Pro vysoké hodnoty  $\rho$ , např.  $\rho = 160$ , je řešení periodické [12]. Pro  $\rho = 23.7$  je řešení stabilní. Dva rovnovážné body lze vypočítat pomocí (4.25). Počáteční podmínky byly poté vypočteny přidáním konstantního vektoru  $\mathbf{v} = (0, 2, 0)$  k rovnovážnému bodu  $Q^+$  [11, 8].

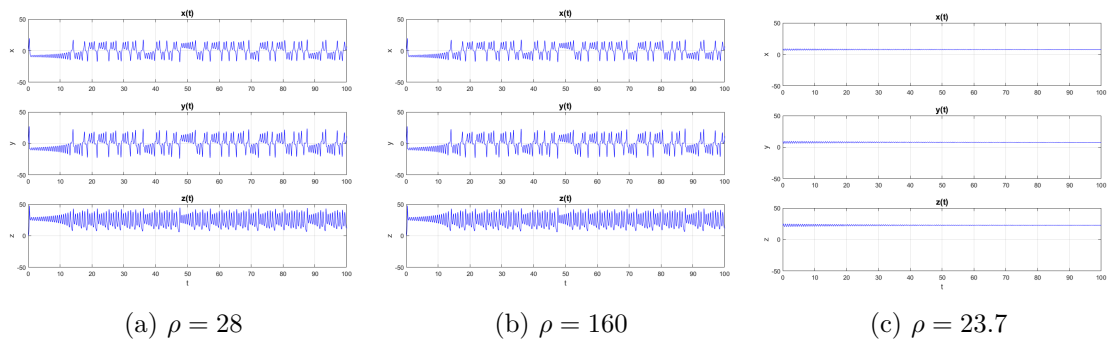
$$Q^\pm = (\pm\sqrt{\beta(\rho - 1)}, \pm\sqrt{\beta(\rho - 1)}, \rho - 1). \quad (4.25)$$

Pro experimenty s Lorenzovým systémem byly tolerance pro všechny numerické řešiče nastaveny na  $1 \times 10^{-10}$ . Maximální simulační čas byl nastaven na  $t_{max} = 100s$  pro všechny experimenty. Obrázek 4.8 ukazuje řešení Lorenzova systému pro různé hodnoty parametru  $\rho$  v rovině yz.

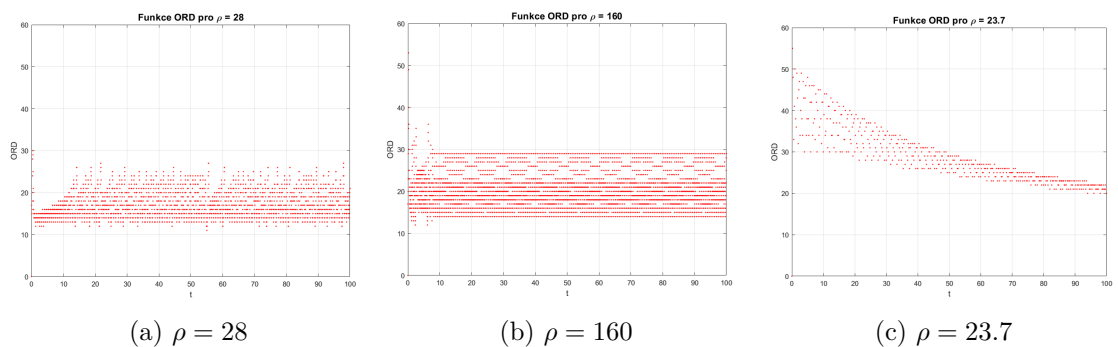
Řešení v časové oblasti je na obrázku 4.9 a graf funkce ORD je na obrázku 4.10.



Obrázek 4.8: Chování Lorenzova systému v rovině yz.



Obrázek 4.9: Chování Lorenzova systému v časové oblasti.



Obrázek 4.10: Grafy funkce ORD pro různé hodnoty  $\rho$ .

