Numerické metody řešení obyčejných diferenciálních rovnic vyššího řádu

Bc. Michal Karafiát

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Michal Karafiát**
Osobní číslo:	**A17458**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Numerické metody řešení obyčejných diferenciálních rovnic vyššího řádu**
Téma práce anglicky:	**Numerical Methods for Higher Order Ordinary Differential Equations**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Popište možnosti řešení obyčejných diferenciálních rovnic vyšších řádů pomocí metod numerické integrace.
3. Sestavte uživatelsky přívětivý program pro řešení obyčejných diferenciálních rovnic vyšších řádů pomocí vybraných numerických metod.
4. Sestavte matematický model (diferenciální rovnici vyššího řádu) vybraného reálného systému.
5. Proveďte simulaci a analyzujte přesnost simulace vybraného reálného systému pomocí jednotlivých numerických metod.
6. Zhodnoťte využití jednotlivých numerických metod a jejich reálné nasazení.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. YA YAN LU. *Numerical Methods for Differential Equations*. Kowloon, Hong Kong. City University of Hong Kong.
2. BUTCHER, J. C. *Numerical methods for ordinary differential equations*. 2nd ed. Hoboken, NJ: Wiley, c2008. ISBN 978-0-470-72335-7.
3. RALSTON, Anthony. *Základy numerické matematiky: příručka pro univerzity ČSR*. 2. čes. vyd. Praha: Academia, 1978.
4. BURDEN, Richard L. a J. Douglas FAIRES. *Numerical analysis*. 8th ed. Belmont, CA: Thomson Brooks/Cole, c2005. ISBN 05-343-9200-8.
5. *ODE Laboratories: A Sabbatical Project by Christopher A. Barker* [online]. San Joaquin Delta College, 5151 Pacific Ave., Stockton, CA 95207, USA: San Joaquin Delta College, 2017 [cit. 2019-11-24]. Dostupné z: http://calculuslab.deltacollege.edu/

Vedoucí diplomové práce: **doc. Ing. Bc. Bronislav Chramcov, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

prof. Mgr. Roman Jašek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 17.05.2021

Michal Karafiát v.r.
podpis diplomanta

ABSTRAKT

Systémy diferenciálních rovnic nejčastěji nelze řešit analyticky. Proto algoritmy založené na numerických metodách jsou potřeba. Cíl této práce je vytvořit obecný program pro řešení diferenciálních rovnic vyšších řádů pomocí různých diferenciálních metod. Bude provedeno porovnání chyb každé metody. Dále bude vybraný reálný systém popsán pomocí diferenciální rovnice vyššího řádu. Pro simulaci tohoto systému bude vytvořeno uživatelsky přívětivé softwarové prostředí. Budou použity vybrané numerické metody.

Klíčová slova: Numerické metody, obyčejné diferenciální rovnice, počáteční úloha, Eulerova metoda, Runge-Kutta, Runge-Kutta-Fehlberg, Adamsovy metody, prediktor-korektor, streamlit,

ABSTRACT

Most often, systems of differential equations can not be solved analytically. Algorithms based on numerical methods are therefore needed. The aim of work is to create a general program for the solution of differential equations of higher order by means of different numerical methods. A comparison of the errors of each method will be performer. Next, the selected real systém will be described using a high-order differential equation. A user-friendly software environment will be created for the simulation of this systém. The selected numerical methods will be used.

Keywords: Numeric methods, ordinary differential equations, initial value problém, Euler's method, Runge-Kutta, Runge-Kutta-Fehlberg, Adams methods, predictor-corrector, streamlit

V první řadě děkuji vedoucímu diplomové práce panu doc. Ing. Bronislavu Chramcovovi, Ph.D. za cenné a užitečné rady a především trpělivost, kterou mi poskytl.

Rád bych také poděkoval své rodině za podporu a umožnění studia na Univerzitě Tomáše Bati, hlavně těm, kterým už jinak poděkovat nemůžu.

OBSAH

ÚVOD

Diferenciální rovnice a numerické metody propojuje společná tématika. Často se totiž společně vyskytují v praxi. Moderní věda by se bez diferenciálních rovnic a numerické matematiky neobešla. Zatímco diferenciální rovnice pohlíží na svět spíše v teoretické rovině, numerická matematika nabízí spíše ten praktický pohled. Je to dáno tím, že numerické metody nabízí řešení tam, kde například není možné nalézt analytické řešení, nebo nalezení takového řešení je příliš komplikované. V praxi se také často můžeme setkat se situací, kdy určitá míra chyby je pro naši potřebu tolerovatelná, protože vyšší přesnost pro nás stejně většinou znamená vyšší cenu. Proto je potřeba se v souvislosti s numerickými metodami zabývat i jejich charakteristickými vlastnostmi jako je odhad chyby numerické metody, stabilita a konvergence.

Většinu problémů spojité matematiky (například téměř jakýkoli problém zahrnující derivace nebo integrály) nelze vyřešit, a to ani v zásadě, v konečném počtu kroků, a proto musí být vyřešen (teoreticky nekonečným) iteračním procesem, který nakonec konverguje k řešení. V praxi samozřejmě není prováděno nekonečně mnoho iterací, ale pouze tolik, než bude odpověď přibližně správná, tzv. „dostatečně blízko“ k požadovanému výsledku pro praktické účely. Jedním z nejdůležitějších aspektů vědeckých výpočtů je tedy nalezení rychle konvergentních iteračních algoritmů a posouzení přesnosti výsledné aproximace. Pokud je konvergence dostatečně rychlá, mohou být i některé problémy, které lze vyřešit konečnými algoritmy, jako jsou systémy lineárních algebraických rovnic, v některých případech lépe vyřešeny iterativními metodami.

Obecně numerické techniky řešení matematických úloh byly vyvinuty a používány před stovkami, ba dokonce i tisíci lety zpátky. Implementace těchto metod však byla velmi složitá, protože veškeré výpočty dříve musely být prováděny ručně nebo s pomocí jednoduchých mechanických výpočetních zařízeních, což omezovalo počet výpočtů, které bylo možno provést, stejně jako rychlost a přesnost. Dnes se numerické metody používají u rychlých elektronických digitálních počítačů, které umožňují provádět mnoho jinak zdlouhavých a opakujících se výpočtů, které produkují přesná (i když ne exaktní) řešení ve velmi krátkém čase.

I. TEORETICKÁ ČÁST

1 DIFERENCIÁLNÍ ROVNICE

Rovnice obsahující derivace závislé proměnné se nazývá diferenciální rovnice. Diferenciální rovnice zahrnující pouze jednu nezávisle proměnnou se nazývá obyčejná diferenciální rovnice. Pokud diferenciální rovnice zahrnuje dvě nebo více nezávisle proměnných (s parciálními derivacemi), nazýváme ji parciální diferenciální rovnice.

Tato práce se věnuje pouze obyčejným diferenciálním rovnicím (dále někdy zkráceně ODR) a jejich řešením pomocí numerických metod.

1.1 Obyčejné diferenciální rovnice

ODR lze klasifikovat dle několika kritérií, např. dle jejich řádu, linearity, homogenity a typu úlohy jako s počátečními podmínkami či okrajové úlohy.

1.1.1 Lineární ODR

ODR lze označit jako lineární nebo nelineární. Rovnice je **lineární**, jestliže je její závislost na y a jeho derivacích lineární. Libovolná lineární ODR může být zapsána v následující formě:

$$a_{n+1}(x)y^{(n)} + a_n(x)y^{(n-1)} + a_3(x)y'' + a_2(x)y' + a_1(x)y = c(x) \tag{1.1}$$

Koeficienty a v rovnici 1.1 jsou funkcemi nezávisle proměnné x.

1.1.2 Nelineární ODR

ODR je **nelineární,** pokud koeficienty a v rovnici 1.1 jsou funkcí y nebo jeho derivacemi, nebo pokud na pravé straně rovnice je c samo o sobě nelineární funkcí y, nebo pokud lineární člen a(x)y je nahrazen nelineární funkcí y. Příkladem pak můžou být rovnice 1.2.

$$y'' + \sin(y) = 4$$

$$y\,y'' + 2y = 9$$

$$y''y' + y = 2$$

$$y'' + 8y = \tan(y) \tag{1.2}$$

1.1.3 Homogenní a nehomogenní ODR

ODR může být homogenní nebo nehomogenní. V případě rovnice 1.1 je rovnice **homogenní** za podmínky, když na pravé straně c(x) = 0. V případě, kdy c(x) ≠ 0 pak ODR je **nehomogenní**.

