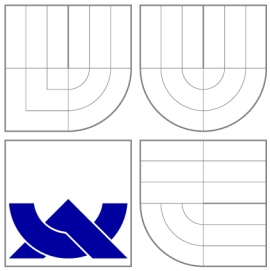
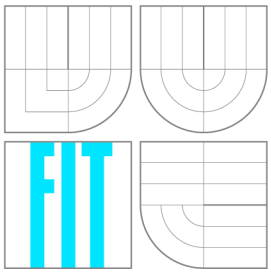


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VÝVOJOVÉ PROSTŘEDÍ NUMERICKÝCH INTEGRÁTORŮ

NUMERICAL INTEGRATORS DEVELOPMENT ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. VÁCLAV VOPĚNKA
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. JIŘÍ KUNOVSKÝ, CSc.
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce se zabývá transformací soustav diferenciálních rovnic do polynomiálního tvaru. Takto transformované soustavy diferenciálních rovnic je poté možno řešit pomocí Taylorova rozvoje. Tato metoda umožňuje počítat numerické řešení počáteční úlohy dynamickou volbou řádu tak, aby byla splněna požadovaná přesnost. Práce matematicky dokazuje, že transformované soustavy diferenciálních rovnic mají stejné řešení, jako soustavy původních rovnic. Tato transformace je využitelná pro všechny matematické funkce běžně používané v technických aplikacích. Práce se dále zabývá optimalizací dané problematiky a implementuje ji v přiloženém programu taylor. Program umožňuje matematické a grafické zpracování řešení zadaných diferenciálních rovnic podle zvolených parametrů.

Abstract

This term project describes transformation of system of diferential equations into polynomial form. Such transformed systems of diferential equations can be subsequently solved using Taylor series. This method enables computing of initial problem's numeric solution using dynamical order selection in order to achieve required accuracy. The work mathematically proves, that transformed systems of diferential equations have the same solution as the original systems. This transformation can be used for all mathematic functions commonly used in technical applications. The work also focuses on optimization of given problem and implements it in programme taylor. This progamme enables user to solve given diferential equations with chosen parameters.

Klíčová slova

Numerické metody, diferenciální rovnice, Taylorova řada, multidimenzionální Pascalovy trojúhelníky

Keywords

Numeric methods, differencial equations, Taylor series, Multidimensional Pascal's triangles

Citace

Václav Vopěnka: Vývojové prostředí numerických integrátorů, diplomová práce, Brno, FIT VUT v Brně, 2011

Vývojové prostředí numerických integrátorů

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana doc. Ing., Jiřího Kunovského, CSc.

.....
Václav Vopěnka
23. května 2011

Poděkování

Chtěl bych poděkovat panu doc. Ing., Jiřímu Kunovskému za čas, který mi věnoval při konzultacích k tomuto tématu. Dále bych chtěl poděkovat panu Mgr. Zdeněku Opluštilovi, Ph.D. a panu RNDr. Karelu Mikuláškovi, Ph.D. za konzultace k matematické části mé práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Numerické řešení diferenciálních rovnic je jedna z nejčastějších aplikací numerických metod. Využívá se při studiu nebo simulaci fyzikálních vlastností objektů či prostředí, která jsou popsána soustavami diferenciálních rovnic (modely pro předpověď počasí, simulace aerodynamických vlastností letadel, popis elektronických obvodů atd.). Tato práce se zabývá numerickým řešením počáteční úlohy soustavy diferenciálních rovnic prvního řádu pomocí Taylorova rozvoje a matematickou transformací soustavy rovnic do tvaru vhodného pro řešení touto metodou.

Za počáteční úlohu považujeme soustavu diferenciálních rovnic prvního řádu ve tvaru:

$$\begin{aligned} y_1' &= \phi_1(y_1, \cdots, y_n) & y_1(t_0) &= a_{1,0} \\ y_2' &= \phi_2(y_1, \cdots, y_n) & y_2(t_0) &= a_{2,0} \\ &\vdots & & \\ y_n' &= \phi_n(y_1, \cdots, y_n) & y_n(t_0) &= a_{n,0} \end{aligned} \tag{1.1}$$

Numerickým řešením této soustavy rozumíme matici:

$$\begin{bmatrix} a_{1,0} & a_{1,1} & \cdots & a_{1,m} \\ a_{2,0} & a_{2,1} & \cdots & a_{2,m} \\ & & \vdots & \\ a_{n,0} & a_{n,1} & \cdots & a_{n,m} \end{bmatrix} \tag{1.2}$$

kde $a_{i,j} = y_i(t_j)$.

Pro získání těchto hodnot se často používá Runge-Kuttova metoda. Tato metoda je ovšem pro některé typy úloh nevhodná (např. stiff systémy). V takových případech je vhodnější metoda Taylorova rozvoje. Tato metoda také umožňuje dynamickou volbu řádu tak, aby byla splněna požadovaná přesnost.

Práce je rozdělena do sedmi hlavních částí. V kapitole 2 je nejprve uveden stručný přehled současného způsobu numerického řešení počáteční úlohy, následně je zde popsán programem TKSL a stručně je vysvětleno, jaké existují způsoby řešení počátečních úloh v polynomiálním tvaru. V kapitole 3 uvedena transformace soustavy diferenciálních rovnic na soustavu v elementárním tvaru pro řešení pomocí Taylerova rozvoje, a matematický důkaz správnosti této transformace a ekvivalence původní soustavy rovnic se soustavou transformovanou. V kapitole 4 je uvedena souhrnná tabulka transformací jednotlivých funkcí na ekvivalentní soustavu diferenciálních rovnic. V kapitole 5 je rozvedena teorie popisující aproximaci funkce pomocí Taylorova polynomu a její využití pro řešení počátečních úloh.

V kapitole 6 se zabýváme možnou optimalizací získávání koeficientů Taylorovy řady. Je zde ukázáno, že získávání vyšších derivací polynomiálních funkcí vede na výpočty pomocí upravených multidimenzionálních Pascalových trojúhelníků. V kapitole 7 rozebíráme implementaci navržených algoritmů a v kapitole 8 implementované algoritmy porovnáváme.

Kapitola 2

Numerické řešení počáteční úlohy

2.1 Metody Runge-Kutta

Soustavy diferenciálních rovnic se ve většině matematické literatury řeší pomocí jedné z metod Runge-Kutta. Tato metoda (popsaná např. v [3]) spočívá v aproximaci následující numerické hodnoty pomocí několika bodů mezi počátečním a koncovou hodnotou. Pro výpočet diferenciální rovnice ve tvaru:

$$y' = f(t, y) \qquad y(t_0) = y_0 \tag{2.1}$$

pomocí Runge-Kuttovy metody čtvrtého řádu se využívá rekurentního vzorce

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.2}$$

$$t_{n+1} = t_n + h \tag{2.3}$$

kde h je velikost časového kroku a

$$k_1 = f(t_n, y_n) \tag{2.4}$$

$$k_2 = f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \tag{2.5}$$

$$k_3 = f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2) \tag{2.6}$$

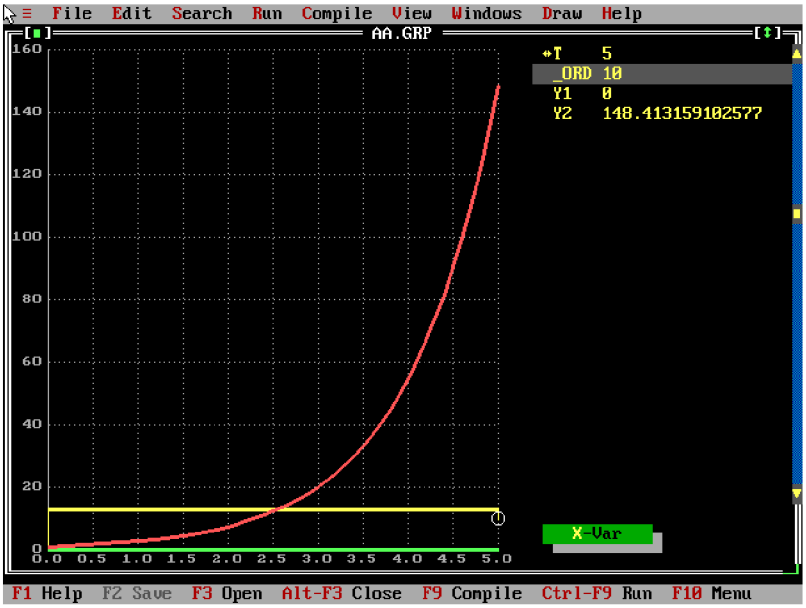
$$k_4 = f(t_n + h, y_n + hk_3) \tag{2.7}$$

Celková chyba této metody je h^4.

2.2 Jazyk TKSL

TKSL/386 byl vyvinut v jazyce Pascal. Poprvé byl prezentován v [4]. Pomocí jazyka TKSL, který je podobný jazyku Pascal, umožňuje zapsat soustavu diferenciálních rovnic prvního řádu a všechny důležité konstanty pro nalezení numerického řešení počáteční úlohy. Tento program využívá Taylorův rozvoj popsaný v kapitole 5.2 pro získávání numerického řešení počáteční úlohy. Jako příklad si můžeme uvést počáteční úlohu ve tvaru:

$$\begin{aligned} y_1' &= y_1 y_2 & y_1(t_0) &= 0 \\ y_2' &= y_2 & y_2(t_0) &= 1 \end{aligned} \tag{2.8}$$



Obrázek 2.1: Grafické zobrazení numerického řešení soustavy diferenciálních rovnic (2.8).

Tuto počáteční úlohu (pro koncovým časem 5 s, časovým krokem 0.2 s a přesností 10^{-20}) můžeme v jazyce TKSL zapsat následujícím způsobem:

```
var y1, y2;

const
    tmax = 15,
    dt = 0.2,
    eps = 1e-20;

system
    y1' = y1 * y2 & 0;
    y2' = y2 & 1;
sysend.
```

Program TKSL/386 také umožňuje grafické zobrazení výsledku výpočtu. Grafické zobrazení numerického řešení výše uvedené soustavy diferenciálních rovnic je zobrazeno na obrázku 2.1.

2.3 Systém numerických integrátorů

Výhodou transformace soustavy diferenciálních rovnic do elementárního tvaru je také možnost sestrojení analogového obvodu, řešícího danou soustavu. Takovýto obvod nazýváme systém numerických integrátorů. Soustavy rovnic v elementárním tvaru neobsahují žádné funkce a tudíž nám pro jejich realizaci pomocí elektronických obvodů stačí pouze integrační, sumační a násobící členy.

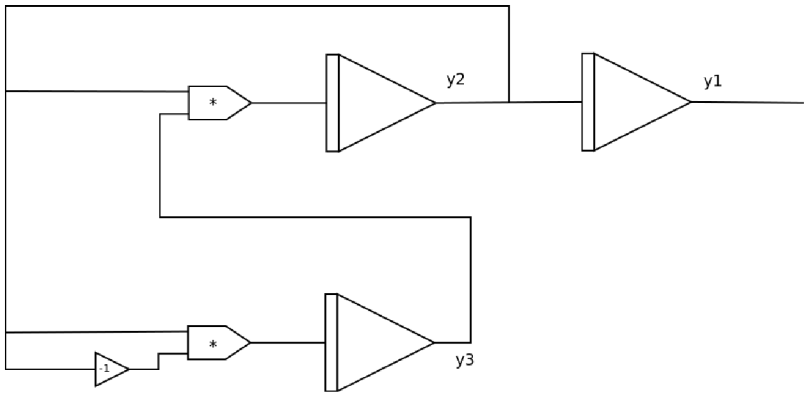
Pro příklad sestrojení systému numerických integrátorů zvolme diferenciální rovnici

$$y_1' = \sin(y_1) \qquad y_1(t_0) = 0 \tag{2.9}$$

Dle transformačních vztahů z tabulky 4.1 lze tuto rovnici transformovat na soustavu

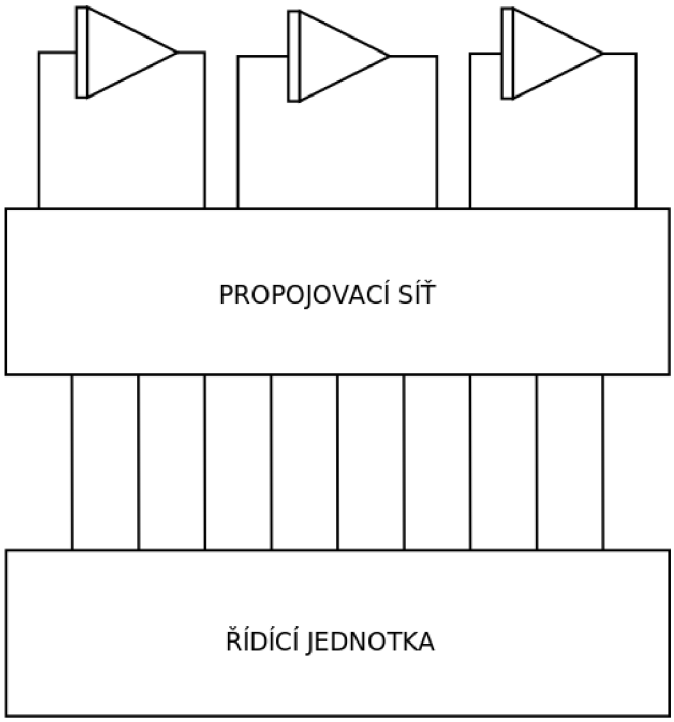
$$y_1' = y_2 \qquad y_1(t_0) = 0 \tag{2.10}$$
$$y_2' = y_3 y_2 \qquad y_2(t_0) = 0 \tag{2.11}$$
$$y_3' = -y_2 y_2 \qquad y_3(t_0) = 1 \tag{2.12}$$

Nalezení numerického řešení této soustavy lze realizovat pomocí elektronického obvodu znázorněného na obrázku 2.2.



Obrázek 2.2: Schéma elektronického obvodu realizujícího výpočet soustavy rovnic (2.10) - (2.12).

Sestrojení unikátního elektronického obvodu pro každou soustavu diferenciálních rovnice je nepraktické a v dnešní době se tento postup již nepoužívá. Systém rovnic lze tedy řešit buď použitím Taylorova rozvoje a matematických výpočtů, nebo simulací elektronických obvodů v paměti počítače a nebo hardwarově pomocí programovatelných hradlových polí, která umožňují přepojení obvodu bez nutnosti fyzického zásahu do struktury obvodu. Na obrázku 2.3 je znázorněno schéma moderní hardwarové implementace, pro řešení soustav libovolných tří diferenciálních rovnic.



Obrázek 2.3: Schéma programovatelného elektronického obvodu realizujícího výpočet soustavy rovnic (2.10) - (2.12).

Kapitola 3

Transformace soustavy rovnic do elementárního tvaru

Tato teorie vychází z prací [5] a [6].

Nechť $\mathbb{R}$ označuje množinu všech reálných čísel.

Definice 1. Nechť $X = \{t, y_1, y_1', y_2, y_2', \cdots\}$ je konečná množina formálních symbolů.

Definice 2. Nechť P je množina všech polynomů p ve tvaru:

$$p(y_1, \cdots, y_n) = a_1 y_{i1}^{n_1} y_{i2}^{n_2} \cdots y_{ir}^{n_r} + a_2 y_{j1}^{m_1} y_{j2}^{m_2} \cdots y_{jr}^{m_r} + \cdots \tag{3.1}$$

kde $a_i \in \mathbb{R}$, $y_i \in X$ a koeficienty $\{i_1, i_2, \cdots, i_r\}$, $\{j_1, j_2, \cdots, j_s\}$ tvoří vzájemně odlišné podmnožiny množiny $\{1, \cdots, n\}$. Počet členů na pravé straně je konečný.

Definice 3. Nechť B je konečná množina funkcí $f : \mathbb{R} \to \mathbb{R}$, které mají na intervalu $I \subseteq \mathbb{R}$ definovány derivace všech řádů a nechť ∂ je unární operátor mapující B na množinu polynomů s formálními proměnnými z B. Pro $f(t) \in B$ platí

$$\partial f(t) = p(f_1(t), f_2(t), \cdots, f_r(t)) \tag{3.2}$$

kde $p \in P$ a $f_i(t) \in B$. Množinu B nazýváme „množinou základních funkcí“.

Příklad 1. Mějme funkci $f(t) = \sin(t)$. Platí, že:

$$\begin{aligned} \sin'(t) &= \cos(t) \\ \cos'(t) &= -\sin(t) \end{aligned}$$

Zvolme množinu $B = \{sin(t), cos(t)\}$. Pak funkce $f(t)$ je základní funkce.

Definice 4. Nechť f je funkce ve tvaru $f : \mathbb{R} \to \mathbb{R}$. Vytvořme množinu $F_0 = \{f\}$. Definujme množiny $F_i, 0 < i$ předpisem

$$F_i = \{f, f_1, \cdots, f_n | \forall f \in F_{i-1}, f'(t) = p(f_1(t), \cdots, f_n(t)), p \in P\}$$

Označme F jako:

$$F = \bigcup_{i=1}^{\infty} F_i$$

Množinu F, pokud lze sestavit, nazýváme „derivační uzávěr funkce“ f, což značíme $f \xrightarrow{\partial} F$.

Lemma 1. Nechť B je množina základních funkcí s n prvky a pro libovolnou funkci $f \in B$ jsou definovány množiny F_i dle definice 4. Pak pro všechna $i \geq n-1$ platí že $F_i = F_{i+1}$.

Důkaz. Bez újmy na obecnosti můžeme předpokládat že $B = \{f_1, \cdots, f_n\}$ a

$$\begin{aligned} f_1' =& p_1(f_{1,1}, \cdots, f_{1,k_1}) \\ &\vdots \\ f_n' =& p_n(f_{n,1}, \cdots, f_{n,k_n}) \end{aligned}$$

Nechť $f = f_1$, pak $F_0 = \{f\}, F_1 = F_0 \bigcup \{f_{1,1}, \cdots, f_{1,k_1}\}, \cdots$. Z definice množiny F_k vyplývá, že pokud pro nějaké k platí $F_k = F_{k+1}$, pak to bude platit i pro všechna $i \geq k$ (množina F_k již obsahuje všechny funkce, které se vyskytují v derivacích ostatních funkcí z množiny F_k). K tomu dojde nejhůř v případě, že

$$f_1' = p_1(f_2), f_2' = p_2(f_3), \cdots, f_{n-1}' = p_{n-1}(f_n), f_n' = p_n(f_1)$$

a tudíž to platí v nejhorším případě pro $k = n-1$. □

Všimněme si, že na základě lemma 1 a definice 3 vyplývá, že každá funkce z B má konečný derivační uzávěr a naopak, pouze ty funkce, které mají konečný derivační uzávěr, mohou být prvky z B.

Definice 5. Nechť $\Phi(B, X)$ je množina výrazů ϕ definována těmito pravidly:

$$\begin{aligned} &(i)\ X \subseteq \Phi \\ &(ii)\ \forall f \in B, \forall \phi \in \Phi : f(\phi) \in \Phi \\ &(iii)\ \forall p \in P, \forall \phi_i \in \Phi : p(\phi_1, \cdots, \phi_n) \in \Phi \end{aligned}$$

Množinu $\Phi(B, X)$ budeme nadále označovat pouze symbolem Φ.

Příklad 2. Položíme-li množinu $B = \{sin(t), cos(t)\}$ a množinu $X = \{t, y_1, y_2, y_1', y_2'\}$ pak následující výrazy jsou prvky množiny Φ: $y_1, y_1 y_2', sin(y_2), cos(t), y_1 sin(t) + cos(y_1 + sin(t))y_2^2$.

Definice 6. Pro všechna $\phi \in \Phi$ definujeme $|\phi|$ těmito pravidly:

$$\begin{aligned} &(i)\ \forall y \in X : |y| = 1 \\ &(ii)\ \forall f \in B, \forall \phi \in \Phi : |f(\phi)| = |\phi| + 1 \\ &(iii)\ \forall p \in P, \tau_i \in \Phi : |p(\tau_1, \cdots, \tau_n)| = \max\{|\tau_1|, \cdots, |\tau_n|\} \end{aligned}$$

$|\phi|$ nazýváme „hloubkou výrazu ϕ“.

Příklad 3. Položme $B = \{sin(t), con(t), e^t\}$, $X = \{t, y_1, y_2, y_1', y_2'\}$ a $\phi_i \in \Phi$ jsou definovány takto:

$$\begin{aligned} \phi_1 =& t \\ \phi_2 =& y_1 y_2 \\ \phi_3 =& \sin(y_1) \\ \phi_4 =& y_1 y_1' + y_2 y_1 y_2' \\ \phi_5 =& y_1 e^{y_2 \sin(t)} \cos(t) + y_2 \end{aligned}$$

Pak hloubka těchto výrazů je: $|\phi_1| = 1$, $|\phi_2| = 1$, $|\phi_3| = 2$, $|\phi_4| = 1$, $|\phi_5| = 3$.

Dále předpokládáme základní pravidla pro derivování. $y_1, \cdots, y_n \in X$ jsou funkcí proměnné t a tedy její derivace lze přepsat na $\frac{dy_i}{dt} = y_i'$. Dále pro funkci $f(\phi)$, kde $f \in B$ a $\phi \in \Phi$ lze psát $\frac{\partial f(\phi)}{\partial \phi} = f'(\phi)$. Pro $\phi = f(\psi)$ lze psát $\phi' = f'(\psi)\psi'$.