## Implementace v MATLABu

Na rozdíl od lineárních problémů, hodnoty ORD pro nelineární problémy už nejsou téměř konstantní, ale mohou se během výpočtu velmi rychle měnit. Výsledky simulačních experimentů jsou v následujících tabulkách.

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	2000	0.140374	1
ode23	84514	2.69183	19.176
ode45	12327	0.205874	1.4666
ode78	3621	0.198075	1.4111
ode89	3233	0.154257	1.0989
ode113	1514	0.231591	1.6498

Tabulka 4.12: Výsledky simulace pro Lorenzův systém,  $\rho = 28$

Pro chaotický režim Lorenzova systému ( $\rho = 28$ ) vykazuje Taylorova metoda dobrou výkonnost s výpočetním časem 0,14 s. Metody ode45, ode78 a ode89 dosahují srovnatelné výkonnosti s poměry časů v rozmezí 1,1-1,5 vůči MTSM. Metoda ode23 je výrazně nejpočetnější s téměř 20násobným časem výpočtu, což odráží její nižší řád přesnosti.

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	4000	0.353951	1
ode23	21621	6.6503	18.789
ode45	31064	0.483237	1.3653
ode78	7609	0.292905	0.8275
ode89	6820	0.253664	0.7166
ode113	3701	0.373851	1.0562

Tabulka 4.13: Výsledky simulace pro Lorenzův systém,  $\rho = 160$

Pro periodický režim ( $\rho = 160$ ) je situace odlišná. Metody ode78 a ode89 nyní překonávají Taylorovu metodu s poměry 0,83 a 0,72 respektive. Tento režim je celkově výpočetně náročnější, což se projevuje vyšším počtem kroků u všech metod. Zejména metoda ode45 vykazuje výrazný nárůst počtu kroků (z 12 327 na 31 064), což naznačuje, že tento režim představuje pro adaptivní metody s nižším řádem větší výzvu.

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	500	0.0539224	1
ode23	23946	0.78971	14.645
ode45	5888	0.077429	1.4359
ode78	1828	0.055291	1.0254
ode89	1665	0.060707	1.1258
ode113	805	0.095550	1.7712

Tabulka 4.14: Výsledky simulace pro Lorenzův systém,  $\rho = 23.7$

Stabilní režim ( $\rho = 23.7$ ) je výpočetně nejméně náročný pro všechny metody, s Taylorovou metodou vyžadující pouze 500 kroků. Metoda ode78 dosahuje téměř identické výkonnosti jako MTSM s poměrem 1,025, zatímco metoda ode89 je jen mírně pomalejší. Celkově výsledky v MATLABu ukazují, že MTSM je vysoce konkurenceschopná a pro některé režimy dynamiky poskytuje nejlepší výkonnost, zatímco pro jiné jsou mírně efektivnější metody vyšších řádů jako ode78 a ode89.

### Implementace v Pythonu

Následná část ukazuje výsledky pro implementaci v prostředí Python. Jsou zde výsledky jak pro neoptimalizovanou verzi bez JIT kompilace, tak pro optimalizovanou verzi s JIT kompilací.

#### Neoptimalizovaná implementace

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	2000	1.16901	1
RK23	613377	18.7053	16.00100
RK45	25001	1.15155	0.98507
DOP853	4326	0.36251	0.31010
LSODA	19265	0.18347	0.15695

Tabulka 4.15: Výsledky simulace Lorenzova systému pro  $\rho = 28$  (neoptimalizovaná verze).

V neoptimalizované verzi Pythonu je pro chaotický režim ( $\rho = 28$ ) Taylorova metoda výrazně pomalejší než v MATLABu, což odráží rozdíl mezi interpretovaným jazykem a optimalizovaným prostředím MATLABu. Metoda RK45 dosahuje srovnatelné výkonnosti s MTSM, zatímco metody DOP853 a LSODA ji výrazně překonávají, s LSODA dosahující více než šestinásobného zrychlení.

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	4000	3.16889	1
RK23	1577306	47.54590	15.00400
RK45	61465	2.77902	0.87697
DOP853	10701	1.01816	0.32130
LSODA	54311	0.50989	0.16091

Tabulka 4.16: Výsledky simulace Lorenzova systému pro  $\rho = 160$  (neoptimalizovaná verze).