1.1.4 Řád diferenciální rovnice

Řád diferenciální rovnice je definován řádem nejvyššího stupně derivace přítomným v dané rovnici. Samotný řád rovnice může sdělit důležité informace. Když je ODR vyřešena, určité konstanty jako např. integrační se objeví v řešení dané rovnice. Počet takových konstant je dán právě řádem diferenciální rovnice. To znamená, že ODR druhého řádu má dvě neurčené konstanty, k určení těchto dvou konstant je potřeba zadat dvě omezení.

Typickým příkladem ODR prvního řádu pak může být rovnice 1.3

$$y' + ay = f(x) \tag{1.3}$$

Typickým příkladem ODR druhého řádu pak může být rovnice 1.4.

$$y'' + y' + 2y = sin(x) \tag{1.4}$$

Typickým příkladem ODR třetího řádu pak může být rovnice 1.5.

$$y''' + ay'' + y' + by = c\sin(x) \tag{1.5}$$

Kde x je nezávisle proměnná, y je závisle proměnná (y = y(x)), f(x) je funkcí x, a, b a c jsou konstanty.

1.1.5 Úlohy s počátečními podmínkami

Rovnice je nazývána úlohou s počátečními podmínkami, jestliže hodnoty závisle proměnné nebo jejich derivace jsou známi v počátečních hodnotách závisle proměnné.

1.1.6 Okrajové úlohy

Rovnice je nazývána okrajovou úlohou, jestliže hodnota závisle proměnné nebo její derivace jsou známé více jak v jednom bodě nezávisle proměnné.

1.1.7 Soustava diferenciálních rovnic a redukce řádu rovnice

Numerickými metody, kterými se tato práce bude zabývat je možno řešit i soustavy diferenciálních rovnic, mnoho matematických problémů je také popsáno takovým systémem. Numerické řešení jedné nebo více ODR vyšších řádů se nejlépe získá, pokud je systém rovnic reprezentován ve formě soustavy n ODR prvního řádu.

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \dots, y_n), & y_1(x_0) &= y_{1,0} \\ y_2' &= f_2(x, y_1, y_2, \dots, y_n) & y_2(x_0) &= y_{2,0} \\ &\vdots \\ y_n' &= f_n(x, y_1, y_2, \dots, y_n) & y_n(x_0) &= y_{n,0} \end{aligned} \tag{1.5}$$

Tuto soustavu (rovnice 1.5) pak můžeme převést do vektorového zápisu. viz 1.6, kterému se říká Lorenzův:

$$y(x) = \begin{pmatrix} y_1(x) \\ y_2(x) \\ \vdots \\ y_n(x) \end{pmatrix}, \qquad f(x, y) = \begin{pmatrix} f_1(x, y_1, .., y_n) \\ f_2(x, y_1, \dots, y_n) \\ \vdots \\ f_n(x, y_1, y_2, \dots, y_n) \end{pmatrix}, \qquad y_0 = \begin{pmatrix} y_1(x) \\ y_2(x) \\ \vdots \\ y_3(x) \end{pmatrix} \tag{1.6}$$

V případě ODR vyššího řádu je pak možné metodou redukce řádů získat podobnou soustavu n diferenciálních rovnic s n počátečními podmínkami. Máme-li ODR vyššího řádu viz rovnice 1.7.

$$y^{(n)} = f\left(x, y, y', \dots, y^{(n-1)}\right) \tag{1.7}$$

Pomocí substituce můžeme převést na soustavu diferenciálních rovnic 1.8.

$$\begin{aligned} y_1 &= y, \\ y_2 &= y', \\ &\vdots \\ y_n &= y^{(n-1)} \end{aligned} \tag{1.8}$$

Dosazením zpátky do rovnice 1.7 získáváme rovnici 1.9.

$$y^{(n)} = f(x, y_1, y_2, \dots, y_n) \tag{1.9}$$

2 NUMERICKÉ METODY

Numerické metody jsou matematické techniky používané pro řešení matematických problémů, které se nedají řešit analyticky nebo jejich analytické řešení je příliš složité. [5]

Analytické řešení nám dává přesnou odpověď ve formě matematického výrazu, tedy matematickou rovnici obsahující proměnné, která reprezentuje problém, pro který je hledané řešení.

Numerické řešení je přibližná číselná hodnota, tzn. numerická řešení jsou pouhou aproximací, přesto mohou být velmi přesná. V mnoha numerických metodách jsou výpočty prováděny iterativním způsobem, dokud není dosaženo požadované přesnosti. [10]

Proces řešení pomocí numerických metod ve vědě a inženýrství můžeme rozdělit do několika fází:

1. **Definice problému –** udává nám popis problému, tedy seznam proměnných, které jsou zahrnuty, identifikování omezení ve formě mezních okrajů úlohy a/nebo počáteční podmínky. [12]
2. **Formulace řešení –** Formulace řešení se skládá z modelu (např. na základě fyzikálních zákonů), který je použit k reprezentaci problému a z odvození rovnic, které je třeba vyřešit. Příklady takových zákonů jsou např. Newtonovy zákony, zákon zachování hmotnosti, zákony termodynamiky apod. Modely, které jsou použity k řešení problému, musí být v souladu s metodami, které se následně použijí pro řešení rovnic. Pokud se očekává, že pro řešení budou použity analytické metody, musí být rovnice typu, který lže analyticky vyřešit. V případě potřeby je třeba formulaci, zjednodušit, aby bylo možné rovnice vyřešit analyticky. Pokud jsou pro řešení použity numerické metody, mohou být modely a rovnice komplikovanější. I tehdy, mohou však existovat určitá omezení. Například pokud formulace je taková, že numerické řešení vyžaduje dlouhý výpočetní čas, formulace možná bude muset být zjednodušena tak, aby bylo dosaženo řešení v rozumné době. Příkladem může být nějaký druh předpovědi události, jako např. počasí. Problém, který je potřebo řešit je komplexního rozsahu. Numerické modely, které se zde používají jsou velmi komplikované a numerická simulace počasí však musí být dostupná dostatečně včas. [12]
3. **Implementace numerického řešení –** pokud je problém řešen numericky, musí se vybrat numerická metoda. Pro každý typ matematického problému existuje celá řada

numerických technik, které lze použít. Techniky se mohou lišit v přesnosti, délce výpočtu a složitosti algoritmu. [12]

4. **Interpretace dosažených výsledků –** protože numerická řešení jsou řešením přibližným, a protože výpočetní program, který řeší numerickou metodu může obsahovat chyby, je potřeba numerické řešení zpětně podrobit analýze. To se dá udělat několika způsoby, v závislosti na daném problému. V případě více komplikovaných problémů jako jsou diferenciální rovnice, numerické řešení může být porovnáno s již známým řešením podobného problému, nebo problém může být řešen s jiným hraničním zadáním, nebo počátečním podmínkám, nebo s použitím jiné numerické metody. [12]

2.1 Charakteristika numerických metod

Numerické metody mají většinou tyto charakteristické vlastnosti: [7]

1. Postup řešení je iterativní proces, přesnost odhadované řešení se zlepšuje s každou iterací.
2. Postup řešení nabízí pouze aproximaci pravého, ale neznámého řešení.
3. Počáteční odhad řešení může být vyžadován
4. Postup řešení je konceptuálně jednoduchý, algoritmus reprezentující danou úlohu může být jednoduše programovatelný.
5. Postup řešení může občas divergovat od skutečného, spíše než konvergovat.

2.2 Analýza chyb

Obecně chyby můžou být klasifikovány na základě jejich původu jako nenumerické a numerické chyby. [3]

2.2.1 Nenumerické chyby

Mezi nenumerické chyby se řadí [3]:

1. Chyby modelů
2. Hrubé omyly a chyby
3. Nejistoty v datech a informacích

Chyby modelů vznikají tím, že predikční modely použité pro nastavování parametrů jsou většinou založeny na předpokladech a limitacích. Proto tyto modely mohou poskytnout predikované hodnoty, které se liší od odpovídajících skutečných hodnot daného problému.