Je-li $\phi \in \Phi$, kde

$$\phi = p(\tau_1, \cdots, \tau_n)$$

kde $\tau_i \in \Phi$, pak lze zapsat

$$\phi' = p'(\tau_1, \cdots, \tau_n) = \sum_{i=1}^{n} p_i(\tau_1, \cdots, \tau_n)\tau_i' \tag{3.3}$$

kde p_i je parciální derivace polynomu p vzhledem k proměnné τ_i

$$p_i(\tau_1, \cdots, \tau_n) = \frac{\partial p(\tau_1, \cdots, \tau_n)}{\partial \tau_i}$$

Definice 7. Nechť $\phi(y_1, \cdots, y_n) \in \Phi(B, X)$. Z předchozího předpokladu derivace lze odvodit, že

$$\phi'(y_1, \cdots, y_n) = \psi(y_1, \cdots, y_n, y_1', \cdots, y_n')$$

kde $\psi \in \Phi(B \cup B', X \cup X')$, $B' = \{f' | f \in B\}$ a $X' = \{y' | y \in X\}$. Místo vytváření množiny B' můžeme každý výskyt $f'(\tau)$, pro $\tau \in \Phi(B, Y)$, nahradit polynomem $p(f_1(\tau), \cdots, f_n(\tau))$, kde $p(f_1(t), \cdots, f_n(t)) = \partial f(t)$. Takto můžeme nadefinovat **B-substituci** předpisem:

$$\mathbf{B}(\phi'(y_1, \cdots, y_n)) = \psi(y_1, \cdots, y_n, y_1', \cdots, y_n') \in \Phi(B, X \cup X')$$

kde ve všech $\psi \in \Phi(B, X \cup X')$ nahradíme výskyt $f'(\tau)$ polynomem $p(f_1(\tau), \cdots, f_n(\tau))$.

Příklad 4. Nadefinujme množinu $B = \{\sin(t), \cos(t)\}$ a množinu $X = \{y_1, y_1'\}$. Položme

$$\phi(y_1) = \sin(y_1)$$

pak

$$\mathbf{B}(\phi'(y_1)) = \psi(y_1, y_1') = p(\tau, y_1')$$

kde

$$\begin{aligned} p &= \tau y_1' \\ \tau &= \cos(y_1) \end{aligned}$$

Lemma 2. Nechť $\phi(y_1, \cdots, y_n) \in \Phi$ a

$$\mathbf{B}(\phi'(y_1, \cdots, y_n)) = \psi(y_1, \cdots, y_n, y_1', \cdots, y_n')$$

pak platí, že

$$|\phi(y_1, \cdots, y_n)| = |\psi(y_1, \cdots, y_n, y_1', \cdots, y_n'))|$$

Důkaz. Je-li „hloubka“ ϕ rovna jedné, to znamená, že $\phi = p(y_1, \cdots, y_n)$, pak dle pravidel derivování můžeme psát

$$p'(y_1, \cdots, y_n) = \sum_{i=1}^{n} p_i(y_1, \cdots, y_n)y_i'$$

Je zřejmé, že v tomto případě lemma platí.

Nechť je hloubka ϕ rovna $r > 1$ a předpokládejme, že lemma platí pro všechny výrazy s výškou menší než r. Můžeme tedy psát $\phi(y_1, \cdots, y_n) = p(\tau_1, \cdots, \tau_n)$, kde $\tau_i \in \Phi$. Dle pravidel derivování platí

$$\mathbf{B}(p'(\tau_1, \cdots, \tau_n)) = \sum_{i=1}^{n} p_i(\tau_1, \cdots, \tau_n)\mathbf{B}(\tau_i')$$

Je zřejmé, že $|p(\tau_1, \cdots, \tau_n)| \geq |p_i(\tau_1, \cdots, \tau_n)|$, jelikož jsou v p_i všechny členy jako v p až na τ_i v případě že se τ_i v p vyskytuje pouze s první mocninou. Stačí tedy dokázat, že $|\mathbf{B}(\tau_i')| = |\tau_i|$ pro všechna $\tau_i \in \Phi, \forall i \in \{1, \cdots, n\}$. Z počátečního předpokladu to platí pro $|\tau_i| < r$. Pokud je $|\tau_i| = r$, pak musí platit $\tau_i = f(\xi)$, kde $f \in B, \xi \in \Phi$ a $|\xi| = r - 1$. τ_i' tedy můžeme přepsat do tvaru $f'(\xi)\xi'$ a $\mathbf{B}(\tau_i') = p(f_1(\xi), \cdots, f_s(\xi))\mathbf{B}(\xi')$ kde $f_i \in B, \forall i \in \{1, \cdots, s\}$. Vzhledem k tomu, že $|\xi| = r - 1$, pak je $|f_i(\xi)| = r$ a z počátečního předpokladu vyplývá, že $\mathbf{B}(\xi') = r - 1$. Pak ovšem platí, že $|\mathbf{B}(\tau_i')| = |p(f_1(\xi), \cdots, f_s(\xi))| = r$. □

Definice 8. Konečnou množinu rovnic Φ_n ve tvaru

$$\Phi_n = \left\{ \begin{array}{rl} y_1' = & \phi_1(y_1, \cdots, y_n) \\ & \vdots \\ y_n' = & \phi_n(y_1, \cdots, y_n) \end{array} \right\}$$

kde $\phi_i \in \Phi, \forall i \in \{1, \cdots, n\}$, nazýváme „soustavu B-rovnic nad Φ“, což značíme $\Phi_n \sqsubset \Phi$. Její hloubku definujeme předpisem

$$|\Phi_n| = \max\{|\phi_1|, \cdots, |\phi_n|\}$$

Definice 9. Nechť Φ_n a Ψ_m jsou dvě soustavy B-rovnic nad Φ ve tvaru

$$\Phi_n = \left\{ \begin{array}{rl} y_1' = & \phi_1(y_1, \cdots, y_n) \\ & \vdots \\ y_n' = & \phi_n(y_1, \cdots, y_n) \end{array} \right\}$$

$$\Psi_m = \left\{ \begin{array}{rl} y_1' = & \psi_1(y_1, \cdots, y_m) \\ & \vdots \\ y_m' = & \psi_m(y_1, \cdots, y_m) \end{array} \right\}$$

pro které platí, že $0 < n < m$. Nechť $\forall i \in \{1, \cdots, n\}$ platí

$$\psi_i(y_1, \cdots, y_n, \tau_1, \cdots, \tau_{m-n}) = \phi_i(y_1, \cdots, y_n)$$

kde $\tau_j(y_1, \cdots, y_n) \in \Phi, \forall j \in \{1, \cdots, m - n\}$ a

$$\begin{aligned} \mathbf{B}(\tau_j') = & \xi_j(y_1, \cdots, y_n, y_1', \cdots, y_n') = \\ & \xi_j(y_1, \cdots, y_n, \psi_1(y_1, \cdots, y_m), \cdots, \psi_n(y_1, \cdots, y_n)) = \\ & \psi_{n+j}(y_1, \cdots, y_m) \end{aligned}$$

Pak říkáme, že soustava Ψ_m rozšiřuje soustavu Φ_n, což značíme $\Phi_n \to \Psi_m$.

Příklad 5. Mějme soustavu rovnic $\Phi_1 \sqsubset \Phi(B, X)$, $B = \{\sin(t), \cos(t)\}$ a $X = \{y_1, y_1'\}$.

$$\begin{aligned}\Phi_1 =& \{y_1' = \phi_1(y_1)\}\\ \phi_1(y_1) =& \sin(y_1)\end{aligned}$$

Označme $\tau_1(y_1) = \sin(y_1)$ a

$$\psi_1(y_1, \tau_1(y_1)) = \tau_1(y_1)$$

Položme $\tau_1(y_1)$ rovno nové neznámé proměnné y_2. Pak

$$y_2' = \mathbf{B}(\tau_1'(y_1)) = y_1' \cos(y_1) = y_2 \cos(y_1) = \psi_2(y_1, y_2)$$

čímž dostáváme novou soustavu rovnice

$$\begin{aligned}\Psi_2 =& \begin{Bmatrix} y_1' = & \psi_1(y_1, y_2) \\ y_2' = & \psi_2(y_1, y_2) \end{Bmatrix}\\ \psi_1(y_1, y_2) =& y_2\\ \psi_2(y_1, y_2) =& y_2 \cos(y_1)\end{aligned}$$

kde $\Psi_2 \sqsubset \Phi(B, X_2), X_2 = \{y_1, y_2, y_1', y_2'\}$ a $\Phi_1 \to \Psi_2$.

Lemma 3 (Existence rozšíření s nižší hloubkou)**.** Pro každou soustavu rovnic $\Phi_n \sqsubset \Phi$ ve tvaru

$$\Phi_n = \begin{Bmatrix} y_1' = & \phi_1(y_1, \cdots, y_n) \\ & \vdots \\ y_n' = & \phi_n(y_1, \cdots, y_n) \end{Bmatrix}$$

kde $|\Phi_n| = d > 1$ existuje soustava rovnic $\Psi_m \sqsubset \Phi$ taková, že $\Phi_n \to \Psi_m$ a $|\Phi_n| > |\Psi_m|$.

Důkaz. Bez újmy na obecnosti můžeme psát

$$\begin{aligned}\phi_1(y_1, \cdots, y_n) =& p_1(\sigma_{1,1}, \cdots, \sigma_{1,k_1}, \pi_{1,1}, \cdots, \pi_{1,l_1})\\ &\vdots\\ \phi_n(y_1, \cdots, y_n) =& p_n(\sigma_{n,1}, \cdots, \sigma_{n,k_n}, \pi_{n,1}, \cdots, \pi_{n,l_n})\end{aligned}$$

kde $p_1 \cdots p_n$ jsou polynomy a $\sigma_{j,i}, \pi_{k,i} \in \Phi$, $|\sigma_{j,i}| < d$, $|\pi_{k,i}| = d$, $\forall i, j, k \in \{1, \cdots, n\}$. Jelikož hloubka je $|\Phi_n| = d$, pak musí alespoň jedno $\pi_{k,i}$ existovat. Vytvořme množinu všech výrazů hloubky d

$$\pi = \bigcup_{i=1}^{n} \{\pi_{i,1}, \cdots, \pi_{i,l_i}\}$$

Vzhledem k tomu, že $d > 1$ pak je množina π ve tvaru

$$\pi = \{f_{1,1}(\xi_1), \cdots, f_{1,r_1}(\xi_1), \cdots, f_{k,1}(\xi_k), \cdots, f_{k,r_k}(\xi_k)\}$$

kde $f_{i,j} \in B$, $\xi_i \in \Phi$ a $|\xi_i| = d - 1$. Pro všechna $i \in \{1, \cdots, k\}$ vytvořme množinu

$$F_{\xi_i} = \{f | f(\xi_i) \in \pi\}$$

a množinu

$$G_{\xi_i} = \bigcup_{f \in F_{\xi_i}} D : f \xrightarrow{\partial} D$$

$G_{\xi_i} \subseteq B$ tedy představuje derivační uzávěr všech funkcí z π, které mají jako nezávislou proměnnou $\xi_i \in \Phi$. Dále pro každé ξ_i a pro každou funkci $g \in G_{\xi_i}$ zaveďme neznámou proměnnou $y_{n+l} = g(\xi_i)$ tak, aby žádné dvě proměnné y neměly stejné indexy. Vytvořme novou množinu proměnných

$$X_2 = X \bigcup \{y_{n+l}, y'_{n+l} | y \text{ je nově zavedená proměnná}\}$$

Je zřejmé, že množinu X_2 lze vyjádřit ve tvaru $\{y_1, \cdots, y_m, y'_1, \cdots, y'_m\}$, kde $n < m$.

Nyní pro všechna $o \in \{1, \cdots, n\}$ nahraďme výraz $\pi_{i,j}$ ve ϕ_o

$$\phi_o(y_1, \cdots, y_n) = p_o(\sigma_{o,1}, \cdots, \sigma_{o,k_o}, \pi_{o,1}, \cdots, \pi_{o,l_o})$$

nově zavedenou proměnnou $y_{o,j}$. Takto nově vytvořené výrazy položme rovny $\psi_o(y_1, \cdots, y_m)$. Nyní pro každý výraz $\xi_i \in \Phi$ a každou funkci $g \in G_{\xi_i}$ máme novou proměnnou y_j. Za předpokladu, že $g'(t) = p(g_1(t), \cdots, g_l(t))$ je derivace nově zavedené proměnné rovna

$$\mathbf{B}(y'_j) = \mathbf{B}(\xi'_i) p_j(g_1(\xi_i), \cdots, g_l(\xi_i))$$

kde $g_k \in G_{\xi_i}$, $\forall k \in \{1, \cdots, l\}$. Tudíž byla pro výraz $g_k(\xi_i)$ zavedena nová proměnná a polynom p_j můžeme převést do tvaru $p_j(y_{n+1}, \cdots, y_m)$.

Označme výraz

$$\mathbf{B}(\xi'_i) p_j(y_{n+1}, \cdots, y_m) = \psi_j(y_1, \cdots, y_m)$$

Sestrojme novou soustavu $\Psi_m \sqsubset \Phi(B, X_2)$ ve tvaru

$$\Psi_m = \left\{ \begin{array}{rc} y'_1 = & \psi_1(y_1, \cdots, y_m) \\ & \vdots \\ y'_m = & \psi_m(y_1, \cdots, y_m) \end{array} \right\}$$

Je zřejmé, že hloubka výrazů $\psi_1, \cdots, \psi_n$ je menší než d, jelikož všechny výrazy výšky d z nich byly odstraněny. Hloubka výrazu ψ_j, $\forall j \in \{n+1, \cdots, m\}$ je

$$|\psi_j| = |\mathbf{B}(\xi'_i) p_j(y_{n+1}, \cdots, y_m)| = \max\{|\mathbf{B}(\xi'_i)|, |p_j(y_{n+1}, \cdots, y_m)|\}$$

Dle definice 6 je hloubka polynomu $p_j(y_{n+1}, \cdots, y_m)$ rovna 1 a dle lemma 2 platí

$$|\mathbf{B}(\xi'_i)| = |\xi_i| = d - 1$$

□

Lemma 4. Pro každý systém rovnic $\Phi_n \sqsubset \Phi$ existuje konečná posloupnost množin $\Phi^1_{n_1}, \cdots, \Phi^k_{n_k}$, $k > 0$ takových, že platí

$$(a)\ \Phi_n = \Phi^1_{n_1}$$
$$(b)\ \Phi^i_{n_i} \to \Phi^{i+1}_{n_{i+1}}, 0 < i \leq k$$
$$(c)\ |\Phi^k_{n_k}| = 1$$

Důkaz. Důkaz je přímým důsledkem lemma 3. Dle tohoto lemma víme, že pro libovolné $\Phi^i_{n_i}$ můžeme sestavit $\Phi^{i+1}_{n_{i+1}}$ takové, že $|\Phi^i_{n_i}| > |\Phi^{i+1}_{n_{i+1}}|$. Vzhledem k tomu, že hloubka soustavy $|\Phi_n|$ je konečná, pak i posloupnost systémů s nižší hloubkou bude konečná. □

Definice 10. Nechť $\Phi_n \sqsubset \Phi(B, X)$ je soustava B-rovnic. Množinu Φ_n můžeme použít pro definování soustavy diferenciálních rovnic prvního řádu

$$\begin{aligned} y_1' =& \phi_1(y_1, \cdots, y_n) \\ &\vdots \\ y_n' =& \phi_n(y_1, \cdots, y_n) \end{aligned} \tag{3.4}$$

s počátečními podmínkami

$$y_1(t_0) = a_{10}, \cdots, y_n(t_0) = a_{n0} \tag{3.5}$$

Soustavu (3.4) společně s počátečními podmínkami (3.5) nazýváme „počáteční úlohou“ a označujeme jí enticí $(\Phi_n, t_0, a_{10}, \cdots, a_{n0})$, zkráceně (Φ_n) nebo zápisem

$$\begin{aligned} y_1' =& \phi_1(y_1, \cdots, y_n) & \qquad y_1(t_0) =& a_{10} \\ &\vdots & &\vdots \\ y_n' =& \phi_n(y_1, \cdots, y_n) & y_n(t_0) =& a_{n0} \end{aligned}$$

Lemma 5. Nechť $\Phi_n \sqsubset \Phi(B, X)$ je soustava B-rovnic a $(\Phi_n, t_0, a_{10}, \cdots, a_{n0})$ je počáteční úloha, pro kterou platí, že $|\Phi_n| > 1$. Pak existuje počáteční úloha $(\Psi_m, t_0, b_{10}, \cdots, b_{n0})$ taková, že $\Phi_n \to \Psi_m$ a

(a) $|\Phi_n| > |\Psi_m|$

(b) pokud existuje řešení pro $(\Psi_m, t_0, b_{10}, \cdots, b_{n0})$ ve tvaru $\{\overline{y}_1(t), \cdots, \overline{y}_m(t)\}$ na intervalu $I \subseteq \mathbb{R}, t_0 \in I$, pak existuje i řešení pro $(\Phi_n, t_0, a_{10}, \cdots, a_{n0})$ na intervalu I ve tvaru $\{y_1(t), \cdots, y_n(t)\}$ takové, že $y_i(t) = \overline{y}_i(t)$, $\forall t \in I$, $\forall i \in \{1, \cdots, n\}$.

Důkaz. Z lemma 3 víme, že existuje soustava B-rovnic Ψ_m taková, že platí $\Phi_n \to \Psi_m$ a $|\Phi_n| > |\Psi_m|$. Z definice 9 víme, že $\forall i \in \{1, \cdots, n\}$ platí

$$\psi_i(y_1, \cdots, y_n, \tau_1(y_1, \cdots, y_n), \cdots, \tau_{m-n}(y_1, \cdots, y_n)) = \phi_i(y_1, \cdots, y_n) \tag{3.6}$$

Z konstrukce množiny Ψ_m v důkazu lemma 3 vyplývá, že počáteční úlohu $(\Psi_m, t_0, b_{10}, \cdots, b_{n0})$ můžeme zapsat ve tvaru

$$\begin{aligned} y_1' =& \psi_1(y_1, \cdots, y_n, y_{n+1}, \cdots, y_m) & \qquad y_1(t_0) =& a_{10} \\ &\vdots & &\vdots \\ y_n' =& \psi_n(y_1, \cdots, y_n, y_{n+1}, \cdots, y_m) & y_n(t_0) =& a_{n0} \\ y_{n+1}' =& \psi_{n+1}(y_1, \cdots, y_n, y_{n+1}, \cdots, y_m) & y_{n+1}(t_0) =& \tau_{n+1}(a_{10}, \cdots, a_{n0}) \\ &\vdots & &\vdots \\ y_m' =& \psi_m(y_1, \cdots, y_n, y_{n+1}, \cdots, y_m) & y_m(t_0) =& \tau_{m-n}(a_{10}, \cdots, a_{n0}) \end{aligned}$$

Nechť řešením této počáteční úlohy na intervalu I je $\{\overline{y}_1(t), \cdots, \overline{y}_m(t)\}$. Rovnici (3.6) tedy můžeme přepsat do tvaru

$$\psi_i(\overline{y}_1, \cdots, \overline{y}_n, \tau_1(\overline{y}_1, \cdots, \overline{y}_n), \cdots, \tau_{m-n}(\overline{y}_1, \cdots, \overline{y}_n)) = \phi_i(\overline{y}_1, \cdots, \overline{y}_n) \tag{3.7}$$

Z konstrukce množiny Ψ_m dále víme, že $\forall j \in \{1, \cdots, m-n\}$

$$\overline{y}_{n+j} = \tau_j(\overline{y}_1, \cdots, \overline{y}_n)$$

a derivace proměnné $\overline{y}_{n+j}$ podle t je

$$\begin{aligned}\overline{y}'_{n+j} =& \mathbf{B}(\tau'_j(\overline{y}_1, \cdots, \overline{y}_n)) = \\ & \xi_j(\overline{y}_1, \cdots, \overline{y}_n, \overline{y}'_1, \cdots, \overline{y}'_n) = \\ & \xi_j(\overline{y}_1, \cdots, \overline{y}_n, \psi_1(\overline{y}_1, \cdots, \overline{y}_m), \cdots, \psi_n(\overline{y}_1, \cdots, \overline{y}_m)) = \\ & \psi_{n+j}(\overline{y}_1, \cdots, \overline{y}_m)\end{aligned}$$

a tedy můžeme dosadit ψ_{n+j} do rovnice (3.7)

$$\phi_i(\overline{y}_1, \cdots, \overline{y}_n) = \psi_i(\overline{y}_1, \cdots, \overline{y}_m) = \overline{y}_i$$

což dokazuje, že množina $\{\overline{y}_1, \cdots, \overline{y}_n\}$ je řešením počáteční úlohy $(\Psi_m, t_0, b_{10}, \cdots, b_{n0})$. □

Lemma 6. Ke každé počáteční úloze $(\Phi_n, t_0, a_{10}, \cdots, a_{n0})$, kde $\Phi_n \sqsubset \Phi(B, Y)$ a $|\Phi_n| > 1$ existuje počáteční úloha $(\Psi_m, t_0, b_{10}, \cdots, b_{m0})$ taková, že $\Phi_n \to \Psi_m$, $|\Psi_m| = 1$ a pokud existuje na intervalu $I \subseteq \mathbb{R}$, $t_0 \in I$ řešení úlohy (Ψ_m) ve tvaru $\{\overline{y}_1(t), \cdots, \overline{y}_m(t)\}$, pak existuje i řešení úlohy (Φ_n) na intervalu I ve tvaru $\{y_1(t), \cdots, y_n(t)\}$, kde $y_i(t) = \overline{y}_i(t)$, $\forall t \in I$ a $\forall i \in \{1, \cdots, n\}$.