Pro periodický režim ( $\rho = 160$ ) v neoptimalizovaném Pythonu jsou rozdíly ještě výraznější. Taylorova metoda vyžaduje devítinásobek času potřebného v MATLABu. Metoda LSODA opět dosahuje nejlepší výkonnosti, přibližně šestkrát rychlejší než MTSM. Patrný je také výrazný nárůst počtu kroků u metody RK23, která potřebuje více než 1,5 milionu kroků pro dokončení simulace.

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	500	0.71344	1
RK23	183740	5.51903	7.73576
RK45	11989	0.53383	0.74824
DOP853	1948	0.16230	0.22749
LSODA	8943	0.08072	0.11314

Tabulka 4.17: Výsledky simulace Lorenzova systému pro  $\rho = 23.7$  (neoptimalizovaná verze).

Ve stabilním režimu ( $\rho = 23.7$ ) je v neoptimalizovaném Pythonu rozdíl mezi Taylorovou metodou a MATLABem ještě výraznější – výpočet v Pythonu je přibližně třináctkrát pomalejší. Metoda LSODA je opět nejrychlejší, téměř devětkrát rychlejší než MTSM. Tyto výsledky naznačují, že v neoptimalizovaném Pythonu je Taylorova metoda výrazně znevýhodněna oproti ostatním metodám implementovaným v SciPy.

### Optimalizovaná implementace s JIT kompilací

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	2000	0.01351	1
RK23	613377	18.37470	1360.25000
RK45	25001	1.08287	80.16310
DOP853	4326	0.35471	26.25890
LSODA	19265	0.17607	13.03430

Tabulka 4.18: Výsledky simulace Lorenzova systému pro  $\rho = 28$  (optimalizovaná verze).

Situace se dramaticky mění v optimalizované verzi s JIT kompilací pro chaotický režim ( $\rho = 28$ ). Taylorova metoda nyní dosahuje času pouhých 0,0135 s, což představuje zrychlení přibližně  $86,5\times$  oproti neoptimalizované verzi a  $10,4\times$  oproti MATLABu. Všechny ostatní metody jsou nyní výrazně pomalejší, s LSODA (nejrychlejší z nich) vyžadující více než  $13\times$  delší čas než MTSM. Tento obrat ukazuje mimořádný vliv JIT kompilace na výkonnost Taylorovy metody.

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	4000	0.03084	1
RK23	1577306	46.82000	1518.21000
RK45	61465	2.70879	87.83660
DOP853	10701	1.04442	33.86700
LSODA	54311	0.53739	17.42580

Tabulka 4.19: Výsledky simulace Lorenzova systému pro  $\rho = 160$  (optimalizovaná verze).

Pro periodický režim ( $\rho = 160$ ) je zrychlení Taylorovy metody ještě výraznější – dosažené zlepšení činí přibližně 103 násobků původní rychlosti. Oproti MATLABové implementaci je to zrychlení přibližně  $11,5\times$ . Relativní výkonnost ostatních metod se dále zhoršuje, s RK23 vyžadující více než  $1500\times$  delší výpočetní čas než MTSM.

Metoda	Počet kroků	Čas [s]	Poměr
MTSM	500	0.00534	1
RK23	183740	5.50479	1030.93000
RK45	11989	0.53963	101.06100
DOP853	1948	0.16074	30.10300
LSODA	8943	0.08080	15.13310

Tabulka 4.20: Výsledky simulace Lorenzova systému pro  $\rho = 23.7$  (optimalizovaná verze).

Ve stabilním režimu ( $\rho = 23.7$ ) vykazuje optimalizovaná Taylorova metoda vůbec nejkratší dobu výpočtu – zrychlení činí zhruba 134 násobků oproti neoptimalizované verzi a více než desetinásobek oproti implementaci v MATLABu. I zde jsou všechny ostatní metody výrazně pomalejší, což potvrzuje mimořádnou efektivitu JIT kompilace pro Taylorovu metodu napříč různými dynamickými režimy.