Předpoklady a limitace kterou se vytváří k formulaci modelu a k redukci analytického řešení na rozumnou úroveň. Chyby modelů jsou tedy odchylky predikovaných hodnot od skutečných hodnot. [2]

Hrubé omyly a chyby můžeme obecně zavést lidský faktor, řadíme zde chyby ve výpočtech, chyby v algoritmech, nesprávné použití rovnice, použití prediktivních modelů za rámce jejich platnosti apod. Tyto chyby můžou být redukovány například zavedením kontrolních postupů, zjednodušení algoritmů, zjednodušení uživatelských rozhraních apod. [3]

Nejistota v datech může být připsána dvěma hlavními zdroji: [13]

- **Objektivní zdroj** – zahrnuje přirozenou míru variability použití a omezených informací. Například pouhý vzorek je použit k vyvození závěrů daného problému. Tento druh nejistoty může být řešen použitím pravděpodobnosti a statistiky. [2]
- **Subjektivní zdroj – s** nejistotou v datech se setkáváme, když nějaký odborník musí provést subjektivní posouzení, např. stav zkorodovaného ocelového nosníku. V tomto případě se posouzení může u jednotlivých odborníků lišit, což má za následek subjektivní nejistotu. [2]

2.2.2 Numerické chyby

Numerické chyby zahrnují [12]:

1. Chyby zaokrouhlování
2. Chyby numerické metody
3. Propagaci chyb
4. Chyby matematické aproximace

Chyby zaokrouhlování jsou způsobeny tím, že čísla jsou ve výpočetních zařízeních reprezentována konečným počtem bitů. Proto reálná čísla, která mají mantisu delší, než je počet bitů, které jsou k dispozici pro jejich reprezentaci, musí být zkráceny. Tento požadavek platí i pro iracionální čísla, která musí být reprezentována v jakémkoli systému konečným počtem číslic. Číslo lze zkrátit buď odříznutím, tedy vyřazením přebytečných číslic, nebo zaokrouhlením. Při osekávání jsou číslice v mantise, které přesahují délku, kterou lze uložit, jednoduše vynechány. Při zaokrouhlování se zaokrouhluje poslední uložená číslice. Jako jednoduchou ilustraci uveďme číslo 2/3. V počítači se zkrácení a zaokrouhlování provádí ve dvojkovém formátu). V desítkové soustavě se čtyřmi významnými číslicemi lze 2/3 zapsat jako 0,6666 nebo jako 0,6667. V prvním případě bylo skutečné číslo osekáno, zatímco v druhém

případě je číslo zaokrouhleno. V obou případech vede takové osekávání a zaokrouhlování reálných čísel k chybám v numerických výpočtech, zejména pokud se provádí mnoho operací. Tento typ (bez ohledu na to, zda je způsoben osekáním nebo zaokrouhlením) se nazývá **chyba zaokrouhlení** [13]. Velikost zaokrouhlovacích chyb závisí na velikosti daných čísel, která jsou zahrnuta, protože interval mezi čísly, která lze v počítači reprezentovat, závisí na jejich velikosti. K chybám zaokrouhlení pravděpodobně dojde, když se čísla, která jsou součástí výpočtů, významně liší svou velikostí a když se od sebe odečtou dvě téměř identická čísla. [12]

Chyby numerické metody vznikají, jestliže nám numerická metoda neposkytne přesné teoretické řešení dané metody. Příkladem je např. matematická konečná řada (např. Taylorova řada). Taylorova řada (viz kap 3.1.3) obsahuje nekonečné množství členů. První člen se nazývá člen nultého řádu. Druhý člen je člen prvního řádu, protože je založen na prvním derivátu. Třetí člen používá druhou derivaci a nazývá se člen druhého řádu atd. Což vede k tomu, že je běžné Taylorovu řadu zkrátit na řadu prvního řádu nebo někdy na řadu druhého řádu. Proto jsou zavedeny chyby metody. Účinky těchto chyb mohou být významné, pokud není správně formulován numerický algoritmus. Chyby numerické metody rozlišujeme na [13]:

- **Globální chyba metody:** lze jednoduše popsat jako chybu mezi skutečným řešením a odhadem řešení v bodě x, dle vztahu 2.0 [8]:

$$globální\ chyba\ v\ bodě\ x_{i+1} = y_{i+1}^{skutečné} - y_{i+1}^{numerické} \tag{2.0}$$

- **Lokální chyba metody:** je pak definována jako rozdíl skutečného řešení a řešením numerickým, které by ale bylo aproximováno z do následujícího bodu s použitím skutečné hodnoty. V případě Eulerovy metody by pak výpočet vypadal dle rovnice 2.1 [8]:

$$lokalní\ chyba\ v\ bodě\ x_{i+1} = y_{i+1}^{skutečné} - \left(y_i^{skutečně} + hf\left(x_i, y_i^{skutečně}\right)\right) \tag{2.1}$$

Propagace chyb lze někdy omezit např. zavedením výpočetního pořadí pro aritmetické operace v algoritmu, například při hodnocení součtu několika číselných hodnot pomocí stanoveného počtu významných číslic je žádoucí před provedením součtu seřadit číselné hodnoty

ve vzestupném pořadí. Výsledek tohoto součtu proto může mít sníženou chybu šíření ve srovnání s libovolným pořadím pro provedení součtu [13]

2.3 Vlastnosti numerických metod

Existuje několik dalších důležitých vlastností, kterými můžeme posuzovat jednotlivé metody pro řešení diferenciálních rovnic.

Mezi tyto vlastnosti řadíme [11]:

1. **Konzistentnost** – numerická metoda je konzistentní, jestliže lokální chyba metody je rovna nule v případě kdy aproximační krok se limitně blíží nule.
2. **Řád** – rychlost s jakou se snižuje globální chyba metody, v případě snižování velikosti aproximačního kroku.
3. **Stabilita** – vlastnost, u které není dovolen růst chyby z jakéhokoliv zdroje (zaokrouhlování, chyba metody) numerického postupu výpočtu z jednoho probíhajícího kroku k druhému.
4. **Konvergence** – metoda je konvergentní, pokud numerické řešení dosahuje analytického řešení, v případě, kdy velikost aproximačního kroku se limitně blíží nule. Obecně lze říct, že nutnou podmínkou konvergence je, aby metoda byla konzistentní a současně stabilní.

2.4 Výhody a nevýhody numerických metod

Jak bylo naznačeno v úvodu této kapitoly, numerické metody jsou alternativou k analytickým řešením. Pokud je to možné, jsou obvykle preferována analytická řešení. U analytického řešení problémů se řešení nachází přímo, nikoli iterativně jako u numerických metod, často však není možné získat analytické řešení a je nutná numerická metoda řešení. Analytické metody řešení problémů poskytují přesné řešení, pokud existuje. [9] Numerické řešení poskytuje pouze aproximaci, i když je-li vytvořen dostatečný počet iterací, nemusí existovat rozdíl mezi numerickým řešením a skutečnou hodnotou. Jelikož analytické metody poskytují řešení přímo, je potřebná doba výpočtu obvykle kratší než doba potřebná pro numerické řešení. Kromě výhod má analytické řešení problémů i nevýhody. Za prvé, postupy řešení jsou platné pouze pro jednoduché problémy a nelze je použít pro mnoho typů problémů, Zadruhé, s mnoha technickými problémy, řešení podléhá řadě omezení. Obecně je obvykle obtížné vyřešit problém analytickou metodou, pokud problém zahrnuje omezení, v takových

případech mají numerické metody tu výhodu, že s omezeními se manipuluje relativně snadno. [7]

Mezi výhody numerických metod tedy patří tyto vlastnosti: [7]

1. jsou koncepčně jednoduché,
2. jsou snadno programovatelné,
3. omezení jsou snadno zpracovatelná,
4. je možné řešení složitých problémů.