Důkaz. Z lemma 5 vyplývá, že lze sestavit konečnou posloupnost počátečních úloh

$$(\Phi^1_{n_1}, t_0, a^1_{10}, \cdots, a^1_{n_1 0}), (\Phi^2_{n_2}, t_0, a^2_{10}, \cdots, a^2_{n_2 0}), \cdots, (\Phi^m_{n_m}, t_0, a^m_{10}, \cdots, a^m_{n_m 0})$$

takovou, že platí

(a) $(\Phi^1_{n_1}, t_0, a^1_{10}, \cdots, a^1_{n_1 0}) = (\Phi_n, t_0, a_{10}, \cdots, a_{n0})$

(b) $\Phi^1_{n_1} \to \Phi^2_{n_2} \to \cdots \to \Phi^m_{n_m}$

(c) $|\Phi^1_{n_1}| > |\Phi^2_{n_2}| > \cdots > |\Phi^m_{n_m}|$

a zároveň platí, že pokud na intervalu $I \subseteq \mathbb{R}$ existuje řešení úlohy $(\Phi^m_{n_m})$, pak existuje i řešení všech ostatních úloh. Tato posloupnost bude tak velká, aby platilo $|\Phi^m_{n_m}| = 1$. Vzhledem k tomu, že hloubka Φ_n je konečná, pak i posloupnost bude konečná. □

3.1 Příklad transformace počáteční úlohy

Mějme soustavu diferenciálních rovnic ve tvaru

$$y'_1 = e^{-\cos(t)} \cdot \sin(y_2) \qquad y_1(t_0) = y_{1,0} \tag{3.8}$$

$$y'_2 = y_2 \qquad y_2(t_0) = y_{2,0} \tag{3.9}$$

Označme

$$y_{11} = \sin(y_2) \tag{3.10}$$

$$y_{12} = \cos(y_2) \tag{3.11}$$

$$y_{13} = \cos(t) \tag{3.12}$$

$$y_{14} = \sin(t) \tag{3.13}$$

$$y_{15} = e^{y_{13}} \tag{3.14}$$

pak lze rovnici (3.8) přepsat do tvaru

$$y_1' = y_{12} \cdot y_{11} \tag{3.15}$$

Nyní sestavíme diferenciální rovnice pro nově zavedené proměnné.

$$y_{11}' = \cos(y_2) y_2' = y_{12} y_2 \qquad y_{11}(t_0) = \sin(y_2(t_0)) \tag{3.16}$$
$$y_{12}' = -\sin(y_2) y_2' = -y_{11} y_2 \qquad y_{12}(t_0) = \cos(y_2(t_0)) \tag{3.17}$$
$$y_{13}' = -\sin(t) = -y_{14} \qquad y_{13}(t_0) = \cos(t_0) \tag{3.18}$$
$$y_{14}' = \cos(t) = y_{13} \qquad y_{14}(t_0) = \sin(t_0) \tag{3.19}$$
$$y_{15}' = e^{y_{13}} y_{13}' = -y_{15} y_{14} \qquad y_{15}(t_0) = e^{y_{13}(t_0)} \tag{3.20}$$

Rovnice (3.16) - (3.20) společně s rovnicemi

$$y_1' = y_{12} \cdot y_{11} \qquad y_1(t_0) = y_{1,0} \tag{3.21}$$
$$y_2' = y_2 \qquad y_2(t_0) = y_{2,0} \tag{3.22}$$

nyní tvoří elementární tvar počáteční úlohy ekvivalentní k úloze zadané rovnicemi (3.8) a (3.9).

Kapitola 4

Transformace elementárních funkcí

Cílem této kapitoly je sestavení transformačních rovnic pro jednotlivé elementární funkce tak, aby se celý proces transformace dal automatizovat.

Většina matematický funkcí se v počítačích počítá pomocí Taylorova rozvoje. Přesnost jejich výpočtu (která závisí na počtu členů rozvoje) většinou řídí použitá matematická knihovna. Vzhledem k tom, že pro výpočet diferenciálních rovnic využíváme Taylorův rozvoj, je zbytečné, aby se používal na dvou místech (při výpočtu matematický funkcí a při výpočtu diferenciálních rovnic). Proto provedeme transformaci funkcí na diferenciální rovnice, což nám zároveň volbou řádu umožní dynamicky řídit přesnost výpočtu funkcí. Myšlenku transformace znázorníme na následujícím příkladu.

Mějme dvě funkce: $\sin(t)$ a $\cos(t)$. Označme si je proměnnými $y_1 = \sin(t)$ a $y_2 = \cos(t)$. Derivací těchto proměnných podle t dostaneme následující soustavu diferenciálních rovnic:

$$\begin{aligned} y_1' &= y_2 & y_1(t_0) &= \sin(t_0) \\ y_2' &= -y_1 & y_2(t_0) &= \cos(t_0) \end{aligned} \tag{4.1}$$

Vyšší derivace se budou získávat dle principu

$$y_1'' = (y_1')' = y_2' \tag{4.2}$$

$$y_1''' = (y_1'')' = (y_2')' = -y_1' \tag{4.3}$$

Dle upraveného vzorce pro Taylorův rozvoj (5.8) můžeme funkci $\sin(t+h)$ přepsat do tvaru

$$\begin{aligned} \sin(t+h) &= \sin(t) + h\sin'(t) + \frac{h^2}{2!}\sin''(t) + \frac{h^3}{3!}\sin'''(t) + \cdots = \\ &= \sin(t) + h\cos(t) + \frac{h^2}{2!}(-\sin(t)) + \frac{h^3}{3!}(-\cos(t)) + \cdots \end{aligned} \tag{4.4}$$

Spustíme-li výpočet na soustavě diferenciálních rovnic (4.1), budou se jednotlivé hodnoty proměnné y_1 počítat dle vzorce

$$y_1(t+h) = y_1(t) + hy_1'(t) + \frac{h^2}{2!}y_1''(t) + \frac{h^3}{3!}y_1'''(t) + \cdots \tag{4.5}$$

Po dosazení jednotlivých derivací do rovnice (4.5) dle přepisů získáváme rovnici

$$y_1(t+h) = y_1(t) + hy_2(t) + \frac{h^2}{2!}(-y_1(t)) + \frac{h^3}{3!}(-y_2(t)) + \cdots \tag{4.6}$$

a tedy dostáváme vztah

$$\sin(t+h) = \sin(t) + h\cos(t) + \frac{h^2}{2!}(-\sin(t)) + \frac{h^3}{3!}(-\cos(t)) + \cdots \tag{4.7}$$

který je totožný se vztahem (4.4). Z toho plyne, že funkce můžeme nahradit proměnnými, které představují přepis rovnic do diferenciálních rovnic a způsob výpočtu takto nahrazených rovnic bude stejný.

V následujícím textu budeme vycházet z označení $y_1 = f(x(t))$ kde f je elementární funkce a $x(t_0) = x_0$. Dále předpokládejme, že

$$x'(t) = p(x_1(t), \cdots, x_n(t)) \tag{4.8}$$

kde p je polynom. Tuto derivaci budeme nadále zkráceně označovat pouze symbolem p.

Transformační rovnice lze sestavit pouze pro funkce, které mají konečný derivační uzávěr, což znamená, že jejich derivační uzávěr tvoří konečnou množinu funkcí. Nicméně ukážeme, že všechny běžně používané funkce v technických aplikacích konečný derivační uzávěr mají.

4.1 Odvození transformačních vztahů

4.1.1 Funkce sin

$$\begin{array}{lll} \sin(x(t)) = y_1 & y_1' = \cos(x(t)) \cdot p = y_2 \cdot p & y_1(0) = \sin(x(0)) \\ & y_2' = -\sin(x(t)) \cdot p = -y_1 \cdot p & y_2(0) = \cos(x(0)) \end{array} \tag{4.9}$$

$\forall x(t) \in \mathbb{R}$

Derivačním uzávěrem funkce sin je $\{\sin, \cos\}$.

4.1.2 Funkce cos

$$\begin{array}{lll} \cos(x(t)) = y_1 & y_1' = -\sin(x(t)) \cdot p = -y_2 \cdot p & y_1(0) = \cos(x(0)) \\ & y_2' = \cos(x(t)) \cdot p = y_1 \cdot p & y_2(0) = \sin(x(0)) \end{array} \tag{4.10}$$

$\forall x(t) \in \mathbb{R}$

Derivačním uzávěrem funkce cos je $\{\sin, \cos\}$.

4.1.3 Funkce tg

$$\begin{aligned} \mathrm{tg}'(t) = \left(\frac{\sin(t)}{\cos(t)}\right)' = \frac{\sin'(t)\cos(t) - \sin(t)\cos'(t)}{\cos^2(t)} = \\ = \frac{\sin^2(t) + \cos^2(t)}{\cos^2(t)} = \mathrm{tg}^2(t) + 1 \end{aligned} \tag{4.11}$$

$$\mathrm{tg}(x(t)) = y_1 \quad y_1' = (1 + y_1^2) \cdot p \quad y_1(0) = \mathrm{tg}(x(0)) \tag{4.12}$$

$\forall x(t) \in \mathbb{R} \setminus \left\{\frac{\pi}{2} + k\pi | k \in \mathbb{N}\right\}$

Derivačním uzávěrem funkce tg tedy je $\{\mathrm{tg}\}$.

4.1.4 Funkce cotg

$$\mathrm{cotg}(x(t)) = y_1 \quad y_1' = -(1 + y_1^2) \cdot p \quad y_1(0) = \mathrm{cotg}(x(0)) \tag{4.13}$$

$\forall x(t) \in \mathbb{R} \setminus \{k\pi | k \in \mathbb{N}\}$

4.1.5 Funkce arcsin

$$\arcsin(t)' = \frac{1}{\sqrt{1-t^2}} \tag{4.14}$$

$$\left((\sqrt{1-t^2})^{-\frac{1}{2}}\right)' = -\frac{1}{2}(\sqrt{1-t^2})^{-\frac{3}{2}}(-2t) \tag{4.15}$$

$$\begin{array}{lll} \arcsin(x(t)) = y_1 & y_1' = y_2 & y_1(0) = \arcsin(x(0)) \\ & y_2' = x \cdot p \cdot y_2 \cdot (1-x^2) & y_2(0) = \frac{1}{\sqrt{1-x^2(0)}} \end{array} \tag{4.16}$$

$\forall x(t) \in \langle -1, 1 \rangle$

4.1.6 Funkce arccos

$$\begin{array}{lll} \arccos(x(t)) = y_1 & y_1' = -y_2 & y_1(0) = \arccos(x(0)) \\ & y_2' = x \cdot p \cdot y_2 \cdot (1-x^2) & y_2(0) = \frac{1}{\sqrt{1-x^2(0)}} \end{array} \tag{4.17}$$

$\forall x(t) \in \langle -1, 1 \rangle$

4.1.7 Funkce arctg

$$\operatorname{arctg}(t)' = \frac{1}{1+t^2} \tag{4.18}$$

$$\left((\frac{1}{1+t^2})^{-1}\right)' = -1(\frac{1}{1+t^2})^{-2}2t \tag{4.19}$$

$$\begin{array}{lll} \operatorname{arctg}(x(t)) = y_1 & y_1' = y_2 & y_1(0) = \operatorname{arctg}(x(0)) \\ & y_2' = -2 \cdot y_2^2 \cdot x \cdot p & y_2(0) = \frac{1}{1+x^2(0)} \end{array} \tag{4.20}$$

$\forall x(t) \in \mathbb{R}$

4.1.8 Funkce arccotg

$$\begin{array}{lll} \operatorname{arccotg}(x(t)) = y_1 & y_1' = -y_2 & y_1(0) = \operatorname{arccotg}(x(0)) \\ & y_2' = -2 \cdot y_2^2 \cdot x \cdot p & y_2(0) = \frac{1}{1+x^2(0)} \end{array} \tag{4.21}$$

$\forall x(t) \in \mathbb{R}$

4.1.9 Funkce exp

$$e^{x(t)} = y_1 \quad y_1' = p \cdot y_1 \quad y_1(0) = e^{x(0)} \tag{4.22}$$

$\forall x(t) \in \mathbb{R}$

4.1.10 Funkce ln

$$\ln(x(t)) = y_1 \quad y_1' = \frac{p}{x} \quad y_1(0) = \ln(x(0)) \tag{4.23}$$

$\forall x(t) \in \mathbb{R}, x(t) > 0$

y_1	rovnice	počáteční podmínky
$\sin(x)$	$y_1' = y_2 p$ $y_2' = -y_1 p$	$y_1(t_0) = \sin(x(t_0))$ $y_2(t_0) = \cos(x(t_0))$
$\cos(x)$	$y_1' = -y_2 p$ $y_2' = y_1 p$	$y_1(t_0) = \cos(x(t_0))$ $y_2(t_0) = \sin(x(t_0))$
$\mathrm{tg}(x)$	$y_1' = (1 + y_1^2)p$	$y_1(t_0) = \mathrm{tg}(x(t_0))$
$\mathrm{cotg}(x)$	$y_1' = -(1 + y_1^2)p$	$y_1(t_0) = \mathrm{cotg}(x(t_0))$
$\arcsin(x)$	$y_1' = y_2$ $y_2' = x \cdot p \cdot y_2 \cdot (1 - x^2)$	$y_1(t_0) = \arcsin(x(t_0))$ $y_2(t_0) = \frac{1}{\sqrt{1-x^2(t_0)}}$
$\arccos(x)$	$y_1' = -y_2$ $y_2' = x \cdot p \cdot y_2 \cdot (1 - x^2)$	$y_1(t_0) = \arccos(x(t_0))$ $y_2(t_0) = \frac{1}{\sqrt{1-x^2(t_0)}}$
$\mathrm{arctg}(x)$	$y_1' = y_2$ $y_2' = -2 \cdot y_2^2 \cdot x \cdot p$	$y_1(t_0) = \mathrm{arctg}(x(t_0))$ $y_2(t_0) = \frac{1}{1+x^2(t_0)}$
$\mathrm{arccotg}(x)$	$y_1' = -y_2$ $y_2' = -2 \cdot y_2^2 \cdot x \cdot p$	$y_1(t_0) = \mathrm{arccotg}(x(t_0))$ $y_2(t_0) = \frac{1}{1+x^2(t_0)}$
$\exp(x)$	$y_1' = p \cdot y_1$	$y_1(t_0) = e^{x(t_0)}$
$ln(x)$	$y_1' = \frac{p}{x}$	$y_1(t_0) = \ln(x(t_0))$

Tabulka 4.1: Transformační tabulka elementárních funkcí.[a]

[a] Funkce $\sqrt[n]{x(t)}$ není implementována, jelikož může být nahrazena ekvivalentní funkcí $e^{n \ln(x(t))}$.

4.2 Algoritmus transformace

Algoritmus 2 formálně popisuje transformaci soustavy diferenciálních rovnic v obecném tvaru (na pravé straně se mohou vyskytovat funkce definované v tabulce 4.1) na soustavu diferenciálních rovnic v polynomiálním tvaru. Druhý cyklus zajišťuje nahrazení derivací vzniklých při transformaci za výrazy bez derivací. Toto je možné, jelikož soustava rovnic definuje vztahy pro jednotlivé derivace a pokud se množina Ψ prochází setříděná podle toho jak vznikala, tak postupně dostáváme výrazy bez derivací. Funkce byly vždy nahrazeny novým symbolem a tudíž původní rovnice jsou již bez derivací. Jediné rovnice které mohou obsahovat derivace jsou nově zavedené rovnice a ty obsahují derivace pouze původních

proměnných, nebo rovnic definovaných dříve. K zacyklení tedy nemůže dojít.

Funkce `NahradFn`

Vstup: Výraz ϕ, ve kterém se mají nahradit funkce a množina transformovaných rovnic Ψ
Výstup: Výraz ψ bez funkcí a množina aktualizovaných transformovaných rovnic Ψ
begin
  if ϕ *obsahuje vyraz ve tvaru fn*(α, Ψ) **then**
    $\beta \leftarrow$ NahradFn(α);
    $\Psi \leftarrow \Psi \cup$ Nové rovnice podle transformační tabulky 4.1 pro výraz fn(β);
    $\psi \leftarrow \phi$ ve kterém výraz fn(α) nahradíme nově zavedenou proměnnou odpovídající výrazu fn(β);
  end
end

Algoritmus 2: Algoritmus transformace rovnic do polynomiálního tvaru

Vstup: Množina diferenciálních rovnic Φ v obecném tvaru
Výstup: Množina diferenciálních rovnic Ψ v polynomiálním tvaru
begin
  $\Psi \leftarrow \emptyset$;
  foreach „$y' = \phi$“ $\in \Phi$ **do**
    $\Psi \leftarrow \Psi \cup$ „$y' =$ NahradFn(ϕ, Ψ)“;
  end
  foreach „$y' = \psi$“ $\in \Psi$ **do**
    if ψ *obsahuje derivaci* **then**
      Nahraď derivaci v ψ za výraz předepsaný příslušnou rovnicí v Ψ pro danou proměnnou;
    end
  end
end

Kapitola 5

Taylorův teorém a jeho aplikace při řešení diferenciálních rovnic

5.1 Taylorovy polynomy generované funkcí

Pro danou funkci f hledáme takový polynom, který se s touto funkcí bude shodovat v určitém okolí bodu a. Předpokládejme, že funkce f má v bodě $x = a$ derivace až do řádu n, kde $n \geq 1$. Pokusme se nalézt polynom P, který se s funkcí f shoduje v bodě a v prvních n derivacích. Máme tedy $n + 1$ podmínek

$$P(a) = f(a), \qquad P'(a) = f'(a), \quad \cdots, \quad P^{(n)}(a) = f^{(n)}(a) \tag{5.1}$$

a polynom řádu n

$$P(x) = c_0 + c_1(x - a) + c_2(x - a)^2 + \cdots + c_n(x - a)^n \tag{5.2}$$

pro který musíme dopočítat $n + 1$ koeficientů. Lze ukázat, že pro každou funkci f existuje nanejvýš jeden polynom P s těmito vlastnostmi.

Věta 1. Je-li dána funkce f v bodě $x = a$ se všemi derivacemi až do řádu n, pak existuje právě jeden polynom P řádu $\leq n$, který splňuje $n + 1$ podmínek

$$P(a) = f(a), \qquad P'(a) = f'(a), \quad \cdots, \quad P^{(n)}(a) = f^{(n)}(a)$$

Tento polynom je dán předpisem

$$P(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x - a)^k \tag{5.3}$$

Důkaz. Polynom P můžeme zapsat ve tvaru:

$$P(x) = c_0 + c_1(x - a) + c_2(x - a)^2 + \cdots + c_n(x - a)^n \tag{5.4}$$

Pokud položíme $x = a$, dostaneme $P(a) = c_0$. Z první podmínky tedy dostáváme $P(a) = f(a)$. Nyní zderivujeme obě strany rovnice (5.4) a opět položíme $x = a$. Dostáváme $P'(a) = c_1$ a tedy $c_1 = f'(a)$. Zopakujeme-li tento postup znovu, dostáváme $P''(a) = 2c_2$ a tedy $c_2 = f''(a)/2$. Po k derivacích dostaneme $P^{(k)}(a) = k!c_k$, což nám dává vzorec:

$$c_k = \frac{f^{(k)}(a)}{k!} \tag{5.5}$$

□

Říkáme, že polynom (5.3) je „Taylorovým polynomem stupně n generovaným funkcí f v bodě a“. Tuto skutečnost budeme značit $T_n f$. Hodnotu tohoto polynomu v bodě x budeme značit $T_n f(x; a)$, nebo pouze $T_n f(x)$, pokud je $a = 0$.

Podrobněji se danou problematikou zabývá [1].

5.2 Taylorův teorém

Teorém 1 (Taylorův teorém)**.** *Za předpokladu, že $f^{(n+1)}$ funkce $f(t)$ je na intervalu (t_0, t_1) spojitá a hodnoty t a $t + h$ leží také v intervalu (t_0, t_1), pak platí*

$$f(t+h) = f(t) + hf'(t) + \frac{1}{2}h^2 f''(t) + \cdots + \frac{1}{n!}h^n f^{(n)}(t) + R_{n+1} \tag{5.6}$$

kde zbytek R_{n+1} je roven

$$R_{n+1} = \frac{1}{(n+1)!}h^{n+1} f^{(n+1)}(\eta) \tag{5.7}$$

a η je bod mezi t a $t + h$.

Pokud má funkce f definovány všechny derivace a tyto derivace jsou na intervalu (t_0, t_1) spojité, nazýváme jí „nekonečně diferencovatelnou“, a vzorec (5.6) lze přepsat do tvaru

$$f(t+h) = f(t) + \sum_{k=1}^{\infty} \frac{h^k}{k!} f^{(k)}(t) \tag{5.8}$$

5.3 Taylorův rozvoj s reziduem

Uvažujme nyní chybu způsobenou aproximací. Je patrné, že reziduum aproximace je $E_n(x) = f(x) - T_n f(x)$. A tedy pokud má funkce f n prvních derivací, můžeme psát

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k + E_n(x) \tag{5.9}$$

Tomuto vzorci se říká „Taylorův vzorec s reziduem“. Tento vzorec je užitečný, pokud můžeme odhadnout velikost $E_n(x)$ a tudíž tak stanovit chybu aproximace. Nyní si funkci $E_n(x)$ převedeme do integrálního tvaru.