### Srovnání implementací v MATLABu a Pythonu

Pro přehlednější srovnání výkonnosti různých implementací je níže uvedena tabulka s výpočetními časy pro všechny tři režimy Lorenzova systému:

Implementace	Výpočetní čas [s]		
	$\rho = 28$	$\rho = 160$	$\rho = 23.7$
MATLAB	0.14037	0.35395	0.05392
Python (neoptim.)	1.16901	3.16889	0.71344
Python (JIT)	0.01351	0.03084	0.00534
Poměr Python (neoptim.)/MATLAB	8.33	8.95	13.23
Poměr Python MATLAB/Python (JIT)	10.39	11.48	10.10
Poměr Python (neoptim.)/Python (JIT)	86.53	102.75	133.60

Tabulka 4.21: Srovnání výpočetních časů Taylorovy metody v různých implementacích pro Lorenzův systém.

Toto srovnání názorně demonstruje dramatické rozdíly mezi implementacemi:

### Závěry pro praktické použití

Na základě provedených experimentů lze formulovat následující závěry:

- Taylorova metoda s JIT kompilací v Pythonu představuje nejefektivnější řešení pro simulaci Lorenzova systému ze všech testovaných přístupů.
- Metoda RK23 (ode23) je konzistentně nejméně efektivní nezávisle na implementačním prostředí.
- JIT kompilace dramaticky mění výkonnostní charakteristiky numerických metod v interpretovaných jazycích, umožňující dosáhnout výrazně lepších výsledků než specializované nástroje jako MATLAB.

- Vliv dynamického režimu na výkonnost metod je významný – periodický režim ( $\rho = 160$ ) představuje nejnáročnější výzvu pro většinu metod, zatímco stabilní režim ( $\rho = 23,7$ ) je výpočetně nejméně náročný.
- Pro reálné aplikace je optimalizovaná Taylorova metoda vysoce konkurenceschopná a v mnohých případech překonává tradiční přístupy k řešení nelineárních dynamických systémů.

Tyto výsledky zdůrazňují význam implementačních detailů a optimalizačních technik při numerickém řešení diferenciálních rovnic, přičemž ukazují, že moderní přístupy mohou výrazně zlepšit efektivitu i klasických numerických metod jako je Taylorova metoda.

# Kapitola 5

## Závěr

V rámci této práce byla představena a experimentálně ověřena numerická metoda vyššího řádu pro řešení soustav obyčejných diferenciálních rovnic, založená na využití Taylorovy řady. Je třeba zdůraznit, že část práce věnovaná implementaci a testování Taylorovy metody v Pythonu přesahovala původní zadání. Hlavní přínos navržené Taylorovy metody spočívá ve schopnosti dosáhnout vysoké přesnosti výpočtu zahrnutím vyšších derivací do lokálního řešení. Díky tomu lze získat velmi přesné výsledky s potenciálně menším počtem kroků integrace oproti tradičním přístupům. Provedené experimenty tuto výhodu potvrdily. Taylorova metoda poskytla vysoce přesná řešení a ukázala se být efektivní i z hlediska výpočetního času.

Ve srovnání s klasickými metodami, jako je explicitní Eulerova metoda či Runge–Kuttova metoda 4. řádu, dosahovala Taylorova metoda výrazně menších numerických chyb při obdobné nebo nižší časové náročnosti. Zatímco Eulerova metoda je sice velmi rychlá, avšak vykazuje velkou chybovost, a naopak klasický RK4 zajišťuje vysokou přesnost za cenu delšího výpočtu, navržený postup nabídl výhodnější kompromis mezi přesností a rychlostí. Adaptivní metoda `ode45` z balíku MATLAB dokázala v některých testech zkrátit dobu simulace oproti Taylorově metodě, avšak za cenu citelně nižší přesnosti výsledků. Po aplikaci pokročilých optimalizačních technik (zejména JIT kompilace v Pythonu) se navíc ukázalo, že Taylorova metoda může v praxi překonat i specializované algoritmy – v případě simulace Lorenzova systému představovala optimalizovaná Taylorova metoda nejrychlejší řešení ze všech testovaných přístupů, překonávající dokonce i pokročilé implementace (např. řešič LSODA) z hlediska rychlosti výpočtu, a to při zachování vysoké přesnosti.