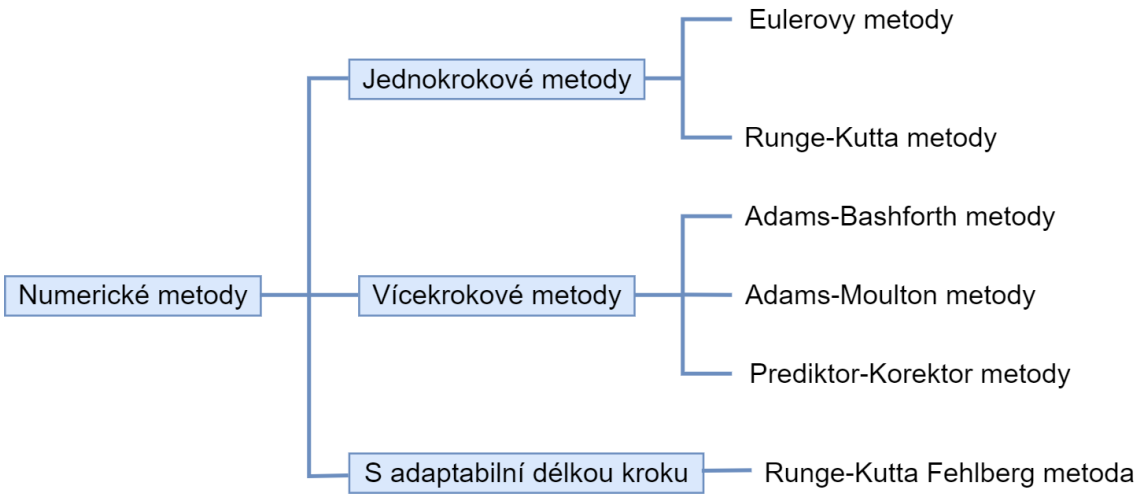
Numerické metody však mají také nevýhody, včetně toho, že: [7]

1. jsou řešeny iterativně, a proto může být zapotřebí více výpočtů,
2. nemusí být nalezeno přesné řešení,
3. jsou často nutné počáteční odhady řešení.

3 NUMERICKÉ METODY PRO ŘEŠENÍ ODR – S POČÁTEČNÍMI PODMÍNKAMI

Diferenciální rovnice n-tého řádu musí být doprovázena n pomocnými podmínkami, které jsou nutné k určení n integračních konstant, které vznikají obecným řešením. Pokud jsou podmínky poskytovány na stejné počáteční hodnotě nezávisle proměnné, máme problém s početní podmínkou.

Základním kritériem, dle kterého se metody dělí je počet a velikost kroku. Jednokrokové metody používají jeden předchozí bod k určení odhadu následující hodnoty. Vícekrokové metody pak používají k odhadu více předchozích bodů. V obou případech jsou všechny body na zkoumaném intervalu děleny ekvidistantně, tedy velikost kroku je stejná. U adaptabilních metod se pak upravuje délka kroku a to tak, aby byla splněna předem stanovená podmínka přesnosti. Základní typy metod můžeme rozdělit v následujícím schématu, viz obrázek 1.



Obrázek 1 Základní dělení a příklady metod

Jednoduchá rovnice prvního řádu s počáteční podmínkou lze formulovat rovnicí 3.0.

$$y' = f(x, y),$$

$$y(a) = y_0,$$

$$x_0 = a \leq x \leq b = x_n \tag{3.0}$$

Kde y_0 je předepsaná počáteční podmínka, nezávisle proměnná x představuje hodnotu v rozmezí na daném intervalu [a, b], Tento interval je rozdělen do n segmentů (kroků) většinou stejné délky značené h, proto platí rovnice 3.1.

$$x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots, x_n = x_0 + nh \tag{3.1}$$

Řešení v bodě x_0 je dáno počáteční podmínkou. Cílem je nalézt odhady v po sobě jdoucích bodech $x_0, x_1, \ldots, x_n$.

3.1 Jednokrokové metody s konstantním krokem

Jednokrokové metody hledají řešení odhadem y_{i+1} v bodě x_{i+1} extrapolací odhadu řešení y_i z posledního bodu x_i. Přesně jak tento nový odhad je extrapolován z předchozího odhadu závisí na konkrétní použité numerické metodě. Na obrázku 2 je vidět velmi jednoduchá jednokroková Eulerova metoda.

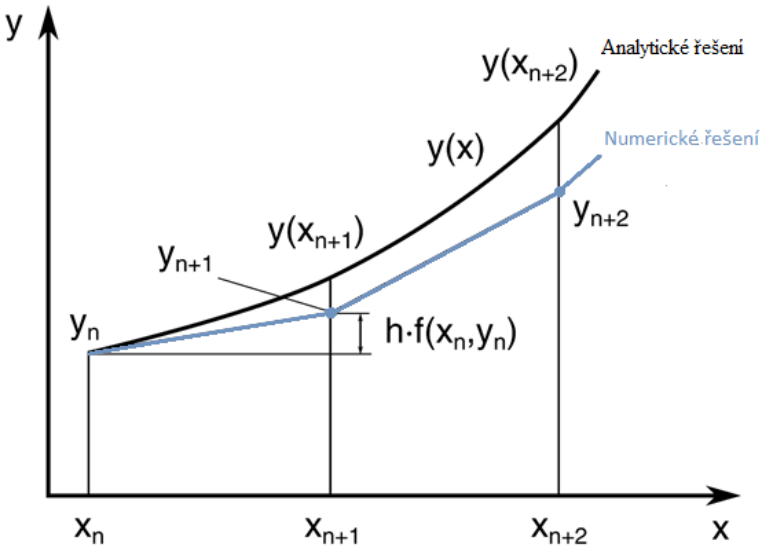
Obecná rovnice jednokrokové metody pak může vypadat dle rovnice 3.2.

$$y_{i+1} = y_i + h\varphi(x_i, y_i) \tag{3.2}$$

Kde hodnota φ je pak definována konkrétní numerickou metodou.

3.1.1 Eulerova metoda

Eulerova metoda je jednou ze základních metod v otázce řešení ODR. Její obliba spočívá především díky její jednoduchosti a snadné programovatelnosti, přestože pro její nízkou přesnost není příliš vhodná pro široké používání k řešení diferenciálních rovnic.



Obrázek 2 Eulerova metoda

S Eulerovou metodou (a mimo jiné i s některými jinými metody) se můžeme setkat jak v explicitním, tak implicitním tvaru. Na příkladu Eulerovy metody to znamená, že **Explicitní Eulerova** metoda někdy zvaná dopředná Eulerova metoda (z angl. „forward Euler method“) je jednokroková metoda pro řešení ODR prvního řádu. Metoda využívá předpokladu, že pro krátkou vzdálenost h, funkce y(x) má konstantní sklon roven sklonu v bodech (x_i, y_i), tímto předpokladem je pak dán následující bod (x_{i+1}, y_{i+1}) pomocí rovnice 3.3. [1]

$$y_{i+1} = y_i + hf(x_i, y_i) \tag{3.3}$$

Chyba, respektive rozdíl mezi získaným a analytickým řešením v bodě x je pak „ignorována“. Velikost této chyby je pak přímo úměrná velikosti zvoleného kroku. [12]

Implicitní Eulerova metoda, někdy zvaná zpětná (z angl. „backward Euler method“) využívá stejného předpokladu. Vzorec pro výpočet se mírně liší, viz 3.4. [1]

$$y_{i+1} = y_i + hf(x_{i+1}, y_{i+1}) \tag{3.4}$$

Neznámá y_{i+1} se nyní vyskytuje na obou stranách rovnice a není jednoduché, nebo spíše možné řešit tuto rovnici explicitně, řešení také může obsahovat více správných řešení v závislosti na funkci f(x,y).

3.1.2 Heunova metoda

Heunova metoda je další jednoduchou metodou, vychází z Eulerovy metody, proto se někdy nazývá druhou modifikací Eulerovy metody. Poskytuje přesnější aproximaci řešení. Základní myšlenkou je snížit chybu původní Eulerovy metody, lze ji zařadit mezi jednu z nejjednodušších metod typu **prediktor-korektor** [3] (tato třída metod bude zmíněna dále v kapitole 3.3.3). V prvním kroku využíváme **Eulerovy explicitní metody** k predikci hodnoty (rovnice 3.5), v druhém kroku využíváme **implicitní Eulerovy metody** a použití lichoběžníkového pravidla (rovnice 3.6), kde dosazujeme hodnotu získanou z předchozího výpočtu ke korekci, a tedy snížení chyby. Kombinace implicitní a explicitní metody se u třídy metod prediktor-korektor běžně využívá. [4]

$$y_{pred} = y_i + hf(x_i, y_i) \tag{3.5}$$

$$y_{i+1} = y_i + \frac{h}{2}\left(f(x_i, y_i) + f\left(x_{i+1}, y_{pred}\right)\right) \tag{3.6}$$

3.1.3 Taylorova řada

Taylorova řada je mocninná řada, Taylorovým rozvojem získáváme způsob, jak najít hodnotu funkce v blízkosti známého bodu, tj. bodu, kde je hodnota funkce známa. Funkce je reprezentována součtem členů konvergentní řady. V některých případech (pokud je funkce polynomem) může Taylorova řada poskytnout přesnou hodnotu funkce. Ve většině případů je však pro přesnou hodnotu nutný součet nekonečného počtu členů. Pokud je použito pouze několik členů, je hodnota funkce, kterou získáme z Taylorovy řady, aproximací. Taylorova řada se hojně používá v numerických metodách. [12]

Zatímco Eulerova metoda je metoda prvního řádu. Lokální chyba metody se dá vyjádřit jako $O(h^2)$ a globální chyba metody pak jako O(h). Kdežto v závislosti na řádu má Taylorova metoda lokální chybu metody $O(h^{k+1})$ a globální chybu metody $O(h^k)$. Proto čím vyšší řád

Taylorovy metody, tím přesnější odhad řešení dostaneme. Nicméně toto snížení chyby vyžaduje derivace funkce f(x.y), což je zjevná nevýhoda.