Věta 2. Předpokládejme, že funkce f má spojitou druhou derivaci f'' v určitém okolí bodu a. Pak pro každé x v tomto okolím platí

$$f(x) = f(a) + f'(a)(x-a) + E_1(x) \tag{5.10}$$

kde

$$E_1(x) = \int_a^x (x-t) f''(t)\,dt \tag{5.11}$$

Důkaz. Rovnici (5.10) můžeme přepsat do tvaru:

$$E_1(x) = f(x) - f(a) - f'(x)(x-a)$$

Zjevně platí, že $f(x) - f(a) = [f(t)]_a^x = \int_a^x f'(t)dt$ podobně $x - a = [t]_a^x = \int_a^x dt$ a tedy

$$E_1(x) = \int_a^x f'(t)dt - f'(a)\int_a^x dt = \int_a^x f'(t)dt - \int_a^x f'(a)dt =$$
$$= \int_a^x [f'(t) - f'(a)]dt$$

Označením $u = f'(t) - f'(a)$ a $v = t - x$, kde $dv/dt = 1$ a $du/dt = f''(t)$ a použitím metody per partes dostáváme vztah

$$\int_a^x [f'(t) - f'(a)]dt = [(f'(t) - f'(a))(t - x)]_a^x - \int_a^x (t - x)f''(t)dt$$
$$E_1(x) = (f'(x) - f'(a))(x - x) - (f'(a) - f'(a))(a - x) - \int_a^x (t - x)f''(t)dt$$
$$E_1(x) = \int_a^x (x - t)f''(t)dt$$

□

Věta 3. Předpokládejme, že funkce f má spojitou derivaci řádu $n+1$ v určitém okolí bodu a. Pak pro každé x v tomto okolím platí

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x - a)^k + E_n(x) \tag{5.12}$$

kde

$$E_n(x) = \frac{1}{n!}\int_a^x (x - t)^n f^{(n+1)}(t)dt \tag{5.13}$$

Důkaz. Důkaz provedeme pomocí indukce. Pro $n = 1$ jsme to již dokázali v předchozím teorému. Nyní předpokládejme, že to platí pro n. Je zřejmé, že platí

$$E_{n+1}(x) = E_n(x) - \frac{f^{(n+1)}(a)}{(n+1)!}(x - a)^{n+1} \tag{5.14}$$

Nyní použijeme integrální tvar $E_n(x)$ a využijeme toho, že

$$\int_a^x (x - t)^n dt = \left[\frac{(x - t)^{n+1}}{n + 1}\right]_a^x = \frac{(x - x)^{n+1}}{n + 1} - \frac{(x - a)^{n+1}}{n + 1} =$$
$$= -\frac{(x - a)^{n+1}}{n + 1} \tag{5.15}$$

Dosazením předpokladu (5.13) a vzorce (5.15) do vzorce (5.14) dostáváme

$$E_{n+1} = \frac{1}{n!}\int_a^x (x - t)^n f^{(n+1)}(t)dt - \frac{f^{(n+1)}(a)}{n!}\int_a^x -(x - t)^n dt =$$
$$= \frac{1}{n!}\int_a^x -(x - t)^n [f^{(n+1)}(t) - f^{(n+1)}(a)]dt \tag{5.16}$$

Podobně jako v předchozím důkazu zavedeme proměnné

$$u = f^{(n+1)}(t) - f^{(n+1)}(a) \qquad \frac{du}{dt} = f^{(n+2)}(t)$$
$$a = -\frac{(x - t)^{n+1}}{n + 1} \qquad \frac{dv}{dt} = -(x - t)^n$$

a integrujeme pomocí metody per partes

$$E_{n+1}(x) = \frac{1}{n!}\left([uv]_a^x - \int_a^x -\frac{(x-t)^{n+1}}{n+1} f^{(n+2)}(t)dt\right) =$$
$$= \frac{1}{n!}\int_a^x \frac{(x-t)^{n+1}}{n+1} f^{(n+2)}(t)dt =$$
$$= \frac{1}{(n+1)!}\int_a^x (x-t)^{n+1} f^{(n+2)}(t)dt \tag{5.17}$$

□

5.4 Odhad velikosti rezidua $E_n(x)$

Teorém 2. *Pokud pro všechna t v intervalu obsahujícím a splňuje $(n+1)$-ní derivace funkce f nerovnost*

$$m \le f^{(n+1)}(t) \le M \tag{5.18}$$

pak pro všechna x v tomto intervalu platí následující odhad

$$m\frac{(x-a)^{n+1}}{(n+1)!} \le E_n(x) \le M\frac{(x-a)^{n+1}}{(n+1)!} \qquad \text{pro } x > a \tag{5.19}$$

a

$$m\frac{(a-x)^{n+1}}{(n+1)!} \le (-1)^{n+1}E_n(x) \le M\frac{(a-x)^{n+1}}{(n+1)!} \qquad \text{pro } x < a \tag{5.20}$$

Důkaz. Uvažujme nejprve $x > a$. Pak integrál ve výrazu $E_n(x)$ probíhá na intervalu $[a, x]$. Pro každé t na tomto intervalu platí $(x-t)^n \ge 0$. Z nerovnosti (5.18) vyplývá, že

$$m\frac{(x-t)^n}{n!} \le \frac{(x-t)^n}{n!} f^{(n+1)}(t) \le M\frac{(x-t)^n}{n!} \tag{5.21}$$

Integrací od a do x pak dostáváme

$$\frac{m}{n!}\int_a^x (x-t)^n dt \le E_n(x) \le \frac{M}{n!}\int_a^x (x-t)^n dt \tag{5.22}$$

a substitucí $u = x - t$ a $du = -dt$ dostáváme

$$\int_a^x (x-t)^n dt = \int_x^a (x-t)^n(-1)dt = \int_{u(x)}^{u(a)} u^n du =$$
$$= \int_0^{x-a} u^n du = \frac{(x-a)^{n+1}}{n+1}$$

a tudíž jsme výraz (5.22) redukovali na výraz (5.19).

Pro $x < a$ budeme postupovat obdobně. Integrál probíhá na intervalu $[x, a]$. Pro každé t na tomto intervalu platí $t \ge x$, a tedy $(-1)^n(x-t)^n = (t-x)^n \ge 0$. Nyní můžeme nerovnost (5.18) přenásobit nezáporným zlomkem $(-1)^n(x-t)^n/n!$

$$m\frac{(-1)^n(x-t)^n}{n!} \le \frac{(-1)^n(x-t)^n}{n!} f^{(n+1)}(t) \le M\frac{(-1)^n(x-t)^n}{n!}$$
$$m\frac{(t-x)^n}{n!} \le \frac{(-1)^n(x-t)^n}{n!} f^{(n+1)}(t) \le M\frac{(t-x)^n}{n!}$$

tento výraz integrujeme od x do a

$$\frac{m}{n!}\int_x^a (t-x)^n dt \leq (-1)^n E_n(x) \leq \frac{M}{n!}\int_x^a (t-x)^n dt \tag{5.23}$$

Substitucí $u = t - x$ a $du = dt$ dostáváme

$$\int_x^a (t-a)^n dt = \int_0^{a-x} u^n du = \frac{(a-x)^{n+1}}{n+1}$$

a tudíž jsme výraz (5.23) redukovali na výraz (5.20), kde $(-1)^{n+1}$ dostaneme z $(-1)^n$ záměnou mezí integrace tak, aby jsme získali výraz $E_n(x)$. □

5.5 Konvergence Taylorových řad

Nekonečnou Taylorovu řadu z rovnice (5.8) můžeme přepsat do tvaru

$$f(x) = \sum_{k=0}^{\infty} a_k (x - t_0)^k \tag{5.24}$$

Tato rovnice představuje mocninnou řadu. Má-li řada konvergovat, musí platit následující pravidlo pro konvergenci řad

$$\lim_{n\to\infty} \frac{|a_{n+1}(x-t_0)^{n+1}|}{|a_n(x-t_0)^n|} < 1$$

Pokud tato limita existuje, lze pravidlo přepsat do tvaru

$$|x - t_0| < \lim_{n\to\infty} \left|\frac{c_n}{c_{n+1}}\right| \tag{5.25}$$

Tento vztah definuje poloměr konvergence, na kterém řada konverguje.

Obecně ovšem limita v rovnici (5.25) nemusí existovat. Pak je třeba brát v úvahu vztah (5.12). Taylorova řada konverguje, právě když hodnota $E_n(x) \to 0$ pro $n \to \infty$. Úpravou tvaru rezidua můžeme odvodit následující větu.

Věta 4. Nechť funkce f je nekonečně derivovatelná na otevřeném intervalu $I = (t_0 - r, t_0 + r)$ a nechť existuje konstanta A taková, že platí

$$|f^{(n)}(x)| \leq A^n, \text{ pro } \forall n \in \mathbb{N} \text{ a } \forall x \in I \tag{5.26}$$

pak Taylorova řada generovaná funkcí f v bodě t_0 konverguje k hodnotě $f(x)$ pro všechna $x \in I$.

Důkaz. Výraz pro $E_n(x)$

$$E_n(x) = \frac{1}{n!}\int_{t_0}^{x} (x - t_0)^n f^{(n+1)}(t) dt \tag{5.27}$$

si můžeme pomocí substituce

$$t = x + (t_0 - x)u \qquad dt = -(x - t_0)du$$

upravit do tvaru

$$E_n(x) = \frac{(x-a)^{n+1}}{n!} \int_0^1 u^n f^{(n+1)}[x + (t_0 - x)u]du \tag{5.28}$$

kde u nabývá hodnot 0 až 1 když t nabývá hodnot t_0 až x.

Použitím nerovnosti (5.26) a upraveného vztahu pro $E_n(x)$ můžeme psát

$$0 \leq |E_n(x)| \leq \frac{|x-t_0|^{n+1}}{n!} A^{n+1} \int_0^1 u^n du = \frac{|x-t_0|^{n+1} A^{n+1}}{(n+1)!} = \frac{B^{n+1}}{(n+1)!} \tag{5.29}$$

kde $B = A|x - t_0|$. Nicméně výraz $B^n/n!$ jde pro libovolné B k nule při $n \to \infty$ a tak i $E_n(x) \to 0$ pro každé $x \in I$. □

Obecně ovšem nelze předpokládat, že Taylorův polynom generovaný funkcí f konverguje k $f(x)$ pro každé x. Vhodným protipříkladem tohoto tvrzení je funkce

$$f(x) = e^{-\frac{1}{x^2}} \tag{5.30}$$

Tato funkce je nekonečně derivovatelná, v bodě 0 má limitu a tudíž hodnota $f(0) = 0$, ale Taylorův polynom generovaný touto funkcí bude roven 0 pro všechna x. Taylorův polynom funkce f v bodě 0 se tedy shoduje s funkcí f pouze v bodě 0.

Tato skutečnost je způsobena tím, že jedna z funkcí derivačního uzávěru funkce f (konkrétně $1/x$) není na intervalu, pro který Taylorův polynom generujeme, omezená. Požadujeme-li tedy, aby se Taylorův polynom s generovanou funkcí f shodoval ve všech bodech daného intervalu, je nutné, aby všechny funkce derivačního uzávěru této funkce byly na zvoleném intervalu omezené.

5.6 Aplikace Taylorova teorému pro řešení soustavy diferenciálních rovnic

Mějme diferenciální rovnici ve tvaru

$$y' = \phi(y, t) \qquad\qquad y(a) = y_0 \tag{5.31}$$

kde ϕ je polynom. Z kapitoly 3 víme, že každou soustavu diferenciálních rovnic, která má na pravé straně pouze polynomy a funkce s konečným derivačním uzávěrem, lze převést na soustavu diferenciálních rovnic, která má na pravé straně pouze polynomy. Dále se tedy budeme zabývat pouze těmito tvary diferenciálních rovnic a soustav diferenciálních rovnic.

Taylorův polynom $T_n y$ můžeme vyjádřit jako

$$T_n y(x; a) = y(a) + y'(a)(x-a) + \frac{y''(a)}{2}(x-a)^2 + \cdots \frac{y^{(n)}(a)}{n!}(x-a)^n \tag{5.32}$$

Zderivujeme-li obě strany této rovnice, dostáváme

$$(T_n y(x; a))' = y'(a) + y''(a)(x-a) + \frac{y'''(a)}{2}(x-a)^2 + \cdots \frac{y^{(n)}(a)}{(n-1)!}(x-a)^{n-1} \tag{5.33}$$

a tedy

$$(T_n y)' = T_{n-1} y' = T_{n-1}\phi \tag{5.34}$$

Z předchozích vztahů je také patrné, že

$$T_n y(x;a) = \int_a^x T_{n-1} y'(t;a)dt + y(a) \tag{5.35}$$

Zaveďme nyní diferenciální operátor B takto:

$$BT_n y = (T_n y)' = T_{n-1} y'$$
$$B^{-1} T_n y' = \int_a^x T_n y'(t)dt + y(a) = T_{n+1} y$$

Pro získání koeficientů polynomu $T_n y$ tedy postačí spočítat koeficienty polynomu $T_{n-1} y'$ a doplnit je o $y(a)$. Dále můžeme využít toho, že funkce, které mají konečný derivační uzávěr (a tudíž je lze převést na konečnou soustavu diferenciálních rovnic, na jejichž pravé straně se vyskytují pouze polynomy), mají také na svém definičním oboru definované všechny derivace.

Věta 5. Pokud má funkce f na intervalu $I \subseteq D$ definován derivační uzávěr F, pak má funkce f na tomto intervalu definovány všechny derivace. Tyto derivace jsou v polynomiálním tvaru $p(t, f_1, \cdots, f_n)$, kde $f_i \in F$.

Důkaz. Z definice množiny F víme, že funkce f má definovánu první derivaci a to konkrétně $f'(t) = p(t, f_1(t), \cdots, f_n(t))$, kde $f_i \in F$.

Nechť je definována i-tá derivace $f^{(i)} = p(f_1, \cdots, f_m)$. Pak platí, že

$$f^{(i+1)} = \frac{dp(t, f_1, \cdots, f_m)}{dt} = \sum_{i=1}^{m} \frac{\partial p(t, f_1, \cdots, f_m)}{\partial f_i} f_i' + \frac{\partial p(t, f_1, \cdots, f_m)}{\partial t} \tag{5.36}$$

Je zřejmé, že $\frac{\partial p(f_1, \cdots, f_m)}{\partial f_i}$ a $\frac{\partial p(t, f_1, \cdots, f_m)}{\partial t}$ jsou polynomy. Z definice množiny F víme, že f_i' je také polynom a tedy

$$f^{(i+1)} = p(t, f_l, \cdots, f_k) \tag{5.37}$$

□

Máme-li tedy soustavu diferenciálních rovnic ve tvaru

$$\begin{aligned} y_1' &= \phi_1(y_1, \cdots, y_m) & y_1(t_0) &= a_{1,0} \\ y_2' &= \phi_2(y_1, \cdots, y_m) & y_2(t_0) &= a_{2,0} \\ &\vdots & & \\ y_m' &= \phi_m(y_1, \cdots, y_m) & y_m(t_0) &= a_{n,0} \end{aligned}$$

kde ϕ_i je polynom, můžeme sestavit m Taylorových polynomů, které se s funkcemi y_i budou shodovat na určitém okolí bodu t_0.

Zaveďme tedy polynomy:

$$P_1 = B^{-1} T_{n_1} \phi_1$$
$$P_2 = B^{-1} T_{n_2} \phi_2$$
$$\vdots$$
$$P_m = B^{-1} T_{n_m} \phi_m$$

Hodnoty jednotlivých proměnných pak budou:

$$\begin{aligned} y_1(t) &= P_1(t) + E_{n_1}(t) \\ y_2(t) &= P_2(t) + E_{n_2}(t) \\ &\vdots \\ y_m(t) &= P_m(t) + E_{n_m}(t) \end{aligned}$$

Jelikož všechny funkce y_i mají konečný derivační uzávěr (podmnožina množiny $\{y_1, \cdots, y_m\}$), pak z Věty 5 víme, že mají také definovány všechny derivace a tudíž můžeme sestavit Taylorův polynom s libovolnou přesností.

5.7 Ukončovací podmínka

Správné určení řádu polynomu je nejobtíženější část celého výpočtu. Je nutné optimalizovat dva protichůdné faktory a to časovou a paměťovou náročnost výpočtu a dosaženou přesnost. Čím větší je řád polynomu, tím menší je chyba výpočtu, ale zároveň se začne projevovat numerická nestabilita. V matematických výpočtech je většinou upřednostňována přesnost dosažených výsledků před rychlostí výpočtu. Pro využití v praxi může být v některých případech důležitější rychlost výpočtů. Implementace programu proto umožňuje zvolit maximální chybu výpočtu pro mezní bod požadovaného intervalu řešení (čím menší chyba, tím vyšší přesnost a nižší rychlost).

Pokud nám stačí pouze numerické řešení soustavy rovnic (diskrétní výsledky pro množinu bodů t), je možné využít opakované spouštění výpočtu tak, že nejprve vypočítáme P_i v bodě t_0 a získáme hodnotu $y_i(t_1)$, pak vypočítáme P_i v bodě t_1 a získáme hodnotu $y_i(t_2)$, až postupně dostaneme hodnotu $y_i(t_k)$. Tímto postupem je okolí, na kterém musí být aproximace přesná, menší a tedy i řád polynomu může být menší. Je-li požadováno analytické řešení pro velký interval, je zapotřebí velkého počtu členů Taylorova rozvoje.

V obou případech je ovšem potřeba stanovit ukončovací podmínku pro výpočet řádů polynomu. Chybu E_{n_i} je obtížné stanovit, jelikož hodnota $f^{(n_i+1)}(t)$ není známa (závisí na neznámých funkcích $y_1, \cdots, y_m$). Pro ukončení používáme empiricky zjištěné vlastnosti, že pokud součet tří po sobě jdoucích členů

$$\frac{y_i^{(k)}(a)}{k!}(t_{max} - a)^k \tag{5.38}$$

je menší než požadovaná přesnost (pro t_{max} rovno konci požadovaného intervalu), pak je reziduum $E_i(t_{max})$ také menší než požadovaná přesnost.

Tato vlastnost nicméně není podložena žádnou matematickou teorií a je třeba ji upřesnit. Matematické zdůvodnění této podmínky přesahuje rámec této práce. V implementaci výpočtu koeficientů Taylorovy řady práce tedy využívá pouze empirického zjištění.

5.8 Algoritmus výpočtu koeficientů Taylorova polynomu

Algoritmus 3 uvádí základní algoritmus pro výpočet koeficientů Taylorovy řadu. Za koeficienty Taylorovy řady považujeme hodnoty $y_i^{(k)}(t_0)$ pro $i \in \{1, \cdots, n\}$ a $k \in \{0, \cdots, maxorder\}$.

Z těchto koeficientů lze určit hodnotu aproximované funkce podle předpisu:

$$y_i(t) = \sum_{k=0}^{maxorder} \frac{y_i^{(k)}(t_0)}{k!}(t - t_0)^k \tag{5.39}$$

Algoritmus 3: Algoritmus výpočtu koeficientů Taylorovy řady pro zadanou počáteční úlohu

Vstup: Množina diferenciálních rovnic Ψ v polynomiálním tvaru, funkce p mapující proměnné na jejich počáteční podmínky a dvojice t_0, t_{max}
Výstup: Uspořádaný seznam vektorů C představující koeficienty Taylorova polynomu pro každou neznámou funkci

```
begin
    C ← ();
    foreach „y' = ψ“ ∈ Ψ do
        a ← vyhodnocení ψ v bodě t_0;
        v ← (p(y), a);
        φ ← ψ;
        while Není splněna ukončovací podmínka v bodě t_max do
            φ ← derivace φ pomocí vztahu (3.3) tak, že každou derivaci proměnné
            nahradíme jejím předpisem z Ψ;
            a ← vyhodnocení φ v bodě t_0;
            v ← (v(:), a);
        end
        C(y) ← v;
    end
end
```

Tento algoritmus není optimální, jelikož mnoho výpočtů se provádí opakovaně pro stejná čísla. V kapitole 6 se budeme tímto algoritmem blíže zabývat a navrhneme možná urychlení.

Kapitola 6

Optimalizace výpočtu koeficientů

V této kapitole budeme zkoumat složitost algoritmu 3 a navrhneme různé postupy jak tuto složitost snížit.

Je patrné, že složitost výpočtu bude velmi záviset na tvaru rovnice v množině Ψ. Pro usnadnění výpočtů a odhadů složitosti budeme předpokládat, že Ψ obsahuje n rovnic, které mají na pravé straně polynomy s m součty, každý s k různých násobků y_i. Prozatím budeme uvažovat pouze polynomy, ve kterých se vyskytují proměnné s mocninou 1. Polynomy s vyššími mocninami lze do tohoto tvaru převést transformací popsanou v kapitole 3.

Analyzujme nyní počet členů v proměnné ϕ. Na počátku zde bude m součtů s k násobky a tedy $m \cdot k$ nutných operací pro vyhodnocení. Zderivujeme-li ϕ, dostaneme pro každý násobek k součtů s k násobky. Celkově tedy dostáváme $m \cdot k^2$ proměnných. Po nahrazení derivací proměnných dle předpisu definovaného soustavou diferenciálních rovnic dostaneme místo každé derivované proměnné $m \cdot k$ proměnných a celkově tedy po tomto kroku bude ϕ obsahovat $m \cdot k^2 \cdot m \cdot k = m^2 \cdot k^3$ proměnných, z čehož je $m^2 \cdot k$ součtů po k^2 násobcích.

Provedením další derivace získáme $m^2 \cdot k^3$ součtů o k^2 násobcích. Po dosazení derivovaných proměnných dostáváme $m^3 \cdot k^4$ součtů o k^3 násobcích. Tímto postupně dostaneme pro i-tou derivaci $m^i \cdot k^{\sum_{j=1}^{i-1} j}$ součtů o k^i násobcích.