Navržený přístup nicméně není bez omezení. Dosažení vyšších řádů přesnosti vyžaduje výpočet odpovídajících derivací pravé strany řešené ODE, což je nutno zajistit například symbolickou manipulací výrazů nebo použitím automatické diferenciace. Tato nutnost činí implementaci podstatně složitější oproti standardním metodám, které vystačí pouze s opakovaným výpočtem funkce  $f(t, y)$ . Výpočet velkého množství derivací navíc představuje dodatečnou zátěž – bez optimalizačních technik (jako je vektorizace výpočtů či JIT kompilace) může být Taylorova metoda výrazně pomalejší než optimalizované knihovní funkce pro řešení obyčejných diferenciálních rovnic. Dalším omezením současné realizace je použití pevného kroku integrace bez adaptivní kontroly chyby. To může způsobit potíže v situacích, kdy řešení vykazuje prudké změny nebo je systém tužší; v takových případech by explicitní Taylorova metoda vyžadovala velmi malý krok pro zachování stability a přesnosti, čímž ztrácí efektivitu (podobně jako jiné explicitní metody) nebo může zcela selhat. Obdobně v přítomnosti nespojitostí či singularit v průběhu řešení nelze vysoký řád Taylorovy metody plně využít, neboť v takových bodech lokální Taylorův rozvoj přestává být platný.

Pro zvýšení robustnosti a další zlepšení efektivity Taylorovy metody se nabízejí určitá rozšíření. Prvním z nich je implementace adaptivního řízení krokové velikosti. Zavedení adaptivního kroku integrace by umožnilo automaticky měnit délku časového kroku na základě odhadu lokální chyby (podobně jako to dělá např. `ode45`), a udržet tak požadovanou přesnost při co nejnižších výpočetních nákladech i v částech simulace s komplikovanější dynamikou. Další možností je využití paralelního zpracování. Výpočet jednotlivých členů Taylorova rozvoje (resp. potřebných derivací) by bylo možné do určité míry paralelizovat a rozdělit mezi více výpočetních jednotek (vícejádrový procesor či GPU). Tím by se zejména u rozsáhlejších soustav ODE dal výrazně zkrátit celkový čas simulace.

Za velmi slibný směr lze považovat také rozšíření navrženého přístupu na parciální diferenciální rovnice (PDE). Jednou z možností je aplikovat Taylorovu metodu jako časový integrátor v rámci metody čar, kdy se nejprve prostorové derivace v PDE aproximují diskrétně (např. konečnými diferencemi nebo elementy) a získaná soustava ODE se poté řeší Taylorovou metodou. Tím by se výhody Taylorova integrátoru – zejména vysoká přesnost při větších krocích – mohly uplatnit i při řešení evolučních PDE. Úspěšné nasazení Taylorovy metody na PDE by významně rozšířilo okruh úloh, pro něž je tento přístup vhodný, a mohlo by inspirovat další výzkum v oblasti vysoce přesných numerických schémat. Celkově dosažené výsledky potvrzují, že vhodně implementovaná Taylorova metoda představuje perspektivní a plně konkurenční alternativu ke stávajícím numerickým řešičům diferenciálních rovnic, čímž naplňuje cíle stanovené v úvodu práce. Díky svému vysokému řádu a možnosti pokročilých optimalizací dokáže nabídnout vynikající kombinaci přesnosti a rychlosti výpočtu. Zmíněné potenciální vylepšení a rozšíření pak mohou její uplatnění dále výrazně rozšířit i na složitější a náročnější úlohy.