Taylorova řada je součet funkcí založený na neustále rostoucích derivacích. Pro funkci f(x), která závisí pouze na jedné nezávislé proměnné x. Hodnotu funkce v bodě $x_0 + h$ získáme aproximací s Taylorovy řady viz rovnice 3.7.

$$f(x_0 + h) = f(x_0) + hf''(x_0) + \frac{h^2}{2!}hf''(x_0) + \frac{h^3}{3!}hf^{(3)}(x_0) + \cdots + \frac{h^n}{n!}hf^{(n)}(x_0) + R_{n+1} \tag{3.7}$$

Teoreticky nám tedy Taylorova řada může poskytnou řešení velice přesné, nicméně protože výpočet je komplikovaný, a tedy i časově náročný, k řešení úloh s počáteční podmínkou, není příliš efektivní. Přesto však z ní vychází spousta jiných excelentních metod. [3]

3.1.4 Metody Runge-Kutta

Metody typu Runge-Kutta (dále někdy zkráceně „RK“) se také řadí mezi jednokrokové metody, využívající znalosti Taylorova rozvoje. Tyto metody generují odhad řešení se stejnou přesností jako Taylorova metoda, velkou výhodou však je, že nevyžadují výpočet zmíněných derivací. [12]

Obecný vzorec pro výpočet odhadu je pak dán rovnicí 3.8. Kde zmíněná hodnota sklonu v daném kroku φ (rovnice 3.1) je inkrementální funkce. Počet bodů, které jsou použity pro tento výpočet určuje řád metody, např. RK metoda druhého řádu využívá 2 body v každém sub-intervalu k výpočtu sklonu, který se následně použije pro další odhad řešení.

Obecný matematicky popis je dán rovnicí: [1]

$$y_{i+1} = y_i + h\sum_{j=1}^{n} B_j k_j \tag{3.8}$$

Kde hodnoty k jsou pak dány rovnicemi 3.9.

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + hC_2, y_i + h(A_{21}k_1))$$

$$k_3 = f(x_i + hC_3, y_i + h(A_{31}k_1 + A_{32}k_2))$$

$$\vdots$$

$$k_n = f(x_i + hC_n, y_i + h(A_{n1}k_1 + A_{n2}k_2 + \cdots + A_{n,n-1}k_{n-1})) \qquad (3.9)$$

Kde konkrétní hodnoty konstant A, B a C jsou definovány rovněž dle zvoleného řádu metody, jak bude uvedeno níže.

Vektor C označuje polohy jednotlivých fází v daném kroku. Matice A označuje závislost jednotlivých fází na derivátech nalezených v předešlých fázích. A nakonec B je vektor kvadraturních vah, udávajících, jak výsledný odhad závisí na hodnotách vypočítaných v jednotlivých fázích. [2]

Tyto konstanty se pak přehledně uvádějí ve formě následující tabulky, která se někdy nazývá Butcherovo tablo, viz tabulka 1.

0				
C_2	A_{21}			
C_3	A_{31}	A_{32}		
⋮	⋮			
C_n	A_{n1}	A_{n2}	…	$A_{n,n-1}$
	B_1	B_2	…	B_m

Tabulka 1 Obecné Butcherovo tablo

Metoda RK druhého řádu nemá unikátní sadu těchto konstant, proto existuje více variant této metody. Například pro metodu středního bodu (angl. „Mid-point method“) pak hodnoty těchto konstant vypadají dle tabulky 2. [2]

C	A	
0		
1/2	1/2	
B	0	1

Tabulka 2 Butcherovo tablo pro metodu středního bodu

Zmíněná Heuneova (někdy též zvaná modifikovaná/vylepšená Eulerova metoda) tabulka 3. [2]

C	A	
0		
1/2	1/2	
B	1	1

Tabulka 3 Butcherovo Tablo Heunovu metodu

Ralstonova metoda tabulka 4. [2]

C	A	
0		
2/3	2/3	
B	1/4	3/4

Tabulka 4 Butcherovo tablo Ralstonovu metodu

Metoda RK třetího řádu principiálně stejný postup jak v předchozím případě, konstanty dle tabulky 5. [2]

C	A		
0			
1/2	1/2		
1	-1	2	
B	1/6	2/3	1/6

Tabulka 5 Butcherovo tablo metodu Runge-Kutta třetího řádu

Metoda RK čtvrtého řádu se někdy také nazývá jako metoda klasický Runge-Kutta. Klasická RK metoda pak používá následující konstanty dle tabulky 6. [2]

C	A			
0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
B	1/6	1/3	1/3	1/6

Tabulka 6 Butcherovo tablo metodu Runge-Kutta čtvrtého řádu

Tato metoda poskytuje odhady s přesností Taylorovy metody čtvrtého řádu bez nutnosti výpočtu derivací funkce f(x,y). Namísto toho je potřeba provést čtyři ohodnocení v každém kroku. Nazývá se klasická, protože obecně vzato je nejvíce používanou technikou pro numerické řešení ODR s počáteční podmínkou, poskytuje dobrou, respektive akceptovatelnou rovnováhu mezi přesností odhadu a výpočetními nároky. [12]

3.2 Jednokrokové metody s adaptivním krokem

Doposud zmíněné metody využívaly pro aproximaci každého následujícího bodu řešení rovnice konstantní velikost kroku. V každém případě je tedy možnost zvýšit přesnost řešení pomocí snížení velikosti tohoto kroku, nicméně menší krok v důsledku znamená významné zvýšení celkového počtu operací. Ovšem existuje alternativní způsob, jak zvýšit přesnost, a to způsobem, kdy na místo konstantní velikosti kroku je snaha nalézt v každé iteraci takový nejvyšší možný krok, který ovlivní globální chybu metody pouze v rámci předem stanovené tolerance. Nicméně globální chyba metody není známa, pokud neznáme exaktní řešení. Proto optimální velikost kroku musí být stanovena na základě lokální chyby metody. [12]

3.2.1 Metoda Runge-Kutta Fehlberg

Jak bylo zmíněné pro určení optimální velikosti kroku se využívá odhadu lokální chyby metody. V případě metod Runge-Kutta je pro tento účel možno využít dvou metod RK různého řádu, kdy odhad této chyby odhadujeme jejich odečtem. Nevýhodou tohoto způsobu však je množství funkcí, které by tak v každé iteraci musely být vypočítány. Například při použití nejběžněji využívané metody Runge-Kutta čtvrtého řádu a pátého řádu musíme dohromady počítat celkem ohodnocení 10 funkcí v každé iteraci (čtyři v případě RK čtvrtého řádu a šest v případě pátého řádu). Tento nevýhodný požadavek optimalizuje metoda Runge-Kutta Fehlberg, ta používá metodu RK5, která využívá vyhodnocení funkcí poskytované její doprovodnou metodou RK4. Z toho důvodu je metoda také běžně známá pod zkratkou RKF45. Zmíněným způsobem se sníží počet potřebných vyhodnocovacích funkcí z 10 na 6. [12]

Přesněji, odhad řešení s přesností čtvrtého řádu je dán vztahem 3.10.

$$y_{i+1}^{(4)} = y_i + h(B_1^* k_1 + B_2^* k_2 + B_3^* k_3 + B_4^* k_4 + B_5^* k_5) \tag{3.10}$$

Zatímco odhad s přesností pátého řádu je dán vztahem 3.11.

$$y_{i+1}^{(5)} = y_i + h(B_1k_1 + B_2k_2 + B_3k_3 + B_4k_4 + B_5k_5 + B_6k_6) \tag{3.11}$$

Kde pro výpočet koeficientů k lze použít obecný vzorec 3.7, koeficienty A, B a C jsou pak dány následovným Butcherovým tablem, viz tabulka 7. [2]

C	A					
0						
1/4	1/4					
3/8	3/32	9/32				
12/13	1932/2197	-7200/2197	7296/2197			
1	439/216	-8	3680/513	-845/4104		
1/2	-8/27	2	-3544/2565	1859/4104	-11/40	
B	16/135	0	6656/12825	28561/56430	-9/50	2/55
B*	25/216	0	1408/2565	2197/4104	-1/5	0

Tabulka 7 Butcherovo tablo pro RKF45

Kde pro obě rovnice se používají stejné hodnoty koeficientů k1, k2 až k6.