Jako příklad si můžeme uvést soustavu diferenciálních rovnic ve tvaru

$$y_1' = \phi(y_1, y_2) = y_1 y_2 \tag{6.1}$$
$$y_2' = y_1 + y_2 \tag{6.2}$$

Postupným derivováním výrazu $\phi(y_1, y_2)$ dostáváme

$$\begin{aligned}
\phi' &= y_1' y_2 + y_1 y_2' = y_1 y_2^2 + y_1(y_1 + y_2) \\
\phi'' &= y_1' y_2^2 + 2 y_1 y_2 y_2' + y_1'(y_1 + y_2) + y_1(y_1' + y_2') = \\
&= y_1 y_2^3 + 2 y_1 y_2 (y_1 + y_2) + y_1 y_2 (y_1 + y_2) + y_1(y_1 y_2 + y_1 + y_2)
\end{aligned}$$

6.1 Souběžný výpočet všech Taylorových polynomů

Při bližším pohledu na nárůst jednotlivých sčítanců a násobků zjistíme, že velký vliv má dosazování výrazu za zderivovanou proměnnou. Vyhodnocení tohoto výrazu v bodě t_0 vede vždy ke stejnému výsledku. Je tedy mnohem efektivnější si tyto mezivýsledky ukládat do paměti a místo dosazování výrazů používat již vypočtené hodnoty. Tímto způsobem nebude narůstat velikost jednotlivých sčítanců a tedy pro i-tou derivaci dostáváme $m \cdot k^{i-1}$ součtů

o k násobcích. Je patrné, že počet součtů ve většině případů mnohonásobně převyšuje počet proměnných jednotlivých sčítanců. Pro úsporu paměti je tedy vhodné sčítance se stejnými proměnnými sečíst. Nevýhodou je, že při každém slučování musíme provést přibližně $(m \cdot k^{i-1})^2$ porovnání.

Budeme-li opět derivovat ϕ tímto způsobem dostáváme

$$\begin{aligned}\phi' &= y_1'' = y_1' y_2 + y_1 y_2' \\ \phi'' &= y_1''' = y_1'' y_2 + 2y_1' y_2' + y_1 y_2'' \\ \phi''' &= y_1^{(4)} = y_1''' y_2 + 3y_1'' y_2' + 3y_1' y_2'' + y_1 y_2'''\end{aligned}$$

Algoritmus 4: Optimalizovaný algoritmus výpočtu koeficientů Taylorovy řady pro zadanou počáteční úlohu

Vstup: Množina diferenciálních rovnic Ψ v polynomiálním tvaru, funkce p mapující proměnné na jejich počáteční podmínky a dvojice t_0, t_{max}

Výstup: Uspořádaný seznam vektorů C představující koeficienty Taylorova polynomu pro každou neznámou funkci

```
begin
    D ← ();
    C ← ();
    foreach „y' = ψ“ ∈ Ψ do
        a ← vyhodnocení ψ v bodě t_0;
        C(y) ← (p(y), a);
        D(y) ← ψ;
    end
    while Není splněna ukončovací podmínka v bodě t_max pro všechny funkce y_i do
        foreach „y' = ψ“ ∈ Ψ do
            D(y) ← derivace D(y) pomocí vztahu 3.3 a optimalizace výrazu sloučením
            stejných sčítanců;
            a ← vyhodnocení D(y) v bodě t_0 s využitím hodnot C(y);
            C(y) ← (C(y)(:), a);
        end
    end
end
```

6.2 Pascalovy trojúhelníky

Postupné získávání jednotlivých derivací z derivace předchozí je jak časově, tak paměťově náročný proces. Využijeme-li skutečnosti, že vstupem jsou pouze polynomy, kde se vyskytují pouze operace násobení, dělení, sčítání a odečítání, tak můžeme jednotlivé derivace získávat přímo z původního polynomu. Víme, že derivace součtu/rozdílu je součet/rozdíl derivací a tudíž

$$(y_1 + y_2)^{(i)} = y_1^{(i)} + y_2^{(i)} \tag{6.3}$$

Máme-li výraz $y = y_1 y_2$ pak jeho postupné derivace budou

$$\begin{aligned} y' &= y_1' y_2 + y_1 y_2' \\ y'' &= y_1'' y_2 + 2y_1' y_2' + y_1 y_2'' \\ y''' &= y_1''' y_2 + 3y_1'' y_2' + 3y_1' y_2'' + y_1 y_2''' \end{aligned}$$

Je patrné, že se tento výraz chová podobně jako $(A+B)^n$ a tedy lze využít rozvoje pomocí Pascalova trojúhelníku. Obecně pro n-tou derivaci y můžeme psát vztah:

$$y^{(n)} = \sum_{k=0}^{n} \binom{n}{k} y_1^{(k)} y_2^{(n-k)} \tag{6.4}$$

6.3 Multidimenzionální Pascalovy trojúhelníky

Máme-li obecný výraz $y = y_1 y_2 \cdots y_m$, budou se jednotlivé derivace rozvíjet podle parametrů m-dimenzionálního Pascalova trojúhelníku. Pro vizualizaci libovolného multidimenzionálního Pascalova trojúhelníku jsem vytvořil skript, který je umístěn na stránkách http://taylor.mrvv.info/derivation. Multidimenzionální Pascalův trojúhelník byl popsán v [2]. Tento článek uvádí vzorec

$$(y_1 + \cdots + y_m)^n = \sum_{i_1=0}^{n} \left(\sum_{i_2=0}^{i_1} \cdots \left(\sum_{i_{m-1}=0}^{i_{m-2}} \binom{n}{i_1} \binom{i_1}{i_2} \cdots \binom{i_{m-2}}{i_{m-1}} y_1^{n-i_1} y_2^{i_1-i_2} \cdots y_m^{i_{m-1}} \right) \right) \tag{6.5}$$

Převedeme-li tento vzorec pro účely výpočtu derivací, dostáváme:

$$y^{(n)} = \sum_{i_1=0}^{n} \left(\sum_{i_2=0}^{i_1} \cdots \left(\sum_{i_{m-1}=0}^{i_{m-2}} \binom{n}{i_1} \binom{i_1}{i_2} \cdots \binom{i_{m-2}}{i_{m-1}} y_1^{(n-i_1)} y_2^{(i_1-i_2)} \cdots y_m^{(i_{m-1})} \right) \right) \tag{6.6}$$

Vyjádříme-li si násobek binominálních členů, dostáváme:

$$\begin{aligned} \binom{n}{i_1} \binom{i_1}{i_2} \cdots \binom{i_{m-2}}{i_{m-1}} &= \frac{n!}{(n-i_1)!i_1!} \frac{i_1!}{(i_1-i_2)!i_2!} \cdots \frac{i_{m-2}!}{(i_{m-2}-i_{m-1})!i_{m-1}!} = \\ &= \frac{n!}{(n-i_1)!(i_1-i_2)! \cdots i_{m-1}!} \end{aligned}$$

Faktoriály ve jmenovateli výsledného zlomku jsou shodné s řády derivací jednotlivých proměnných. Provedeme-li součet těchto řádů, dostáváme:

$$(n-i_1) + (i_1-i_2) + \cdots + (i_{m-2}-i_{m-1}) + i_{m-1} = n \tag{6.7}$$

Pro všechna i_j zároveň platí, že

$$i_1 \geq i_2 \geq \cdots \geq i_{m-1} \tag{6.8}$$

Všechna i_j nabývají hodnoty od 0 do n. Vzhledem k nerovnostem mezi koeficienty i všechny jejich rozdíly $(i_{j-1} - i_j)$ nabývají hodnot od 0 do n.

Vzhledem k těmto skutečnostem můžeme vzorec 6.6 přepsat do tvaru:

$$y^{(n)} = \sum_{k_1,\cdots,k_m \in \mathbb{N}^+} \frac{n!}{\prod_{j=1}^{m} k_j!} \prod_{j=1}^{m} y_j^{(k_j)}, kde\ \forall k_1, \cdots, k_m \in \mathbb{N}^+ : \sum_{j=1}^{m} k_j = n \tag{6.9}$$

Označme $\mathbf{k} = (k_1, \cdots, k_m)$. Vektor $\mathbf{k}$ lze generovat pomocí funkce getStartVector a getNextVector (viz níže).

Funkce `getStartVector` vrací první vektor **k**

Vstup: Počet členů m a řád derivace n
Výstup: Vektor **k**
begin
    k $\leftarrow$ $zeros(1, m)$;
    k(1) $\leftarrow$ n;
end

Funkce `getNextVector(`**k**$, m, n$`)` vrací následující vektor **k**

Vstup: Počet členů m, řád derivace n a předchozí vektor **k**
Výstup: Aktualizovaný vektor **k** a logická hodnota true/false, pokud další vektor existuje (true), nebo se již prošly všechny hodnoty (false)
begin
    if *k(m) = n* **then**
        return false;
    end
    pozLast $\leftarrow$ pozici poslední nenulové hodnoty ve vektoru **k**;
    if *pozLast != m* **then**
        k(pozLast) $\leftarrow$ **k**(pozLast) - 1;
        k(pozLast + 1) $\leftarrow$ **k**(pozLast + 1) + 1;
    else
        pozNext $\leftarrow$ pozice předposlední nenulové hodnoty ve vektoru **k**;
        if *pozNext + 1 = pozLast* **then**
            k(pozNext) $\leftarrow$ **k**(pozNext) - 1;
            k(pozLast) $\leftarrow$ **k**(pozLast) + 1;
        else
            k(pozNext + 1) $\leftarrow$ **k**(pozLast) + 1;
            k(pozNext) $\leftarrow$ **k**(pozNext) - 1;
            k(pozLast) $\leftarrow$ 0 ;
        end
    end
    return true;
end

6.4 Algoritmus využívající multidimenzionální Pascalovy trojúhelníky

Algoritmus 4 může být upraven tak, aby pro získávání hodnot vyšších derivací používal multidimenzionální Pascalovy trojúhelníky, popsané v předešlé kapitole. Tento algoritmus zde uvádíme jako Algoritmus 8. Tento algoritmus využívá funkci evalDerivation, která je popsána níže.

Počet různých vektorů **k** pro m násobků a i-tou derivaci, které splňují podmínku

$$\sum_{j=1}^{m} k_j = i$$

je

$$\binom{i+m-1}{m-1} \tag{6.10}$$

Funkce `evalDerivation`(ϕ, n, C, t_0) vypočítá hodnotu derivace ϕ v t_0

Vstup: Násobek ϕ ve tvaru $y_1 \cdots y_m$, řád derivace n, hodnoty všech derivací $y_1, \cdots, y_m$ nižšího řádu než n ve vektoru C a bod t_0
Výstup: Hodnota s představující n-tou derivaci ϕ v bodě t_0

```
begin
    cont ← true;
    k ← getStartVector();
    s ← 0;
    while cont do
        v ← n!;
        foreach k_i ∈ k do
            v ← v C(y_i)(k_i) / k_i! ;
        end
        s ← s + v;
        cont ← getNextVector(k, m, n);
    end
end
```

Algoritmus 8: Algoritmus výpočtu koeficientů Taylorovy řady pro zadanou počáteční úlohu s využitím multidimenzionálních Pascalových trojúhelníků

Vstup: Množina diferenciálních rovnic Ψ v polynomiálním tvaru, funkce p mapující proměnné na jejich počáteční podmínky a dvojice t_0, t_{max}
Výstup: Uspořádaný seznam vektorů C představující koeficienty Taylorova polynomu pro každou neznámou funkci

```
begin
    C ← ();
    foreach „y' = ψ“ ∈ Ψ do
        a ← vyhodnocení ψ v bodě t_0;
        C(y) ← (p(y), a);
    end
    n ← 2;
    while Není splněna ukončovací podmínka v bodě t_max pro všechny funkce y_i do
        foreach „y' = ψ“ ∈ Ψ do
            a ← vyhodnocení n-té derivace ψ v bodě t_0 s využitím hodnot C(y) a
            funkce evalDerivation pro získání hodnot derivací násobků;
            C(y) ← (C(y)(:), a);
        end
        n ← n + 1;
    end
end
```

Výpočet koeficientů Taylorova polynomu pomocí Algoritmu 8 tedy provede

$$n \cdot \sum_{i=1}^{o} \binom{i+m-1}{m-1} \tag{6.11}$$

součtů, kde n je počet rovnic v Ψ, o je nejnižší řád polynomu, který již splňuje ukončovací podmínku, m je počet násobků v každém sčítanci a

$$m \cdot n \cdot \sum_{i=1}^{o} \binom{i+m-1}{m-1} \tag{6.12}$$

je počet násobení.

6.5 Derivace mocnin

Jelikož vstupní diferenciální rovnice jsou v polynomiálním tvaru, jednotlivé sčítance nejsou pouze ve tvaru $y_1 y_2 \cdots y_m$, ale jednotlivé proměnné mohou být umocněny celočíselným koeficientem. Je tedy třeba vytvořit algoritmus výpočtu n-té derivace výrazu y^i, kde $i \in \mathbb{Z}$.

Zaměřme se nejprve na výraz $x = y^i$, kde $i \in \mathbb{N}$. Tento výraz lze přepsat do tvaru $yy \cdots y$ a použít algoritmus odvozený v kapitole 6.4. Je ale patrné, že tento algoritmus by zcela zbytečně opakovaně počítal příliš mnoho shodných členů. Pro libovolný vektor $\mathbf{k}$ definovaný v kapitole 6.3 platí, že

$$\frac{n!}{\prod_{j=1}^{m} \mathbf{k}(j)!} \prod_{j=1}^{m} y_j^{(\mathbf{k}(j))} = \frac{n!}{\prod_{j=1}^{m} p(\mathbf{k})(j)!} \prod_{j=1}^{m} y_j^{(p(\mathbf{k})(j))} \tag{6.13}$$

kde $p(\mathbf{k})$ je libovolná permutace vektoru $\mathbf{k}$. Rovnici (6.9) můžeme upravit do tvaru

$$x^{(n)} = \sum_{\mathbf{k} \in \mathbf{V}(i,n)} c(\mathbf{k}) \frac{n!}{\prod_{j=1}^{i} \mathbf{k}(j)!} \prod_{j=1}^{i} y^{(\mathbf{k}(j))} \tag{6.14}$$

kde $\mathbf{V}(i,n)$ je množina všech vektorů z prostoru $\mathbb{N}^{+i}$, takových, že $\forall \mathbf{k} \in \mathbf{V}(i,n)$ platí, že $\sum_{j=1}^{i} \mathbf{k}(j) = n$ a pro libovolné dva vektory $\mathbf{k}, \mathbf{k}' \in \mathbf{V}(i,n)$ neexistuje permutace p taková, aby platilo $\mathbf{k} = p(\mathbf{k}')$. $c : \mathbf{V}(i,n) \to \mathbb{N}$ je funkce, která každému vektoru $\mathbf{k} \in \mathbf{V}(i,n)$ přiřazuje počet opakování, které by nastalo podle rovnice (6.9), ale které se při tomto způsobu výpočtu neprovádí. $c(\mathbf{k})$ musí tedy zahrnovat počet opakování jednotlivých číselných kombinací, ale nesmí zahrnovat opakování, která již jsou vyjádřena koeficientem v původní rovnici. Toho dosáhneme použitím následujících vztahů.

Pro účely vyjádření funkce c je vhodné zavést další funkci $s : \mathbf{V}(i,n) \to \mathbb{N}^+$, která vrací počet nenulových složek vektoru $\mathbf{k}$. Funkce s může být zavedena takto:

$$s(\mathbf{k}) = \sum_{k \in \mathbf{k}} (1 - \delta_{k,0}) \tag{6.15}$$

kde δ je Kroneckerova funkce definovaná předpisem

$$\delta_{i,j} = \begin{cases} 1, & \text{pro} i = j \\ 0, & \text{pro} i \neq j \end{cases} \tag{6.16}$$

Funkci c nyní můžeme definovat vztahem:

$$c(\mathbf{k}) = \frac{i!}{(i - s(\mathbf{k}))! \prod_{j=1}^{\max(\mathbf{k})} \left(\sum_{\forall k \in \mathbf{k}} \delta_{k,j}\right)!} \tag{6.17}$$

Rovnici (6.9) tedy můžeme zapsat ve tvaru

$$x^{(n)} = \sum_{\mathbf{k} \in \mathbf{V}(i,n)} \frac{i!}{(i - s(\mathbf{k}))! \prod_{j=1}^{\max(\mathbf{k})} \left(\sum_{\forall k \in \mathbf{k}} \delta_{k,j}\right)!} \frac{n!}{\prod_{j=1}^{s(\mathbf{k})} \mathbf{k}(j)!} y^{i - s(\mathbf{k})} \prod_{j=1}^{s(\mathbf{k})} y^{(\mathbf{k}(j))} \tag{6.18}$$

Nyní zbývá vyšetřit případ $x = y^{-i}$, kde $i \in \mathbb{N}$. Vyjádříme-li si jednotlivé derivace, zjistíme, že se oproti vztahům (6.18) liší pouze v prvních derivacích y. Ostatní derivace jsou již stejné. Vliv prvních derivací v $c(\mathbf{k})$ zachycuje vztah

$$\frac{i!}{(i - s(\mathbf{k}))!}$$

Složitějším případem je derivování záporných mocnin. Derivací záporných mocnin získáváme vztah

$$(-1)^{s(\mathbf{k})} \frac{(i + s(\mathbf{k}) - 1)!}{(i - 1)!}$$

Upravením vzorce (6.18) pro záporné mocniny dostáváme

$$x^{(n)} = \sum_{\mathbf{k} \in \mathbf{V}(i,n)} (-1)^{s(\mathbf{k})} \frac{(i + s(\mathbf{k}) - 1)!}{(i - 1)! \prod_{j=1}^{\max(\mathbf{k})} \left(\sum_{\forall k \in \mathbf{k}} \delta_{k,j}\right)!} \frac{n!}{\prod_{j=1}^{s(\mathbf{k})} \mathbf{k}(j)!} y^{-(i + s(\mathbf{k}))} \prod_{j=1}^{s(\mathbf{k})} y^{(\mathbf{k}(j))} \tag{6.19}$$

Funkci evalDerivation je tedy třeba upravit tak, aby místo výpočtu hodnoty derivace funkce y_i v bodě t_0 provedla výpočet derivace mocniny. V programu je tedy implementována pouze tato upravená funkce evalDerivation2. Tato funkce využívá funkce evalPowerDerivation a getNextPowerVector. Tyto funkce jsou podrobně rozepsány níže.

Rozdíly ve funkci Algoritmu 4 a 8 demonstruje kapitola 8.

Funkce `evalDerivation2`(ϕ, n, C, t_0) vypočítá hodnotu derivace ϕ v t_0

Vstup: Násobek ϕ ve tvaru $y_1^{i_1} \cdots y_m^{i_1}$, řád derivace n, hodnoty všech derivací $y_1, \cdots, y_m$ nižšího řádu než n ve vektoru C a bod t_0
Výstup: Hodnota s představující n-tou derivaci ϕ v bodě t_0
begin
  cont $\leftarrow$ true;
  $k \leftarrow$ getStartVector();
  $s \leftarrow 0$;
  while *cont* **do**
    $v \leftarrow n!$;
    foreach $k_j \in k$ **do**
      $v \leftarrow v \frac{evalPowerDerivation(y_j, i_j, k_j, C, t_0)}{k_i!}$;
    end
    $s \leftarrow s + v$;
    cont $\leftarrow$ getNextVector(k, m, n);
  end
end

Funkce `evalPowerDerivation`(y_j, i, n, C, t_0) vypočítá hodnotu derivace y_j^i v t_0

Vstup: Násobek y_j^i, řád derivace n, hodnoty všech derivací y_j nižšího řádu než n ve vektoru C a bod t_0
Výstup: Hodnota s představující n-tou derivaci y_j^i v bodě t_0
begin
  if $i < 0$ **then**
    return Vyhodnocení rovnice (6.19) s využitím getNextPowerVector.
  else
    return Vyhodnocení rovnice (6.18) s využitím getNextPowerVector.
  end
end

Funkce `getNextPowerVector(`**k**, m, n`)` vrací následující vektor **k** pro výpočet derivací mocnin

Vstup: Počet členů m, řád derivace n a předchozí vektor **k**
Výstup: Aktualizovaný vektor **k** a logická hodnota true/false, pokud další vektor existuje (true), nebo se již prošly všechny hodnoty (false)

```
begin
    if k(m) ≠ 0 then
        return false;
    end
    pozLast ← pozici poslední nenulové hodnoty ve vektoru k;
    if k(pozLast) ≠ 1 then
        k(pozLast) ← k(pozLast) - 1;
        k(pozLast + 1) ← k(pozLast + 1) + 1;
        return true;
    end
    pozNext ← první pozice nalevo od pozLast, která se nerovná 1;
    if pozNext = 1 or k(pozNext) = k(1) then
        pocet ← pozLast - pozNext;
        pozNext ← pozNext + 1; nastav všechny pozice od pozNext do pozLast v k
        na 0;
        while pocet > 0 do
            if k(pozNext-1) < pocet then
                k(pozNext) ← k(pozNext-1);
                pocet ← pocet - k(pozNext);
            else
                k(pozNext) ← pocet; pocet ← 0;
            end
            pozNext ← pozNext + 1;
        end
    else
        k(pozNext) ← k(pozNext) - 1;
        pozLast ← první pozice napravo od pozNext, která má odlišnou hodnotu od
        pozNext;
        k(pozLast) ← k(pozLast) + 1;
    end
    return true;
end
```

Kapitola 7

Implementace

Program je rozdělen na dvě části - matematické jádro a grafické rozhraní. Matematické jádro výpočtu tvoří základ této práce a je implementováno v jazyce C++. Podporuje pouze komunikace pomocí parametrů předávaných v příkazovém řádku. Grafické rozhraní je implementováno v jazyce Java. Využívá knihovnu Swing a umožňuje jednoduchou práci s matematickým jádrem.