# Literatura

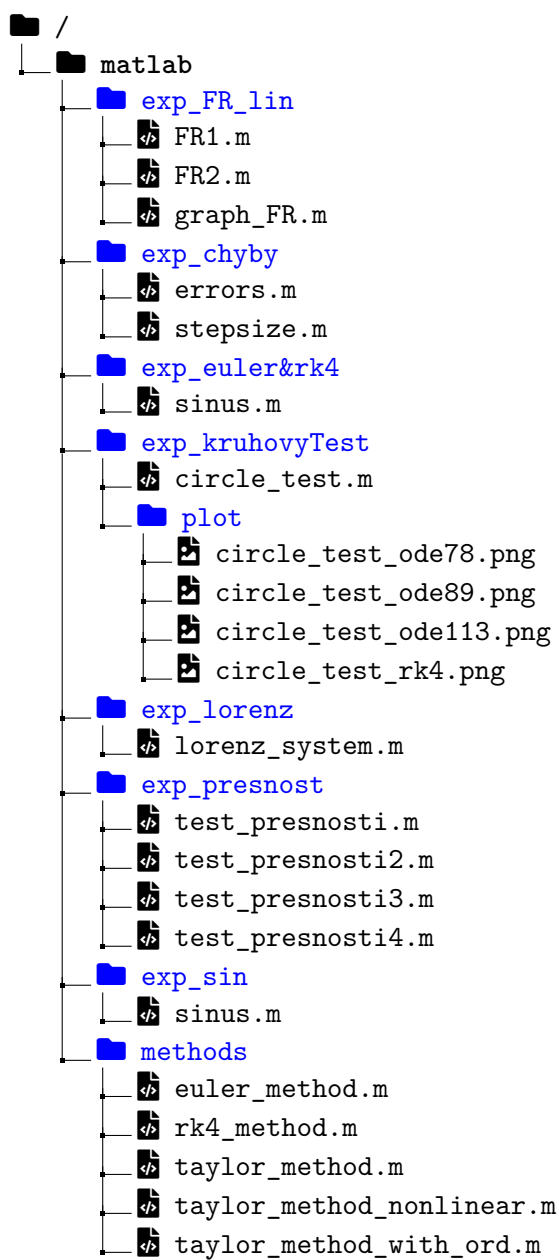
- [1] BARRIO, R. Performance of the Taylor series method for ODEs/DAEs. *Applied Mathematics and Computation*, 2005, sv. 163, č. 2, s. 525–545. ISSN 0096-3003. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0096300304002401>.
- [2] BOGACKI, P. a SHAMPINE, L. F. A 3(2) pair of Runge-Kutta formulas. *Applied Mathematics Letters*, 1989, sv. 2, č. 4, s. 321–325.
- [3] BRUGNANO, L.; GURIOLI, G. a SUN, Y. Energy-conserving Hamiltonian Boundary Value Methods for the numerical solution of the Korteweg-de Vries equation. *Journal of Computational and Applied Mathematics*, 2018, sv. 351, s. 117–135. Dostupné z: <https://doi.org/10.1016/j.cam.2018.10.014>.
- [4] BUTCHER, J. C. *Numerical Methods for Ordinary Differential Equations*. 2nd. Chichester: John Wiley & Sons, 2008. ISBN 978-0-470-72335-7.
- [5] CHANG, Y. a CORLISS, G. ATOMFT: solving ODEs and DAEs using Taylor series. *Computers & Mathematics with Applications*, 1994, sv. 28, č. 10, s. 209–233. ISSN 0898-1221. Dostupné z: <https://www.sciencedirect.com/science/article/pii/0898122194001936>.
- [6] CODDINGTON, E. A. a LEVINSON, N. *Theory of Ordinary Differential Equations*. New York: McGraw–Hill, 1955. ISBN 978-0-07-011542-2.
- [7] DORMAND, J. R. a PRINCE, P. J. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 1980, sv. 6, č. 1, s. 19–26.
- [8] HATELEY, J. *The Lorenz System* online. 2014. Dostupné z: <http://web.math.ucsb.edu/~jhateley/paper/lorenz.pdf>.
- [9] HIGHAM, N. *Note on High-order ODE Solvers in MATLAB* online. 2023. Dostupné z: <https://nhigham.com/>. [cit. 2025-03-08].
- [10] HINDMARSH, A. C. ODEPACK, A Systematized Collection of ODE Solvers. In: STEPLEMAN, R. S.; CARVER, M.; PESKIN, R.; GAGLIARDI, F. a HEMENWAY, D., ed. *Scientific Computing*. Amsterdam: North-Holland, 1983, sv. 1, s. 55–64. IMACS Transactions on Scientific Computation.
- [11] HIRSCH, M. W.; SMALE, S. a DEVANEY, R. L. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. 2nd. San Diego: Academic Press, 2004. ISBN 0-12-349703-5.