Hodnoty odhadů se pak využívají pro výpočet lokální chyby metody dle vztahu 3.12.

$$Error = \frac{1}{h}\left|y_{i+1}^{(4)} - y_{i+1}^{(5)}\right| \tag{3.12}$$

Úpravou lze získat vzorec, pomocí kterého pak odpadá nutnost přímo počítat odhad pátého řádu (rovnici 3.11), a tedy i nutnost použití vektoru B. Výsledná rovnice pak vypadá dle 3.13.

$$Error = \frac{1}{h}\left|\frac{1}{360}K_1 - \frac{128}{4275}K_3 - \frac{2167}{75240}K_4 + \frac{1}{50}K_5 + \frac{2}{55}K_6\right| \tag{3.13}$$

V každé iteraci je tedy dán odhad s přesností čtvrtého řádu, kde lokální chyba metody je pak dána rovnicí 3.12 respektive 3.13. Pokud je chyba v rámci tolerovaných mezí, které se předem definují, pak odhad čtvrtého řádu je přijatý jako řešení s daným krokem a jsou poté

použity pro výpočet dalšího kroku. V opačném případě je velikost kroku upravena, dokud podmínka tolerance není splněna.

Úprava velikosti kroku se pak obecně počítá v závislosti na řádu metody. Metoda řádu p + 1 odhaduje řešení s přesností řádu p. V každém kroku se pak tedy upravuje velikost kroku následovně pomocí vztahu 3.14. [12]

$$h_{adj} = \alpha \left[\frac{\varepsilon h}{\left| y_{i+1}^{(p)} - y_{i+1}^{(p+1)} \right|} \right]^{\frac{1}{p}} = \alpha \left[\frac{\varepsilon h}{Error} \right]^{\frac{1}{p}} \tag{3.14}$$

Kde $\alpha \leq 1$ je vhodný přizpůsobovací faktor a ε je požadovaná tolerance. Hodnotu α je vhodné volit spíše konzervativním způsobem, tzn. konkrétně pro RKF45 metodu se doporučuje na základě experimentů a investigací volit hodnotu $\alpha = 0{,}84$, tzn. pro RKF45 pak platí po dosazení rovnice 3.15. [12]

$$h_{adj} = 0{,}84 \left[\frac{\varepsilon h}{error} \right]^{\frac{1}{4}} \tag{3.15}$$

Jak bylo zmíněno, rozdíl odhadů čtvrtého a pátého řádu ve jmenovateli zlomku v rovnici 3.14 pak dává lokální chybu metody. Pro zajištění rozumné míry výpočetní přesnosti k počtu kroků ohodnocení je možné předem definovat povolenou hodnotu pro minimální krok h_{min} a pro maximální krok h_{max}.

Pokud h_{adj} splňuje definované meze, pak je tato hodnota přijata a použita pro následující krok.

V případě, kdy h_{adj} je vetší než h_{max} pak $h_{adj} = h_{max}$, v opačném případě je algoritmus neúspěšně ukončen, protože není současně možné splnit zadanou toleranci při definovaném minimálním kroku. [12]

3.3 Vícekrokové metody

Na rozdíl od jednokrokových metod, které využívají pouze jednu předchozí hodnotu k aproximaci následné hodnoty, vícekrokové metody aproximují numerické hodnoty řešení použitím více než jedné předchozí hodnoty. Vícekrokové metody proto mohou často dosáhnout větší přesnosti než jednokrokové metody, které používají stejný počet vyhodnocení funkce,

protože využívají většího počtu informací o známé části řešení než jednokrokové metody. Zvláštní kategorií vícekrokových metod jsou lineární vícekrokové metody, kde je numerické řešení ODR v určitém místě vyjádřeno jako lineární kombinace hodnot numerického řešení a hodnot funkcí v předchozích bodech. [6]

Z principu této vlastnosti však nelze vícekrokové metody startovat samostatně k dosažení prvního odhadu, protože v ten okamžik je pro nás známa pouze počáteční podmínka, proto pro start musí být použita libovolná jednokroková metoda jako např. klasická RK.

Vícekrokové metody můžou být jak explicitní, tak implicitní. Explicitní metody využívají explicitní vzorec pro výpočet odhadu. Pro příklad, vícekroková explicitní metoda třetího řadu využívá 2 předchozí kroky, obecný tvar pak pro výpočet odhadu je dán vztahem 3.20.

$$y_{i+1} = F(x_i, y_i, x_{i-1}, y_{i-1}, x_{i-2}, y_{i-2}) \tag{3.20}$$

Tímto způsobem jsou na pravé straně rovnice pouze hodnoty, které známe. V případě implicitní metody třetího řádu je potřeba znát pouze jedné hodnoty z předchozího kroku, nicméně neznámý odhad y_{i+1} je zahrnutý na obou stranách rovnice, viz rovnice 3.21.

$$y_{i+1} = F(x_{i+1}, y_{i+1}, x_i, y_i, x_{i-1}, y_{i-1}) \tag{3.21}$$

3.3.1 Metoda Adams-Bashforth

Metoda Adams-Bashforth je explicitní vícekroková metoda, k získání odhadu řešení y_{i+1} s použitím znalosti dvou nebo více odhadů z předchozích kroků. Počet předchozích bodů, které musíme znát pro použití metody je dán řádem metody, tzn. pro Adams-Bashforth metodu druhého řádu je potřeba pro interpolaci znát hodnotu v x_i a jednu předchozí hodnotu v x_{i-1} a pak analogicky pro vyšší řády metody.

Adams-Bashforth metody mají také tendenci mít malé oblasti absolutní stability, což inspirovalo konstrukci implicitních Adamsových metod (tzv. metod Adams-Moulton), které jsou popsány v následující kapitole. [12]

Obecně lze metodu Adams-Bashforth vyjádřit vztahem 3.22.

$$y_{i+1} = y_i + hC \sum_{j=1}^{p} B_j f(x_{i+1-j}, y_{i+1-j}) \tag{3.22}$$

Kde p je řád metody a pro j pak platí $0 \leq j \leq (p-1)$.

Prvním řádem je klasická jednokroková Eulerova explicitní metoda, pro výpočet vyšších řádů se používají koeficienty z tabulky 8.

p	C	B			
1	1	1			
2	1/2	3	-1		
3	1/12	23	-16	5	
4	1/24	55	-59	37	-9

Tabulka 8 Tabulka koeficientů řádu p pro výpočet metody AB

Pro AB metodu čtvrtého řádu pak bude rovnice po dosazení vypadat dle 3.23.

$$y_{i+1} = y_i + \frac{h}{24}(55f(x_i, y_i) - 59f(x_{i-1}, y_{i-1}) + 37f(x_{i-2}, y_{i-2}) - 9f(x_{i-3}, y_{i-3})) \tag{3.23}$$

Pro AB metodu pak platí, že musí být inicializována jednokrokovou metodou, výpočtem bodů, který je roven vztahem p-1.