7.1 Matematické jádro

Matematické jádro je určeno pro: 1) načítání počáteční úlohy ve formě soustav diferenciálních rovnic prvního řádu s počátečními podmínkami, 2) pro transformaci diferenciálních rovnic do polynomiálního tvaru popsané v kapitole 3 a 3) pro výpočet koeficientů Taylorovy řady popsaný v kapitole 6. Matematické jádro podporuje oba algoritmy výpočtu koeficientů Taylorovy řady, jak Algoritmus 4, který postupně získává jednotlivé derivace a ty pak vyhodnocuje, tak Algoritmus 8, který počítá hodnoty jednotlivých derivací pomocí multidimenzionálních Pascalových trojúhelníků. Výstup požadovaného výpočtu vypisuje na standardní výstup, nebo do zadaného souboru a to buď ve formě prostého textu, nebo ve formě XML. Formát XML je vhodný pro načítání výstupu jinými programy, které s výsledky dále pracují. Formát XML je také načítán grafickým rozhraním Taylor.jar.

Vstupní data musí obsahovat rovnici ve tvaru

```
prom' = expr & cond;
```

kde `prom` označuje název derivované proměnné (proměnné mají syntaxi stejnou jako programovací jazyk C++), `expr` označuje výraz složený z proměnných, číselných konstant, interních a definovaných konstant a funkcí. Výraz může obsahovat operace sčítání, odčítání, násobení, dělení a umocnění s tím, že mocniny mohou být pouze celočíselné. Pokud je zapotřebí mocnin s desetinnými koeficienty, je nutné použít opisný tvar

$$a^x = e^{x \ln(a)}$$

Matematické jádro podporuje dvě interní konstanty E a PI a všechny funkce popsané v tabulce 4.1. `cond` může být výraz jako `expr` s tím rozdílem, že tento výraz musí být vyhodnotitelný při začátku výpočtu. Nesmí tedy obsahovat žádnou proměnnou, jejíž hodnota se při výpočtu počítá. Může obsahovat proměnnou `t` která při vyhodnocení nabývá hodnoty t_0 a nebo uživatelem definovanou konstantu.

Uživatel dále může definovat konstanty ve tvaru

```
prom = expr;
```

Konstanty mohou být použity jak pro definici výrazu v derivační rovnici, tak pro definici počáteční podmínky (pak ale musí být vyhodnotitelné na počátku výpočtu). Konstanty mohou využívat i jiné konstanty. Matematické jádro umí detekovat cyklus v deklaraci konstant a uživatele o této skutečnosti informovat.

Mezery a symboly nového řádku program při načítání dat přeskakuje. Ukončení definice derivace či konstanty se provádí znakem ; (středník).

Na libovolném řádku vstupních dat může být definice systémových proměnných. Tato definice začíná klíčovým slovem `system` následovaným složenými závorkami. Uvnitř závorek pak následuje definice systémových proměnných, oddělených středníkem. Program podporuje tyto systémové proměnné: `tmin`, `tmax`, `order` a `eps`. `tmin` nastavuje hodnotu t_0, pro kterou jsou uvedeny počáteční podmínky diferenciálních rovnic. `eps` nastavuje požadovanou maximální chybu výpočtu. `tmax` nastavuje velikost intervalu ($[t_0, t_{max}]$), na kterém provádíme aproximaci polynomu s maximální chybou `eps`. `order` udává řád generovaných polynomů. Pokud je tato hodnota záporná, řád se stanoví dynamicky tak, aby bylo splněno kritérium pro ukončovací podmínky (definované v kapitole 5.7). Je-li některá systémová proměnná definována vícekrát, použije se poslední použitá definice. Standardně nastavené (implicitní) hodnoty těchto systémových proměnných jsou uvedeny v tabulce 7.1.

Proměnná	Implicitní hodnota
`tmin`	0
`tmax`	1
`order`	-1
`eps`	10^{-10}

Tabulka 7.1: Implicitní hodnoty systémových proměnných

Příklady zápisu počátečních úloh pro program „taylor“ a popis parametrů spouštění jsou uvedeny v Příloze B.1.

Program standardně používá aritmetiku C++ (long, double), ale do budoucna umožňuje i její výměnu za aritmetiku s větší přesností. Toho je dosaženo tím, že čísla jsou reprezentována abstraktní třídou Number, se kterou program pracuje. Díky této vlastnosti může zbytek výpočtů používat obecný typ čísla a nemusí rozlišovat, jestli se jedná o celé číslo, nebo číslo desetinné. To se určuje až při provádění operace. Matematické operace nad instancemi třídy Number jsou koncentrovány na jednom místě zdrojového kódu. Při výměně aritmetiky tedy stačí přepracovat pouze tuto malou část. Tato vlastnost také umožňuje dynamické přetypování čísel ve chvíli, kdy program detekuje možné přetečení způsobené aritmetickou operací.

Pro syntaktickou analýzu vstupních rovnic využívá matematické jádro program Bison (verze 2.4) v kombinaci s programem flex (verze 2.5). Zdrojové kódy a zakompilovaný program jsou přiloženy na CD ve složce taylor. Zakompilovaný program je spustitelný na školním serveru Merlin, nicméně samotnou kompilaci server Merlin neumožňuje. Důvodem je to, že server používá starou verzi programu Bison, ve které se vyskytuje nahlášená chyba, zabraňující překladu zdrojových kódů. Přiložený binární soubor taylor je také spustitelný na většině Linuxových platforem.

7.2 Grafické rozhraní

Grafické rozhraní Taylor.jar bylo vyvinuto pro usnadnění práce s matematickým jádrem programu. Program je psaný v jazyce Java a na přiloženém CD v adresáři GUI jsou jak zdrojové kódy, tak spustitelná verze ve formátu Java Archive.

Program umožňuje v textovém okně zapsat formát vstupních rovnic a všech potřebných konstant, který lze ukládat do textového souboru. Dále lze v menu „Edit/Settings“ zvolit umístění matematického jádra. Jsou podporovány dva typy umístění. Buď uživatel program Taylor.jar spouští v Linuxovém prostředí a má dostupný binární soubor matematického jádra. Pak může využít vlastní výpočetní výkon a nastavit možnost „Use local file“. V tomto případě se výpočty provádí spouštěním jádra na stejném počítači. Druhou možností je využití Remote Procedure Call pomocí formátu XML, který program také podporuje. Tato volba je vhodná, pokud jsme v prostředí Windows, nebo pokud nemáme k dispozici spustitelnou verzi matematického jádra. Všechny požadavky na výpočet se pak pomocí RPC posílají na server, kde je umístěno matematické jádro a kde se také provedou výpočty. V současné době je dostupný pouze jeden server na adrese `http://taylor.mrvv.info/rpc`. Tento server není příliš výkonný a pro složitější rovnice může výpočet trvat značně dlouhou. Je tedy vhodné používat ho pouze pro demonstrační účely. Program Taylor.jar si automaticky ukládá nastavení umístění matematického jádra a tudíž není třeba toto nastavení měnit při každém spuštění.

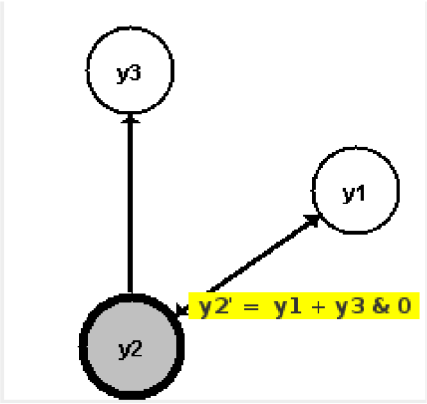
Hlavní funkce jsou dostupné přes menu „Action“. Volba „Transform“ převede vstupní rovnice do polynomiálního tvaru popsaném v kapitole 3 a zobrazí je v textové podobě. Volba „Show relation graph“ zobrazí relační graf závislostí derivací na hodnotách neznámých proměnných. Relační graf umožňuje přemisťování uzlů pro zpřehlednění závislostí a grafické zobrazení tranzitivního uzávěru pro zvolený uzel. Jako příklad je uvedena soustava diferenciálních rovnic

$$y_1' = y_1 \cdot y_2 \qquad y_1(t_0) = 0 \tag{7.1}$$
$$y_2' = y_1 + y_3 \qquad y_2(t_0) = 0 \tag{7.2}$$
$$y_3' = y_3 \qquad y_3(t_0) = 0 \tag{7.3}$$

jejíž výsledný relační graf zobrazen na obrázku 7.1. Z grafu je vidět, že pro výpočet derivací y_2 potřebujeme znát hodnoty derivací y_1 a y_3, zatímco pro výpočet y_3 nepotřebujeme znát hodnoty žádné jiné proměnné. Relační graf slouží pro analýzu velkých soustav diferenciálních rovnic a rozvržení možné paralelizace. Chceme-li řešit velké soustavy diferenciálních rovnic na více procesorech s oddělenými operačními paměťmi, je vhodné sdružovat silně souvislé komponenty relačního grafu na jeden procesor, čímž se minimalizuje meziprocesorová komunikace. Tvoří-li graf jednu silně souvislou komponentu, pak je třeba hledat řez relačního grafu tak, aby přetínal nejmenší počet hran. Každá hrana představuje nutnost sdílení mezivýsledků.

Volba „Show derivations“ zobrazí tvary vyšších derivací jednotlivých proměnných. Řád nejvyšší derivace se v tomto případě nastavuje pomocí systémové proměnné `order`. Volba „Show polynoms“ vygeneruje Taylorovy polynomy pro zadané soustavy diferenciálních rovnic a tyto polynomy textově vypíše. Pro výpočet polynomů se interně používá multidimenzionálních Pascalových trojúhelníků.

Volby „Plot results“ a „Plot results using derivations“ vygenerují Taylorovy polynomy pro zadané diferenciální rovnice a zobrazí grafické rozhraní pro vykreslování těchto polynomů. Pro výpočet polynomů se používá Algoritmus 8 respektive Algoritmus 4. Pro vykreslování výsledných grafů používá program Taylor.jar knihovnu JMathPlot [7].



Obrázek 7.1: Relační graf pro soustavu rovnic (7.1) - (7.3).

Kapitola 8

Výsledky

V této kapitole se budeme zabývat výsledky generovanými vytvořeným programem a jejich srovnáním s výsledky získanými analytickým řešením. Budeme také srovnávat řešení získaná algorimem využívajícím posupné získávání jednotlivých derivací a algoritmem využívající multidimenzionálních Pacsalových trojúhleníků. Vlastnosti a odlišnosti Algoritmu 4 a Algoritmu 8 budeme demonstrovat na zvolených modelových rovnicích. Budeme srovnávat délku polynomů, potřebnou pro dosažení požadované přesnosti, paměťovou a časovou náročnost výpočtů vztahujících se k jednotlivým funkcím, a na závěr zhodnotíme numerickou nestabilitu způsobenou velkým nárůstem velikosti koeficientů Taylorovy řady.

8.1 Polynomiální transformace

V této části si popíšeme princip polynomiální transformace na složitější počáteční úloze a srovnáme jí s dosaženým výsledkem vytvořeného programu.

Mějme počáteční úlohu ve tvaru

$$y_1' = y_{11} + y_{12} \qquad y_1(t_0) = 0 \tag{8.1}$$
$$y_2' = y_{21} + y_{22} \qquad y_2(t_0) = 0 \tag{8.2}$$

kde y_{11} až y_{22} je rovno

$$y_{11} = \frac{1}{\sqrt[3]{y_1^4 + 1}} \tag{8.3}$$
$$y_{12} = t^4 \tag{8.4}$$
$$y_{21} = \arcsin(y_2 + y_1) \tag{8.5}$$
$$y_{22} = e^{-\sin(y_1)} \tag{8.6}$$

Zavedeme-li si nové proměnné

$$y_{111} = y_1^4 + 1 \tag{8.7}$$
$$y_{112} = \sqrt[3]{y_{111}} \tag{8.8}$$

můžeme výraz (8.3) přepsat do tvaru

$$y_{11} = y_{112}^{-1} \tag{8.9}$$

Stejným způsobem můžeme zavést nové výrazy

$$y_{221} = \sin(y_1) \tag{8.10}$$

a výrazy (8.5) a (8.6) upravit do tvaru

$$y_{21} = e^t \tag{8.11}$$
$$y_{22} = e^{-y_{221}} \tag{8.12}$$

Pro nově zavedené výrazy vytvoříme diferenciální rovnice tímto způsobem

$$y'_{111} = 4y_1^3 y'_1 = 4y_1^3(y_{11} + t^4) \qquad y_{111}(t_0) = y_1(t_0)^4 + 1 = 1 \tag{8.13}$$

$$y'_{112} = \frac{1}{3} y_{111} \sqrt[3]{y_{111}} y'_{111} = \\ = \frac{4}{3} y_{111} y_{112} y_1^3 (y_{11} + t^4) \qquad y_{112} = \sqrt[3]{y_{111}(t_0)} = 1 \tag{8.14}$$

$$y'_{11} = -y_{112}^{-2} y'_{112} = \\ = \frac{-4}{3} y_{111} y_{112}^{-1} y_1^3 (y_{11} + t^4) \qquad y_{11}(t_0) = y_{112}^{-1} = 1 \tag{8.15}$$

$$y'_{221} = \cos(y_1) y'_1 = y_{222}(y_{11} + y_{12}) \qquad y_{221}(t_0) = 0 \tag{8.16}$$

$$y'_{222} = -\sin(y_1) y'_1 = y_{221}(y_{11} + y_{12}) \qquad y_{222}(t_0) = 1 \tag{8.17}$$

$$y'_{21} = e^t = y_{21} \qquad y_{21}(t_0) = e^{t_0} \tag{8.18}$$

$$y'_{22} = -y'_{221} e^{-y_{221}} = y_{22} y_{222}(y_{11} + y_{12}) \qquad y_{22}(t_0) = 1 \tag{8.19}$$

Rovnice (8.1) a (8.2) společně s nově zavedenými rovnicemi (8.13) - (8.19) tvoří počáteční úlohu v polynomiálním tvaru, která vede ke stejnému výsledku jako počáteční úloha původní. Program taylor provede transformaci podobným způsobem a získá rovnice

```
y_1'= __v1 + t^4&0;
y_2'= __v2 + __v5&0;
__v1'=-1.33333 * y_1^3 __v1^2 __v7^-1 -1.33333 * t^4 y_1^3 __v1 __v7^-1&1;
__v2'= __v2&1;
__v3'= __v1 __v4 + t^4 __v4&0;
__v4'=-1 * __v1 __v3 -1 * t^4 __v3&1;
__v5'=-1 * __v1 __v4 __v5 -1 * t^4 __v4 __v5&1;
__v7'= y_1^4 +1&1;
```

8.2 Dynamická volba řádu polynomu

V této části se budeme zabývat vlivem délky intervalu (volba t_{max}) a požadované maximální chyby (volba *eps*)na řád polynomu. Čas t_0 je v každém experimentu zvolen jako $t_0 = 0$. Jako modelové funkce pro testování jsme zvolili: funkci $\sin(t)$, která vede na počáteční úlohu ve tvaru

$$y' = cos(t) \qquad y(t_0) = 0 \tag{8.20}$$

funkci e^t, která vede na počáteční úlohu ve tvaru

$$y' = y \qquad y(t_0) = 1 \tag{8.21}$$

funkci $\mathrm{tg}(t)$, která vede na počáteční úlohu ve tvaru

$$y' = y^2 + 1 \qquad y(t_0) = 0 \tag{8.22}$$

a funkci $\sin(t) \cdot \cos(t)$, která vede na počáteční úlohu ve tvaru

$$y' = \cos^2(t) - \sin^2(t) \qquad y(t_0) = 0 \tag{8.23}$$

V tabulce 8.1 a 8.2 jsou uvedené zjištěné hodnoty pro funkce $\sin(t)$ a e^t. Jedná se o jedny z nejjednodušších funkcí pro aproximaci Taylorovým polynomem, jelikož koeficienty těchto polynomů nabývají pouze hodnoty 1 v případě funkce e^t a hodnoty -1, 0, 1 v případě funkce $\sin(t)$. Numerická nestabilita se tedy při výpočtu těchto funkcí neprojevuje. Na obrázku 8.1 a 8.2 jsou graficky znázorněny naměřené hodnoty a trendy zvyšování řádu polynomu v závislosti na délce intervalu a zvolené maximální chybě.

	eps			
`tmax`	1E-1	1E-5	1E-10	1E-20
1	4	11	16	24
5	16	23	31	43
10	28	36	44	60
20	56	64	72	91
30	83	92	100	120
50	136	144	> 150	> 150

Tabulka 8.1: Počet řádů polynomu, nutných pro výpočet funkce $\sin(t)$ se zadanými parametry tmax a eps.

	eps			
`tmax`	1E-1	1E-5	1E-10	1E-20
1	6	11	16	24
5	16	23	31	43
10	29	37	46	60
20	56	65	74	91
30	83	92	102	120
50	137	146	> 150	> 150

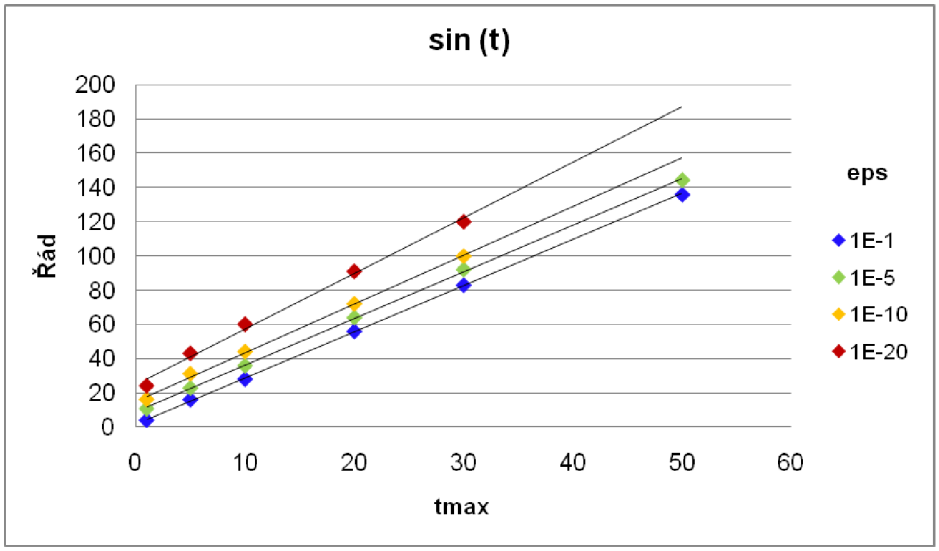
Tabulka 8.2: Počet řádů polynomu, nutných pro výpočet funkce e^t se zadanými parametry tmax a eps.

V tabulce 8.3 jsou uvedené zjištěné řády polynomů pro funkci $\mathrm{tg}(t)$. Taylorův polynom aproximující tuto funkci nabývá tvaru

$$\mathrm{tg}(t) = t + \frac{2}{3!}t^3 + \frac{16}{5!}t^5 + \frac{272}{7!}t^7 + \frac{7936}{9!}t^9 + \cdots \tag{8.24}$$

Z grafického znázornění naměřených hodnot na obrázku 8.3 je vidět mírný pokles trendu nárůstu řádu při zvětšujícím se intervalu a zvyšující se požadované přesnosti. Toto je pravděpodobně způsobeno numerickou nestabilitou, jelikož již 13. koeficient Taylorovy řady je automaticky konvertován v průběhu výpočtu na desetinné číslo, aby nedošlo k přetečení a tudíž se začne projevovat chyba vzniklá při přičítání malých hodnot.

Tabulka 8.4 a obrázek 8.4 zachycuje zjištěné řády polynomů pro funkci $\sin(t) \cdot \cos(t)$. Jak bude rozebráno v kapitole věnované numerické nestabilitě, nestabilita výpočtu se u této funkce projevuje velmi rychle, a tudíž nebylo možné ověřovat závislost na velkém intervalu.



Obrázek 8.1: Vliv délky požadovaného intervalu a požadované přesnosti výpočtu na řád polynomu pro funkci $\sin(t)$.

	eps					
`tmax`	1E-1	1E-5	1E-10	1E-20	1E-50	1E-100
0,01	3	4	6	12	24	66
0,05	3	6	8	16	34	72
0,1	4	6	10	18	42	80
0,2	4	8	14	24	54	100
0,3	4	10	16	30	64	118
0,5	4	12	22	40	86	148
1,0	8	26	44	72	138	> 150
1,5	30	58	86	120	> 150	> 150

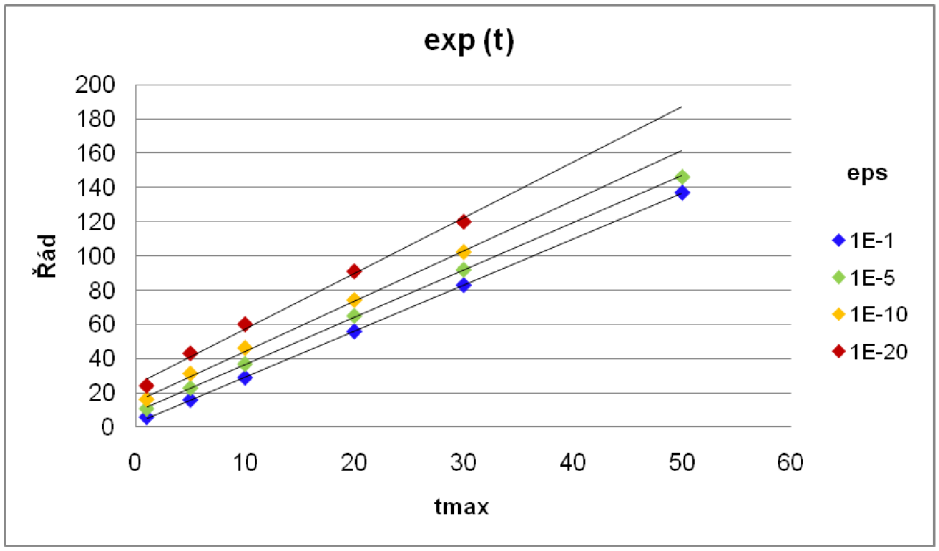
Tabulka 8.3: Počet řádů polynomu, nutných pro výpočet funkce $\mathrm{tg}(t)$ se zadanými parametry tmax a eps.