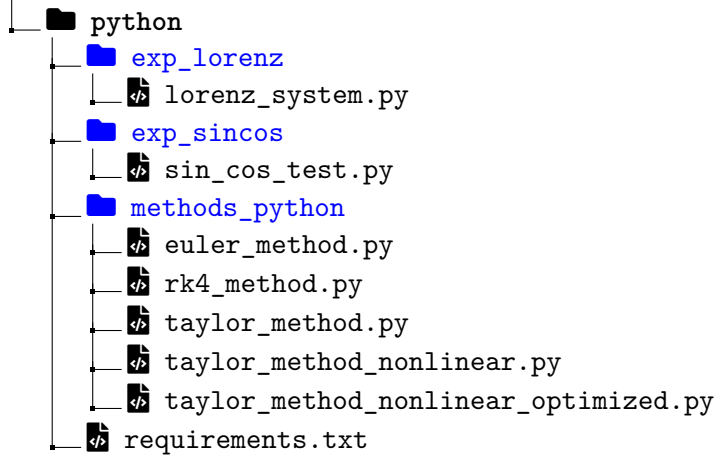
- [12] KNILL, O. *The Lorenz System* online. 2005. Dostupné z: [http://www.math.harvard.edu/archive/118r\\_spring\\_05/handouts/lorenz.pdf](http://www.math.harvard.edu/archive/118r_spring_05/handouts/lorenz.pdf). [cit. 2025-03-10].
- [13] LEBL, J. *Notes on Diffy Qs: Differential Equations for Engineers*. 5th. Independently Published, 2019. ISBN 978-1-70623-023-6. Dostupné z: <https://www.jirka.org/diffyqs/>.
- [14] LORENZ, E. N. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*. Boston MA, USA: American Meteorological Society, 1963, sv. 20, č. 2, s. 130–141. Dostupné z: [https://journals.ametsoc.org/view/journals/atsc/20/2/1520-0469\\_1963\\_020\\_0130\\_dnf\\_2\\_0\\_co\\_2.xml](https://journals.ametsoc.org/view/journals/atsc/20/2/1520-0469_1963_020_0130_dnf_2_0_co_2.xml).
- [15] NEČASOVÁ, G.; VEIGEND, P. a ŠÁTEK, V. Modern Taylor Series Method in Numerical Integration: Part 2. In: VŠB – Technical University of Ostrava. *17th Czech–Polish Conference Modern Mathematical Methods in Engineering (3mi)*. Horní Lomná: [b.n.], 2018, s. 211–220. ISBN 978-80-248-4135-9.
- [16] RAISSI, M.; PERDIKARIS, P. a KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 2019, sv. 378, s. 686–707. Dostupné z: <https://arxiv.org/abs/1711.10561>.
- [17] SANZ SERNA, J. M. Symplectic Runge–Kutta schemes for adjoint equations, automatic differentiation, optimal control, and more. *SIAM Review*, 2016, sv. 58, č. 1, s. 3–33. Dostupné z: <https://arxiv.org/abs/1503.04021>.
- [18] STROGATZ, S. H. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. 2nd. Boulder: Westview Press, 2015. ISBN 978-0-8133-4910-7.
- [19] THE MATHWORKS, INC.. *MATLAB ode113 Solver – R2023b Documentation*. 2023. Dostupné z: <https://www.mathworks.com/help/matlab/ref/ode113.html>.
- [20] THE MATHWORKS, INC.. *MATLAB ode15s Solver – R2023b Documentation*. 2023. Dostupné z: <https://www.mathworks.com/help/matlab/ref/ode15s.html>.
- [21] THE MATHWORKS, INC.. *MATLAB ode23 Solver – R2023b Documentation*. 2023. Dostupné z: <https://www.mathworks.com/help/matlab/ref/ode23.html>.
- [22] THE MATHWORKS, INC.. *MATLAB ode45 Solver – R2023b Documentation*. 2023. Dostupné z: <https://www.mathworks.com/help/matlab/ref/ode45.html>.
- [23] THE MATHWORKS, INC.. *MATLAB ode78 Solver – R2023b Documentation*. 2023. Dostupné z: <https://www.mathworks.com/help/matlab/ref/ode78.html>.
- [24] THE MATHWORKS, INC.. *MATLAB ode89 Solver – R2023b Documentation*. 2023. Dostupné z: <https://www.mathworks.com/help/matlab/ref/ode89.html>.
- [25] TREFETHEN, L. N.; BIRKISSON, Á. a DRISCOLL, T. A. Exploring ODEs. *SIAM Review*, 2017, sv. 59, č. 3, s. 469–491. Dostupné z: <https://people.maths.ox.ac.uk/trefethen/ExplODE/>.