3.3.2 Metoda Adams-Moulton

Analogicky metoda Adams-Moulton je implicitní vícekrokovou metodou k odhadování řešení y_{i+1}. Stejně jako v předchozím případě tak i zde platí, čím vyšší řád metody, tím víc hodnot z předchozích kroků musí být použito. Na rozdíl od AB metody však při stejném řádu stačí znát o jeden předchozí bod méně. Dále AM metody používají menší konstanty, vykazují větší oblasti stability a zahrnují nižší zaokrouhlovací chybu než jejich Adams-

Bashforth protějšky (stejného řádu). Metody AM lze popsat následovně popsat dle rovnice 3.24. [12]

$$y_{i+1} = y_i + hC \sum_{j=1}^{p} B_j f(x_{i+2-j}, y_{i+2-j}) \tag{3.24}$$

Pro výpočet AM metody se pak používají tyto konstanty koeficienty z tabulky 9.

p	C	B			
1	1	1			
2	1/2	1	1		
3	1/12	5	8	-1	
4	1/24	9	19	-5	1

Tabulka 9 Tabulka koeficientů řádu p pro výpočet metody AB

Po dosazení pro AM čtvrtého řádu pak dostaneme rovnici 3.25

$$y_{i+1} = y_i + \frac{h}{24}(9f(x_{i+1}, y_{i+1}) + 19f(x_i, y_i) - 5f(x_{i-1}, y_{i-1}) + f(x_{i-2}, y_{i-2})) \tag{3.25}$$

Jak bylo zmíněno metoda Adams-Moulton poskytuje lepší výsledky než explicitní Adams-Bashforth metoda stejného řádu. Ačkoli tomu tak obecně je, implicitní metody mají tu slabinu, že je třeba metodu nejprve algebraicky převést na explicitní reprezentaci pro y_{i+1}. Tento postup však není vždy možný. Proto se implicitní metody prakticky používají v metodách třídy prediktor-korektor.

3.3.3 Metody Prediktor-Korektor

Metody prediktor-korektor je speciální třída metod a technik, které využívají kombinaci explicitních a implicitních formulí k řešení ODR.

V prvním kroku je využito explicitní rovnice k predikování hodnoty y_{i+1}. Tato predikovaná hodnota je poté použita ke zlepšení aproximace v implicitní rovnici k získání nového, více

přesného řešení. Nejjednodušší takovou metodou je Heunova metoda, která byla představena v kapitole 3.1.2, další takovou metodou je například metoda Adams-Basfort-Moulton.

Metoda prediktor-korektor Adams-Bashforth-Moulton čtvrtého řádu používá rovnici AB4 řádu jako prediktor viz 3.26 a AM4 jako korektor viz 3.27.

$$y_{pred} = y_i + \frac{1}{24}h\big(55f(x_i, y_i) - 59f(x_{i-1}, y_{i-1}) + 37f(x_{i-2}, y_{i-2}) - 9f(x_{i-3}, y_{i-3})\big) \tag{3.26}$$

$$y_i = y_i + \frac{1}{24}h\left(9f\big(x_{i+1}, y_{pred}\big) + 19f(x_i, y_i) - 5f(x_{i-1}, y_{i-1}) + f(x_{i-2}, y_{i-2})\right) \tag{3.27}$$

V metodách třídy prediktor se pak někdy využívá toho, že se korekční krok opakuje, dokud není splněna podmínka předem stanovené tolerance. U metody ABM je však obvykle efektivnější neprovádět tyto iterace a namísto toho volit menší velikost kroku, pokud je potřeba zvýšit přesnost. [12]

3.4 Zhodnocení dostupných nástrojů využívajících numerické metody

Aktuálně již existuje pár řešení, které můžou sloužit k řešení ODR a demonstrovat principy numerických metod. Subjektivním zhodnocením za nejvíce relevantní lze považovat řešení společnosti Wolfram [19]. K dispozici je řada numerických metod, je možnost řešit ODR vyšších řádů. Nástroj zobrazuje symbolicky první iteraci řešení. Poskytne i analytické řešení (jestli jej najde), stabilní oblast metody a srovnání globální chyby metody s jinými metody. Tento nástroj je poměrně pokročilý, chybí však přehlednější graf, není možnost srovnat různé metody v jednom grafu a také chybí vícekrokové metody. Další obsah je však možné zpřístupnit v placené části obsahu.

K dispozici jsou i další nástroje, ty už ale zdaleka nenabízí tolik možností, jako např. CodeSansar [20]

II. PRAKTICKÁ ČÁST

4 ÚVOD DO PRAKTICKÉ ČÁSTI

Součástí praktické částí bylo navržení interaktivní webové aplikace, jejíž cílem je na názorných příkladech demonstrovat principiální funkčnost numerických metod, tedy představení praktického použití metod, které byly popsány v teoretické části. Cílem tedy je usnadnit porozumění tomu, jak tyto metody fungují.

V kapitole 5 je popsán nejdříve návrh této aplikace, tedy zvolené technologie a nástroje, které byly použity při implementaci samotných numerických metod a poté technologie s jejíchž pomocí byla aplikace nasazena na web. Poté je formou dokumentace vysvětlen základní popis, funkčnost a struktura aplikace.

V kapitole 6 je vybrán reálný systém, podle kterého jsou verifikovány jednotlivé numerické modely a podle, kterého pak dojde k porovnání vybraných numerických metod. V závěru práce pak na základě těchto informací budou zhodnoceny metody pro jejich reálné nasazení.

5 NÁVRH A IMPLEMENTACE APLIKACE PRO ŘEŠENÍ ODR

Základem celého systému je implementace knihovny vybraných numerických metod, které byly představeny v teoretické části. Tato knihovna je součástí přílohy PI. Nástavbou jsou pak použité systémy sloužící pro manipulaci s dosaženými výsledky a jejich vizualizaci a systémy pro vytvoření webového rozhraní, přes které uživatel k aplikaci přistupuje.

Aplikace je dostupná na adrese: https://num-methods.herokuapp.com/

5.1 Návrh interaktivní aplikace

Interaktivní aplikace jsou vhodnými prostředky pro zobrazování dat, kdy uživatel má možnost a určitou kontrolu vybrat si konkrétní věc, která je momentálně předmětem jeho zájmu a kterou daná aplikace umožňuje. Právě proto se podobné aplikace často využívají k podpoře výuky. Pro vyučujícího představují podporu výkladu. Pro studenta nabízí interaktivní aplikace možnost vizuálně zobrazit probírané téma a usnadnit tak pochopení dané problematiky.

5.2 Programovací jazyk Python

Python je vysokoúrovňový interpretovaný univerzální programovací jazyk. Jedná se o tzv. hybridní jazyk, což znamená, že je multi-paradigmatický, podporuje objektově orientované programování i procedurální a částečně funkcionální a mnoho dalších paradigmat je pak podporovaných různými rozšířeními. Klasický Python je napsaný v C, z něhož pak vychází i syntaxe, která je do jisté míry zjednodušena. Dále na rozdíl od klasického C obsahuje správu paměti (z angl. tzv. „Garbage Collector“), dynamickou kontrolu typů, nepodporuje funkcionalitu ukazatelů, nicméně nevýhodou však je, že oproti C je pomalejší.

Jednoduchost Pythonu spočívá v samotné mantře, na kterou je kladen důraz (tzv.: „Zen of Python) [14]:

- Krása je lepší než ošklivost
- Explicitní je lepší než implicitní
- Jednoduché je lepší než složité
- Složité je lepší než komplikované
- Čitelnost se počítá
- Zvláštní případy nejsou dost zvláštní na to, aby porušovala pravidla

I proto se v dnešní době těší velké oblibě a řadí se mezi jeden z nejpoužívanějších programovacích jazyků. Z těchto důvodu byl tento programovací jazyk v aktuální verzi 3.8 zvolen pro implementaci zadaná aplikace.

5.3 Možnosti usnadnění výpočtů a prezentování dosažených dat

Protože je požadováno řešit obyčejné diferenciální rovnice vyšších řádů, tedy po zjednodušení systém diferenciálních rovnic prvního řádu, a protože sami některé numerické metody lze zjednodušit řešením pomocí operací s maticemi, byla využita knihovna NumPy.

Dále důležitým aspektem aplikace je prezentování dosažených výsledků. Cílem je vytvoření přehledné tabulky vypočítaných hodnot a grafického zobrazení průběhu těchto hodnot, ve kterém je možné interaktivní výběr zvolené metody. Pro splnění těchto kritérií byly zvoleny technologie Pandas a Altair.