8.3 Časová a paměťová náročnost výpočtů

V této části se zabýváme časovou a paměťovou náročností výpočtů koeficientů Taylorovy řady. Pro každou zvolenou počáteční úlohu testujeme tyto parametry na Algoritmu 4, který pro rozvoj polynomu nevyužívá Pascalovy trojúhleníky, a označovaném v grafech jako „Normal“, a na Algoritmu 8, který Pascalovy trojúhleníky využívá a je v grafech je označen jako „Pascal“.

Na obrázku 8.5 a 8.6 jsou zobrazené závislosti potřebné operační paměti na řádu Taylorova polynomu pro funkce $\sin(t)$ a e^t, reprezentovanými počáteční úlohou (8.20), respektive (8.21). Je vidět, že pro tento tvar rovnic je paměťová náročnost obou algoritmů lineární, a že se velikost operační paměti, využité oběma algoritmy, příliš neliší. To je způsobeno tím, že získávání vyšších derivací je v tomto případě výpočetně nenáročné. Doba trvání výpočtů u obou počátečních úloh byla zanedbatelná a proto ji zde neuvádíme.

Na obrázku 8.8 a 8.7 je zachycena naměřená paměťová a časová náročnost funkce



Obrázek 8.2: Vliv délky požadovaného intervalu a požadované přesnosti výpočtu na řád polynomu pro funkci e^t.

	eps				
`tmax`	1E-5	1E-10	1E-20	1E-50	1E-100
1	12	16	24	44	72
2	12	20	28	52	84
3	16	24	32	60	96
4	20	28	40	64	104

Tabulka 8.4: Počet řádů polynomu, nutných pro výpočet funkce $\sin(t) \cdot \cos(t)$ se zadanými parametry tmax a eps.

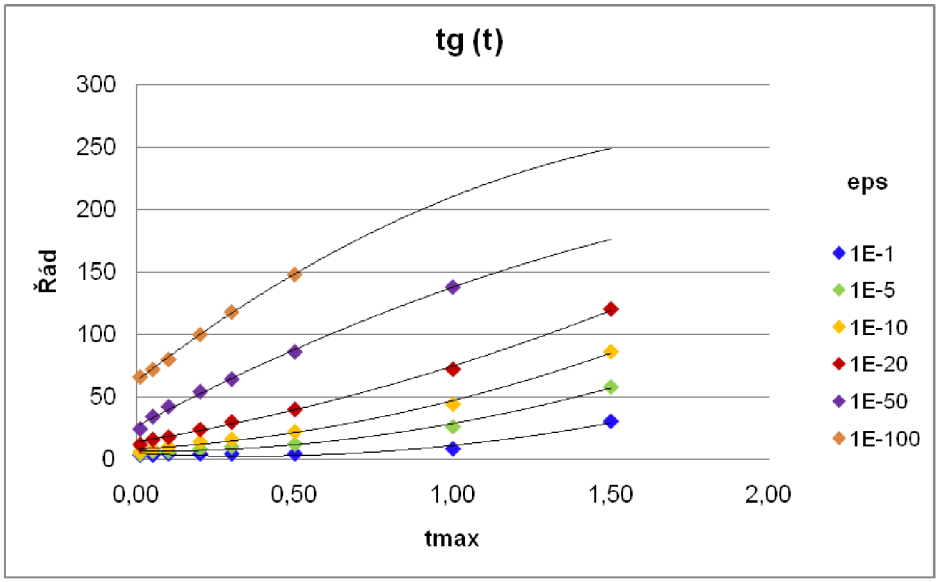
$\sin(y(t))$, reprezentované počáteční úlohou ve tvaru

$$y' = \sin(y) \qquad y(t_0) = 0 \tag{8.25}$$

Tato počáteční úloha je zajímavá tím, že se od úlohy (8.20) liší jen minimálně, ale její časová i paměťová složitost je zcela jiná. Z naměřených závislostí je patrné, že paměťová náročnost pro výpočet koeficientů Taylorovy řady pomocí multidimenzionálních Pascalových trojúhelníků roste mírně, zatímco bez jejich využití roste exponenciálně. Na druhou stranu algoritmus využívající prostý výpočet má velmi krátkou dobu trvání výpočtu, zatímco algoritmus využívající Pascalovy trojúhelníky má časovou složitost exponenciální. Máme-li tedy dostatečně velkou operační paměť, může být výhodnější použít standardní postup pro výpočet vyšších derivací. Pokud ovšem vyčerpáme dostupnou paměť a operační systém je nucen využívat swap, pak se rychlost algoritmu výrazně zpomalí, jelikož algoritmus vždy využívá celou alokovanou paměť.

Na obrázku 8.9 a 8.10 je zobrazena časová a paměťová náročnost funkce $\mathrm{tg}(t)$, reprezentované počáteční úlohou (8.22). Časová i paměťová náročnost se svým charakterem podobají výsledkům získaným při řešení počáteční úlohy (8.25) a tudíž i zde platí stejné závěry jako v předchozím případě.

Totéž platí pro obrázky 8.11 a 8.12, které řeší funkci $\sin(t) \cdot \cos(t)$, reprezentovanou počáteční úlohou (8.23).



Obrázek 8.3: Vliv délky požadovaného intervalu a požadované přesnosti výpočtu na řád polynomu pro funkci $\mathrm{tg}(t)$.

Na obrázku 8.13 a 8.14 je časová a paměťová složitost počáteční úlohy ve tvaru

$$y_1' = y_1 y_2 y_3 \qquad y_1(t_0) = 1 \tag{8.26}$$
$$y_2' = y_1 y_3 \qquad y_2(t_0) = 1 \tag{8.27}$$
$$y_3' = y_3 \qquad y_3(t_0) = 1 \tag{8.28}$$

Tato počáteční úloha je již složitější. Počet členů jednotlivých derivací rychle narůstá a tím se i Algoritmus 4, využívající postupné získávání jednotlivých derivací, stává zcela nepoužitelným. Z naměřených výsledků je vidět, že jak časová, tak paměťová složitost tohoto algoritmu roste exponenciálně, což nám také neumožnilo změřit výsledky až do řádu 400, jako u ostatních počátečních úloh, ale byli jsme nuceni ukončit měření při řádu 100. Algoritmus 8, využívající Pascalovy trojúhelníky, je v tomto případě velmi efektivní.

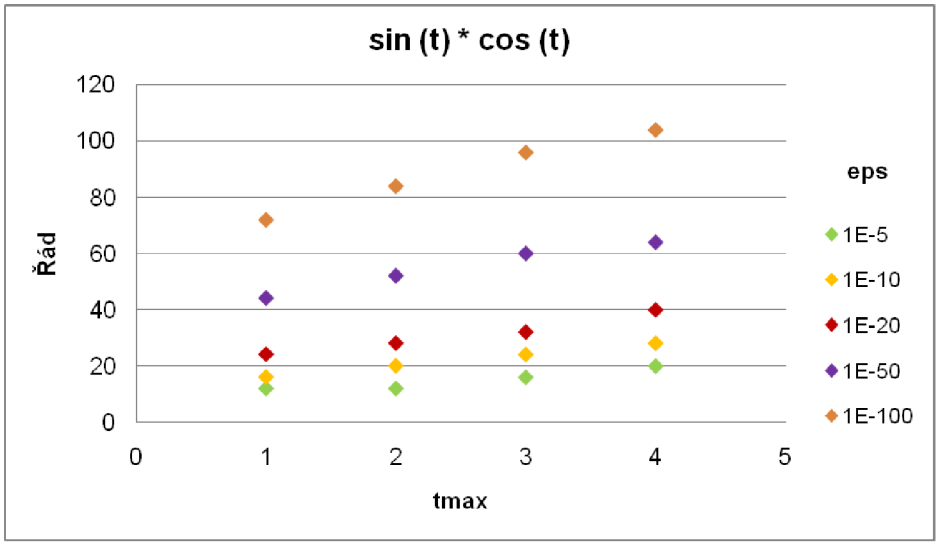
8.4 Numerická nestabilita

V této části se budeme věnovat numerické nestabilitě způsobené sčítáním desetinných čísel s řádově odlišným exponentem. Pro demonstraci korektnosti nalezeného řešení jsme vybrali funkce $\sin(t) \cdot \cos(t)$ a $\ln(t)$, reprezentované počáteční úlohou (8.23), respektive

$$y' = \frac{1}{t} \qquad y(t_0) = 1 \tag{8.29}$$

Vzhledem k tomu, že funkce $\ln(t)$ není definována v bodě 0, je třeba posunout počáteční bod t_0 do 1. Program Taylor.jar ovšem umožňuje změnit mez výpočtu hodnot Taylorova polynomu a tak můžeme zpětně dopočítat hodnoty pro interval $(0; 1)$.

Na obrázku 8.15 je zobrazeno srovnání aproximovaného řešení počáteční úlohy (8.23) s jejím analytickým řešením. Je patrné, že na intervalu vyšším než $(0; 3, 5)$ je již aproximované řešení nedostatečné. To je způsobeno velikostí koeficientů Taylorova polynomu a



Obrázek 8.4: Vliv délky požadovaného intervalu a požadované přesnosti výpočtu na řád polynomu pro funkci $\sin(t) \cdot \cos(t)$.

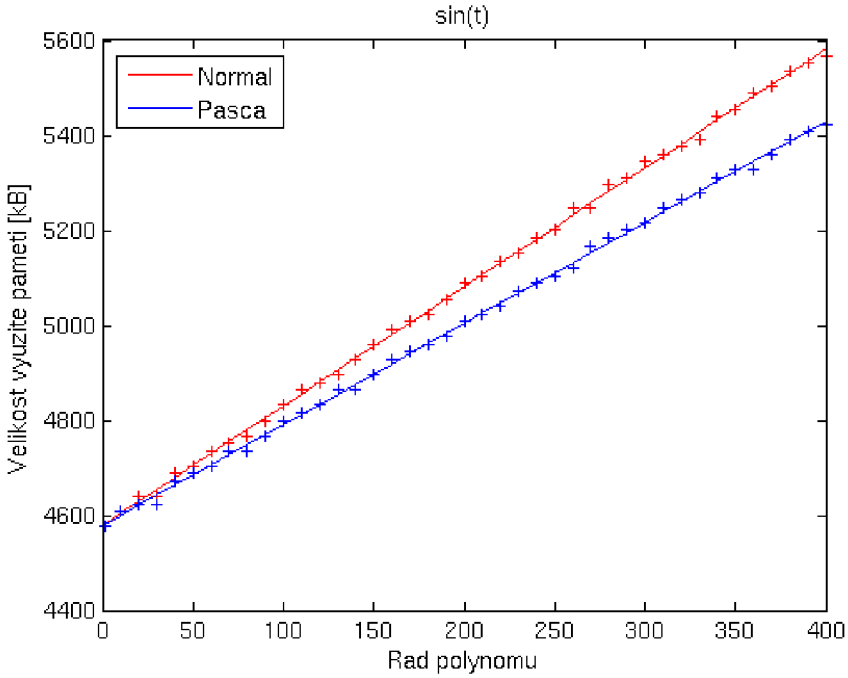
použitou aritmetikou. Koeficienty Taylorova polynomu pro počáteční úlohu (8.23) narůstají pozvolna. V řádu 35 ovšem dojde k automatické konverzi na desetinné číslo, aby se zabránilo přetečení. Pokud by koeficienty narůstaly rychle, tak by chyba při zaokrouhlování nebyla vysoká a interval by mohl být větší. Koeficienty pro tuto počáteční úlohu ovšem zůstávají dlouho okolo hodnoty, kdy dochází ke konverzi. Jednotlivé sčítance při výpočtu multidimenzionálních Pascalových trojúhelníků jsou tedy malé a při výpočtu se často zaokrouhlují. To vede k poměrně rychlému nárůstu nedetekovatelné chyby.

Na obrázku 8.16 je zobrazeno analytické řešení počáteční úlohy (8.29) a její aproximace Taylorovým polynomem. Toto řešení má obdobný problém jako v předchozím případě a interval, na kterém se aproximace shoduje s analytickým řešením, je poměrně malý.

8.5 Shrnutí výsledků

V této kapitole jsme demonstrovali rozdílné výsledky aproximací různých počátečních úloh. Obecně lze konstatovat, že algoritmus využívající postupné získávání jednotlivých derivací může být na jednodušších typech počátečních úloh rychlejší, ale vždy také paměťově náročnější, než algoritmus získávání vyšších derivací pomocí multidimenzionálních Pascalových trojúhelníků. U složitějších počátečních úloh je často prostý výpočet také neúnosně časově náročný. Proto, pokud nejsme schopni odhadnout složitost počáteční úlohy, je lepší standardně využívat algoritmus založený na multidimenzionálních Pascalových trojúhelnících, u kterého máme jistotu, že při výpočtu nevyčerpáme celou operační paměť počítače.

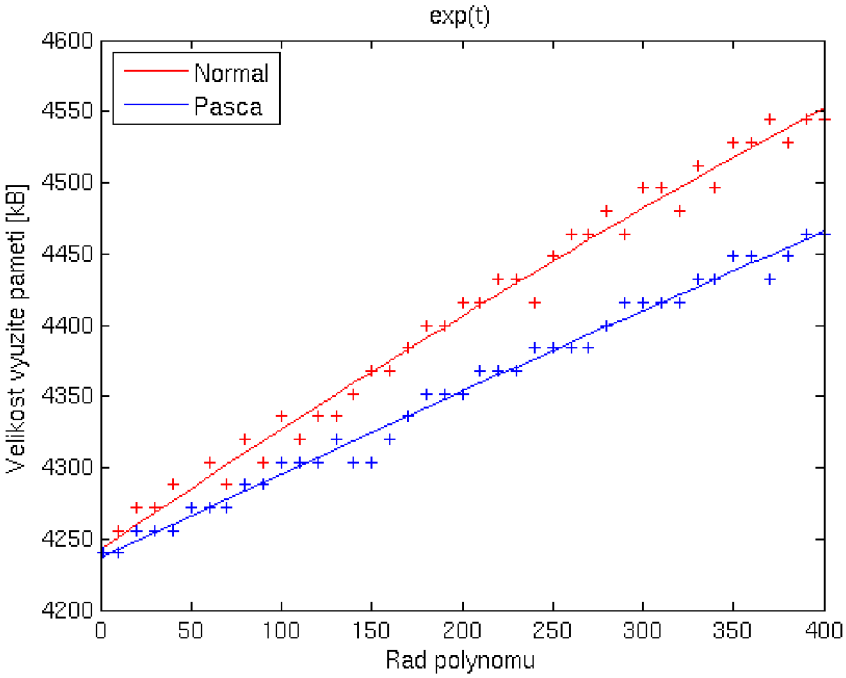
Jedním z cílů této diplomové práce bylo prozkoumat vlastnosti Taylorových polynomů aproximujících dané funkce. Zjistili jsme, že tyto polynomy aproximují funkce na poměrně velkém okolí počátečního bodu, ale pokud je požadováno okolí větší než $(t_0; t_0+4)$, pak se již u mnoha funkcí projeví numerická nestabilita. Tento problém může být odstraněn použitím aritmetiky s vyšší přesností. Obecně při výpočtu používáme pouze racionální čísla. Všechny výpočty jsou pouze dělení, násobení, sčítání a odečítání, což vede opět jen na racionální čísla. Proto lze této vlastnosti využít a všechna čísla v paměti udržovat ve tvaru podílu



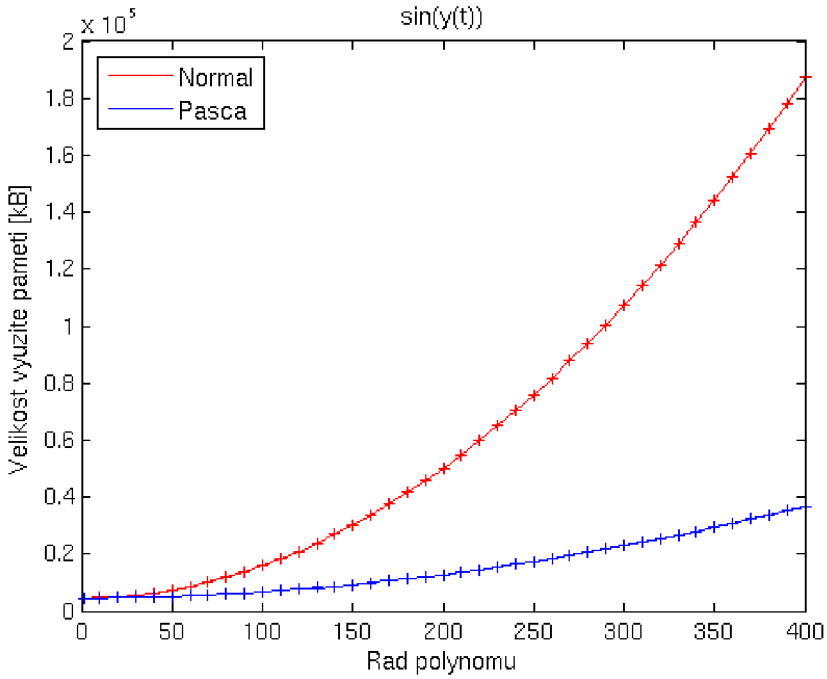
Obrázek 8.5: Závislost velikosti využité operační paměti na počítaném řádu Taylorova polynomu pro funkci $\sin(t)$.

dvou celých čísel dostatečné velikosti.

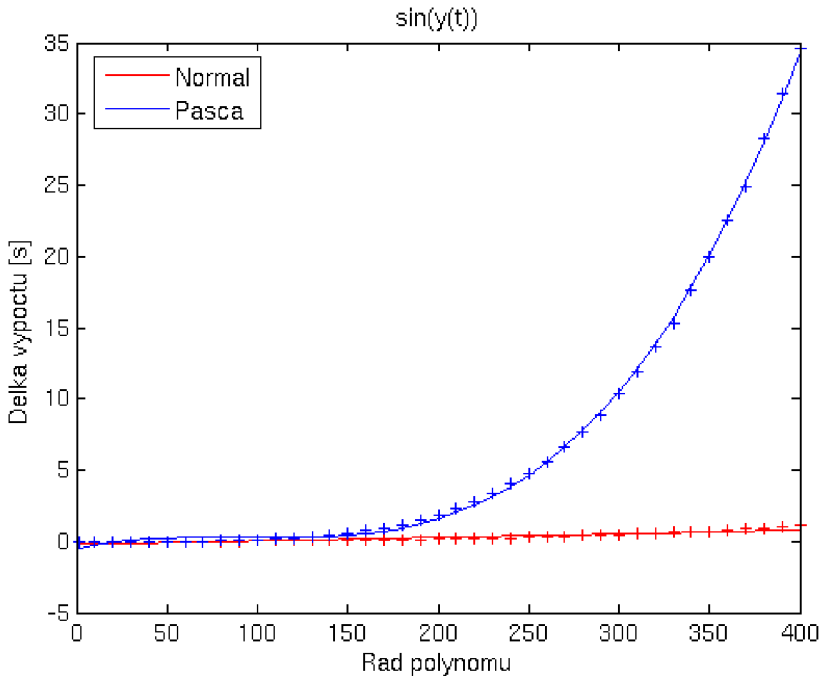
Druhý problém je velmi rychlý nárůst potřebného řádu aproximovaného polynomu při zvětšování intervalu $[t_0; t_max]$. Je vidět, že u složitějších funkcí může trvat výpočet 400 členů Taylorova polynomu řádově desítky sekund. Větší interval by vedl na vyšší požadavek na počet polynomů, což by mělo za následek neúnosný nárůst doby výpočtu. Proto je lepší použít krokovou metodu a počítat více polynomů pro kratší intervaly s tím, že počáteční podmínky výpočtu dalších polynomů se vypočítají jako okrajové hodnoty polynomů předchozích.