- [26] VEIGEND, P. *High Order Numerical Method in Modelling and Control Systems*. Brno, 2023. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor: Ing. Václav Šátek, Ph.D.
- [27] WIKIPEDIE. *Součínové pravidlo* online. 2023. Dostupné z: [https://cs.wikipedia.org/wiki/Sou%C4%8Dinov%C3%A9\\_pravidlo](https://cs.wikipedia.org/wiki/Sou%C4%8Dinov%C3%A9_pravidlo). [cit. 2025-04-20].

# Příloha A

## Obsah paměťového média





# Příloha B

## Manuál

Tato část stručně popisuje způsob spuštění kódu a nezbytné požadavky pro jeho běh.

### B.1 Požadavky

#### Python

- Verze Pythonu: 3.8 nebo novější
- Závislosti: `numpy`, `scipy`, `matplotlib`, `tabulate`, `numba`
- Instalace pomocí:

```
pip install -r requirements.txt
```

#### MATLAB

- Není potřeba žádný speciální toolbox
- Funkce musí být přístupné (např. ve složce `methods/`), přidat na začátku skriptu:

```
addpath(' ../methods/');
```

### B.2 Spuštění

#### Python

- Spuštění testu přesnosti:

```
python3 sin_cos_test.py
```

- Spuštění výpočtu Lorenzova systému:

```
python3 lorenz_system.py -v optimized
```

## MATLAB

- Spuštění libovolného skriptu (např.):

```
>> test_presnosti.m
```

- Skripty volají metody jako například:

```
[t, y] = taylor_method(A, b, y0, h, t_final, EPS);
```

## B.3 Poznámka

Ukázkové skripty včetně `test_presnosti.m`, `circle_test.m` nebo `sinus.m` slouží jako příklady; reálných experimentů je více a lze je libovolně upravovat nebo rozšiřovat.

# Příloha C

## Použité nástroje

- **Python 3.x** – Interpretační jazyk pro vývoj a testování numerických algoritmů založených na Taylorově řadě.
- **MATLAB R2024b** – Referenční prostředí pro maticově-vektorové výpočty a pro porovnání výkonu implementace s vestavěnými řešiči.
- **NumPy** – Základní knihovna pro práci s vícerozměrnými poli; použita k efektivním aritmetickým operacím a přípravě dat.
- **SciPy (solve\_ivp)** – Umožnila snadnou integraci referenčních metod pro srovnání s vlastní Taylorovou metodou.
- **Numba** – Just-In-Time kompilátor výrazně zrychlující kritické smyčky při opakovaném vyhodnocování členů Taylorovy řady.
- **Matplotlib** – Generování grafů výsledků (časové průběhy, kruhové testy, ORD křivky) přímo z Pythonu.
- **tabulate** – Přehledná textová prezentace naměřených tabulkových výsledků v příkazové řádce i ve výstupním souboru.
- **LaTeX** – Sazba celé bakalářské práce.
- **Git & GitHub** – Správa verzí zdrojových kódů (.py, .m).
- **Visual Studio Code** – Hlavní vývojové IDE s rozšířeními Python, MATLAB.
- **ChatGPT** – Konzultace textové struktury, bib<sub>T</sub>E<sub>X</sub> citací a rychlá zpětná vazba k textu či kódu.