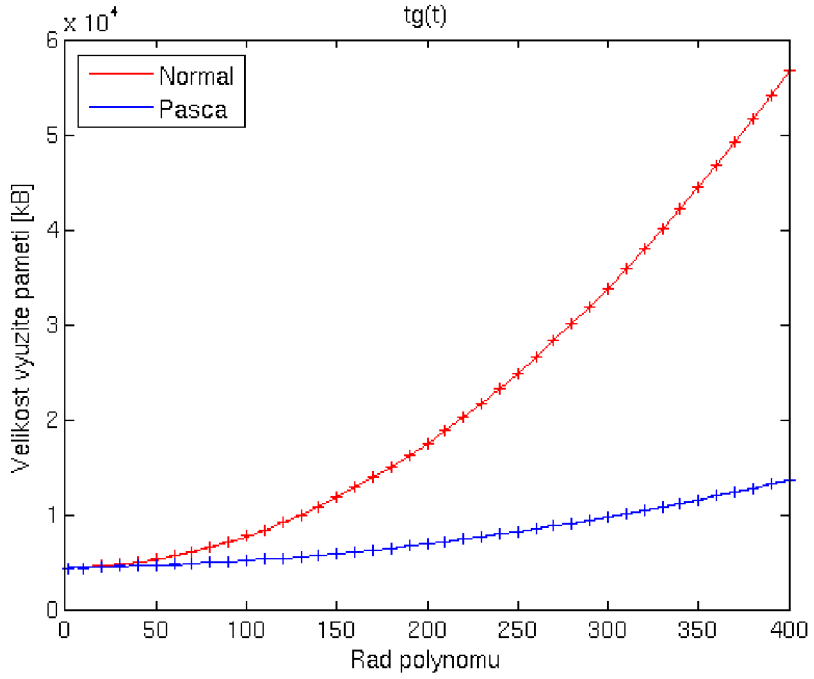
Obrázek 8.6: Závislost velikosti využité operační paměti na počítaném řádu Taylorova polynomu pro funkci e^t.



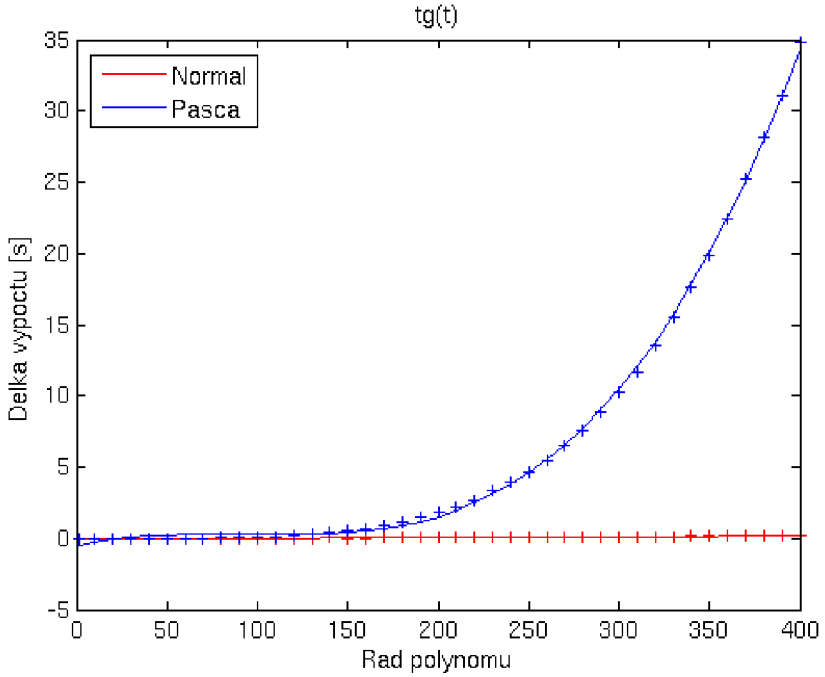
Obrázek 8.7: Závislost velikosti využité operační paměti na počítaném řádu Taylorova polynomu pro funkci $\sin(y)$.



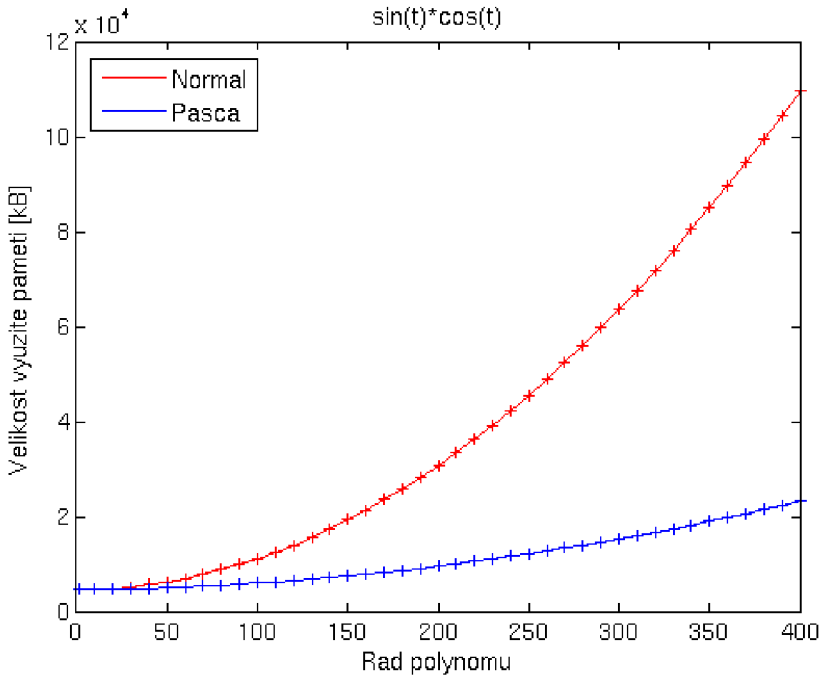
Obrázek 8.8: Závislost délky výpočtu na počítaném řádu Taylorova polynomu pro funkci $\sin(y)$.



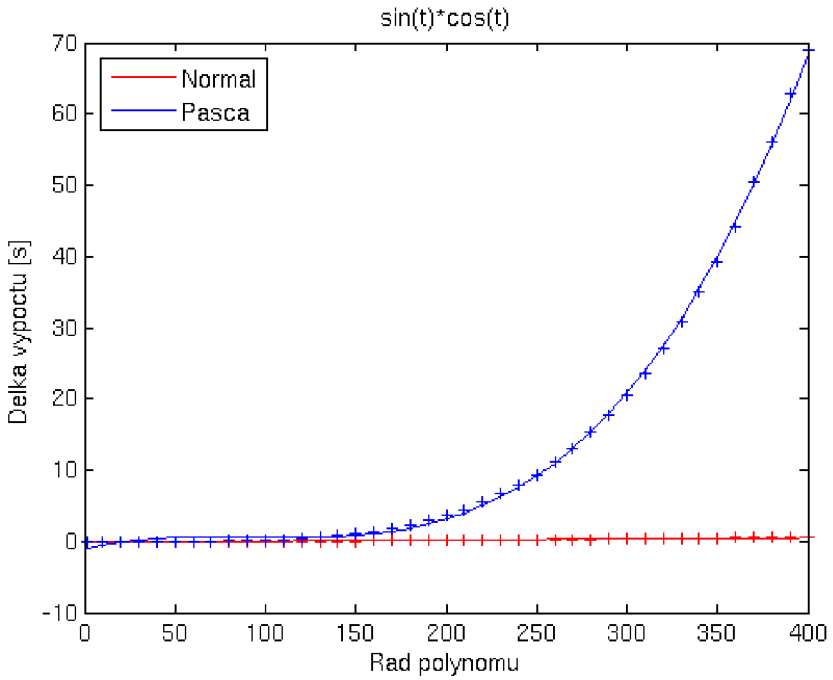
Obrázek 8.9: Závislost velikosti využité operační paměti na počítaném řádu Taylorova polynomu pro funkci $\mathrm{tg}(t)$.



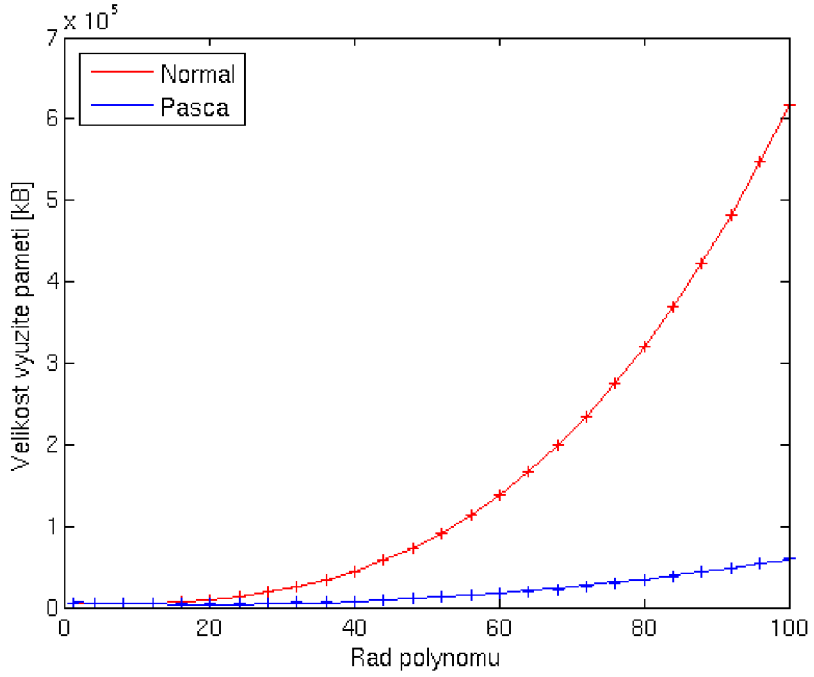
Obrázek 8.10: Závislost délky výpočtu na počítaném řádu Taylorova polynomu pro funkci $\mathrm{tg}(t)$.



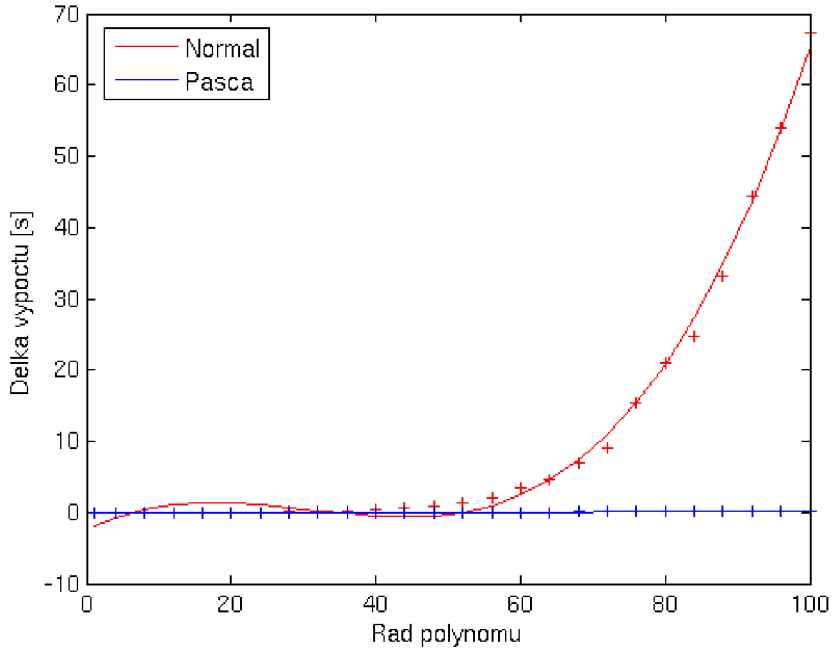
Obrázek 8.11: Závislost velikosti využité operační paměti na počítaném řádu Taylorova polynomu pro funkci $\sin(t) \cdot \cos(t)$.



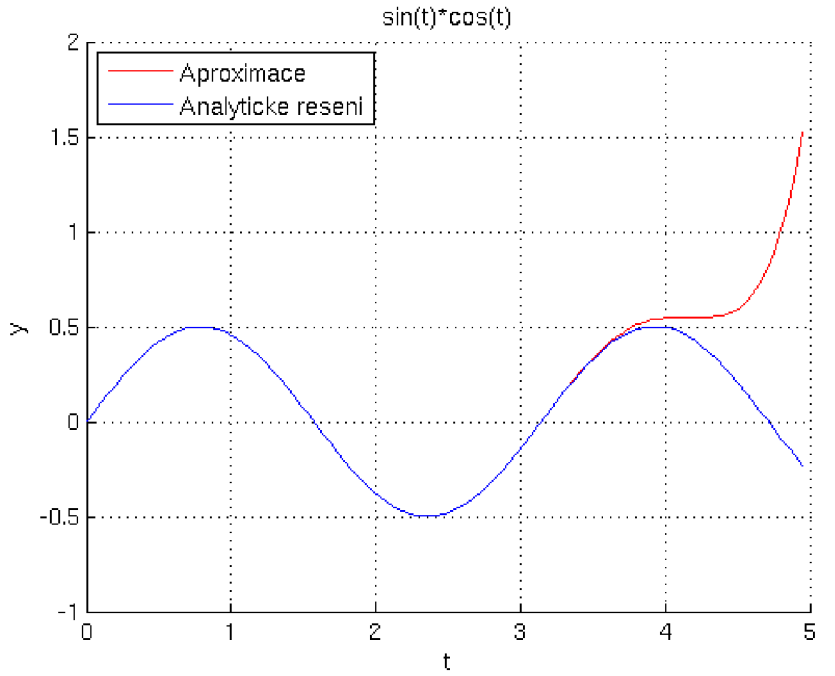
Obrázek 8.12: Závislost délky výpočtu na počítaném řádu Taylorova polynomu pro funkci $\sin(t) \cdot \cos(t)$.



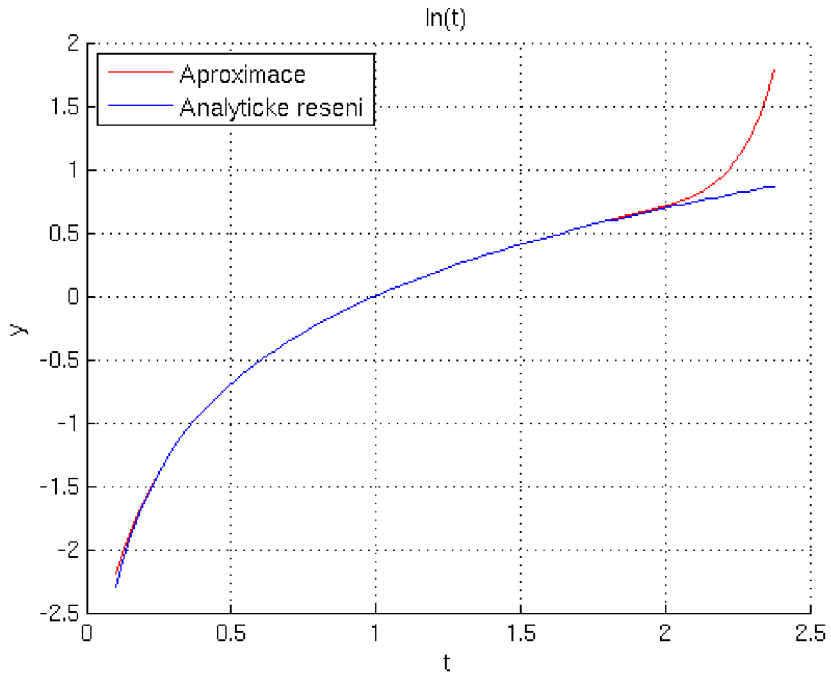
Obrázek 8.13: Závislost velikosti využité operační paměti na počítaném řádu Taylorova polynomu pro počáteční úlohu (8.26).



Obrázek 8.14: Závislost délky výpočtu na počítaném řádu Taylorova polynomu pro počáteční úlohu (8.26).



Obrázek 8.15: Srovnání aproximace funkce $\sin(t) \cdot \cos(t)$ s analytickým řešením.



Obrázek 8.16: Srovnání aproximace funkce $\ln(t)$ s analytickým řešením.

Kapitola 9

Závěr

Tato práce se zabývá řešením diferenciálních rovnic pomocí Taylorovy řady a transformací matematických funkcí na soustavy diferenciálních rovnic, což umožňuje nahrazeních funkcí v diferenciálních rovnicích nově zavedenými proměnnými a následnou optimalizaci rovnic pro jejich výpočet. Je zde dokázáno, že transformované soustavy diferenciálních rovnic mají stejné řešení, jako soustavy původních rovnic.

Pro charakteristiku funkcí, které lze transformovat na soustavu diferenciálních rovnic práce nově zavádí pojem „derivační uzávěr funkce“ a ukazuje, že tento derivační uzávěr musí být konečný, aby funkce mohla být transformována. Zároveň práce matematicky dokazuje, že všechny matematické funkce běžně využívané pro popis technických systémů lze transformovat na soustavu diferenciálních rovnic a uvádí transformační rovnice pro tyto funkce.

Dále práce popisuje simulační jazyk TKSL využívaný programem TKSL/386, který řeší počáteční úlohu pomocí Taylorovy řady a ukazuje, jak se dají transformované soustavy diferenciálních rovnic převést na speciální systém numerických integrátorů.

Práce se také zabývá aproximací funkcí pomocí Taylorových polynomu a jejich aplikací pro řešení počátečních úloh. Pro účely efektivních výpočtů koeficientů Taylorových řad předkládáme upravené vztahy a algoritmy pro multidimenzionální Pascalovy trojúhelníky a jejich efektivní výpočet.

Součástí práce je implementace všech navržených algoritmů v jednom matematickém jádře a implementace grafického rozhraní pro usnadnění práce s matematickým jádrem.

Ze zjištěných výsledků vyplývá, že se zvětšujícím se intervalem, na kterém hledáme řešení, rychle narůstá řád polynomu, což vede ke zvýšené výpočetní náročnosti. Na základě těchto poznatků jsme dospěli k závěru, že je lepší pro velký interval počítat více Taylorových polynomů, které budou pokrývat vždy jen část intervalu. Tento způsob také sníží náchylnost k numerické nestabilitě.

Zajímavou tématikou na další směřování v této problematice je konvergence Taylorových řad a jejich shod s aproximovanými funkcemi, pokud se omezíme pouze na funkce, které mají konečný derivační uzávěr a pro které platí, že všechny funkce z jejich derivačního uzávěru jsou v bodě t_0 omezené. Veškeré poznatky nás zatím vedou k domněnce, že v tomto případě nekonečná Taylorova řada vždy konverguje k aproximované funkci na nejmenším intervalu spojitosti funkcí z derivačního uzávěru, který obsahuje bod t_0. Tato problematika je úzce spojena s ukončovací podmínkou pro výpočet koeficientů Taylorových řad, kterou jsme prozatím ověřili pouze empiricky.

Literatura

[1] Apostol, T. M.: *One-Variable Calculus, with an Introduction to Linear Algebra*. John Wiley & Sons, Inc., 1967.

[2] Eaton, E. A.: A multidimensional extension of Pascal's triangle. *SIGSMALL/PC Notes*, ročník 16, May 1990: s. 34–37, ISSN 0893-2875, doi:http://doi.acm.org/10.1145/1059960.1059964.
URL http://doi.acm.org/10.1145/1059960.1059964

[3] Holmes, M. H.: *Introduction to Numerical Methods in Differential Equations*. Springer, 2007.

[4] Kunovský, J.: *Modern Taylor Series Method*. Brno, 1994.

[5] Mikulásek, K.; Kunovsky, J.: Taylorian Initial Problems. In *Proceedings of the 12th European Simulation Multiconference on Simulation - Past, Present and Future*, SCS Europe, 1998, ISBN 1-56555-148-6, s. 718–722.
URL http://portal.acm.org/citation.cfm?id=647916.760912

[6] Mikulášek, K.: *Polynomial Transformations of Systems of Differential Equations and Their Applications*. Dizertační práce, Brno University of Technology, Faculty of Electrical Engineering and Computer Science, 2000.

[7] Richet, Y.: JMathPlot. 2003.
URL http://jmathtools.berlios.de/

Dodatek A

Obsah CD

V kořenovém adresáři přiloženého CD je uložena PDF verze této práce. Dále je zde adresář thesis, který obsahuje zdrojové kódy k přeložení práce z formátu Latex a všechny potřebná obrázky a citace.

V adresáři taylor se nachází zdrojové kódy k matematickému jádru programu společně s binárním zakompilovaným souborem, který je spustitelný na většině Linuxových platformách (včetně serveru Merlin).

V adresáři GUI je umístěn spustitelný Java Arichive soubor s programem Taylor.jar a zdrojovými soubory tohoto programu.

Dodatek B

Manuál programu taylor

Program taylor se spouští příkazem

`./taylor [options] inputFile [outputFile]`

Pokud není zadán outputFile, vypíše se výsledek na standardní výstup. Program taylor podporuje tyto volby:

- h - Vypíše nápovědu
- t - Provede transformaci vstupních rovnic do polynomiálního tvaru a výsledek vypíše
- x - Vypíše výsledné polynomy ve formátu XML.
- s - Testuje funkci softwaru. Pokud vše funguje, vypíše „Test OK.“ (využívá se pro ověření spojení v Taylor.jar)
- d - Vypíše derivace vstupních rovnic až do řádu zadaném pomocí `order`. Pokud řád nebyl zadán, tak vypíše 5 derivací.
- p - Vygeneruje Taylorovy polynomy pomocí metody postupné derivace.
- a - Vygeneruje Taylorovy polynomy pomocí upravených multidimenzionálních Pascalových trojúhelníků.

Pří výpočtu koeficientů Taylorovy řady se buď počítá tolik koeficientů, kolik je zadáno pomocí systémové proměnné `order`, nebo se řád určuje dynamicky podle zvolené maximální chyby a intervalu. Maximální řád dynamického určení je 150.

B.1 Příklad zápisu vstupních rovnic

Počáteční úlohu (8.1) by jsme zapsaly takto:

```
system {
 tmax = 10;
 eps = 1e-20;
}
y_1' = y_11 + y_12 & 0;
y_2' = y_21 + y_22 & 0;
y_11 = exp(-1/3*ln(y_1^4 + 1));
```

```
y_12 = t^4;
y_21 = exp(t);
y_22 = exp(-1*sin(y_1));
```

Počáteční úlohu (8.29) můžeme zapsat takto:

```
system {
 tmin = 1;
 order = 20;
}
y' = t^-1 & 0;
```