5.3.1 Knihovna NumPy

NumPy je fundamentální balík pro vědecké výpočty v jazyce Python. Jedná se o knihovnu, která poskytuje objekt vícerozměrného pole, různé odvozené objekty (například maskovaná pole a matice) a řadu rutin pro rychlé operace s poli, včetně matematických, logických, manipulace s tvary, třídění, výběru, I/O, diskrétní Fourierovy transformace, základní lineární algebru, základní statistické operace, náhodné simulace a mnoho dalších. [15]

Výhodou použití je, že poskytuje optimalizace operací n-dimenzionálních polí homogenního typu, a to tak, že operace jsou prováděny kompilovaným kódem pro zvýšení výkonu. Další výhodou je, že maticové operace vedou k méně řádkům kódu, což obecně znamená menší riziko chyb.

5.3.2 Knihovna Pandas

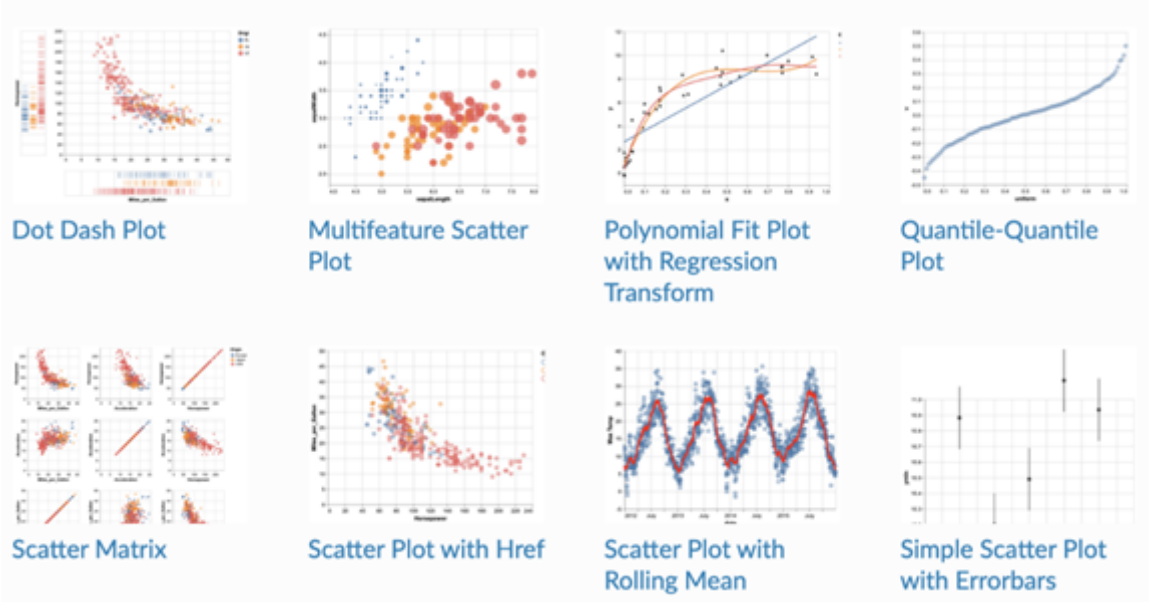
Pandas je open-source knihovna postavená nad numpy, která poskytuje vysoce výkonné a snadno použitelné datové struktury a nástroje pro analýzu dat v programovacím jazyce Python. Umožňuje rychlou analýzu a čištění a přípravu dat. Vyniká výkonem a produktivitou. Dokáže pracovat s daty z nejrůznějších zdrojů. Pandas se hodí pro mnoho různých druhů dat: tabulková data, data časových řad, libovolná maticová data s řádkovými a sloupcovými popisky a jakoukoli jinou formu souborů statistických dat. Dalším důvodem volby této knihovny je, že pro vizualizaci grafů je použit Altair jak bude níže uvedeno, který pracuje s datovým typem „DataFrame“, který Pandas nabízí. [16]

5.3.3 Knihovna Altair

Altair je deklarativní knihovna pro statistickou vizualizaci v jazyce Python, vychází z knihoven Vega a Vega-Lite, což jsou vizualizační gramatiky, které umožňují popsat vizuální vzhled a interaktivní chování vizualizace ve formátu JSON. Altair nabízí výkonnou a stručnou vizualizační gramatiku, která umožňuje rychle vytvářet širokou škálu statistických vizualizací.

Python již má k dispozici spoustu jiných nástrojů knihoven pro úkol vizualizace dat a mnoho z nich dokonce podporuje interaktivní vizualizaci. Altair se však odlišuje právě deklarativním stylem. U imperativních API se musí soustředit pozornost spíše na „jak něco udělat“ než na „co je potřeba dělat“. Je potřeba více přemýšlet o kódu potřebném k vytvoření vizualizace a ručně zadávat kroky k vykreslování, k limitám os, velikosti, rotaci, legendě atd.

Altair umožňuje soustředit svůj čas a snažit se více o svá data, pochopit je, analyzovat a vizualizovat, spíše než na kód, který je k tomu potřeba. To znamená, že je možno definovat data a výstup, který je očekáván (jak by měla nakonec vypadat vizualizace), a Altair se už o úkon postará automaticky a provede potřebné manipulace.



Obrázek 3 Příklady grafů vytvořených knihovnou

Klíčovou myšlenkou této knihovny je, že jsou deklarovány vazby mezi sloupci dat a kanály vizuálního kódování, jako jsou osa x, osa y, barva atd. Zbytek detailů grafu je zpracován automaticky. Na základě této deklarativní myšlenky vykreslování lze pomocí relativně

stručné gramatiky vytvářet překvapivou škálu jednoduchých až sofistikovaných grafů a vizualizací. Principiálně Altair používá k převodu kódu pythonu na grafiku Vega-Lite a to tak, že Altair v podstatě převádí kód na objekt JSON, který lze pak vykreslit jako PNG viz příklady na obrázku 3. [17]

5.4 Vytvoření interaktivního řídícího panelu pomocí Streamlit

Streamlit je open-source webový aplikační framework, který umožňuje velice snadno vytvářet a sdílet vlastní webové aplikace jak pro strojové učení, tak pro datové vědy.

Zjednodušeně řečeno, Streamlit je jednoduchý nástroj, který účinně pomůže vytvořit efektivní, přehledný a vysvětlující řídící panel tzv. „dashboard“. Pro práci s tímto nástrojem nejsou nutné žádné znalosti z oblasti front-end vývoje. Tento framework převede datové skripty na webovou aplikaci v několika řádcích kódu a také jen za několik sekund. Streamlit je také kompatibilní s celou řadou dalších knihoven, včetně zmíněných v předešlých kapitolách. [18]

Streamlit poskytuje řadu widgetů pro tvorbu interaktivní obsahu, např pro zobrazování textu nabízí, zobrazení textu v různých formátech (obyčejný text, nadpis, podnadpis, latex).

Dále jak bylo zmíněno je možné zobrazovat data ať už ve formátu tabulek, tak interaktivních grafů, zobrazování různých stavových hlášení, např. tzv. „progress spinnerů“ informující o probíhajícím výpočtu, chybových a varovných hlášení apod. Je také celá řada klasických widgetů na získávání vstupu od uživatele, jako tzv. „checkboxy“, číselné vstupy, textové vstupy, rozbalovací menu apod.

Dalšími možnostmi je tvoření struktury pomocí tzv. „layoutů“ neboli kontejnerů, která aplikaci člení do logických rámců, které obsahují různé prvky, může se jednat o boční panel, možnost dělit hlavní panel do sloupců s definovaným poměrem, normální kontejner a kontejner který lze expandovat nebo schovat.

Je možno také zpracovávat data z vlastního souboru nebo souboru nahraným uživatelem a uložit jej do perzistentní paměti, aby byl optimalizován běh aplikace.

5.5 Nasazení Streamlit aplikace na web

Existuje více možností, jak udělat aplikaci přístupnou skrz web. Jednou z nejpoužívanějších možností je použití systému Heroku.

Podmínkou je, že repositář aplikace musí být veřejně přístupný na GitHubu. Jakmile je provedeno, jsou k nasazení aplikace Streamlit na Heroku potřeba tyto tři soubory:

1. **Procfile –** informuje Heroku o typu aplikace (v tomto případě Python).
2. **requirement.txt** – obsahuje všechny knihovny potřebné pro aplikaci.
3. **setup.sh –** obsahuje informace o konfiguraci aplikace Streamlit.

Po jejich vytvoření je potřeba jít do systému Heroku a ve složce „Deploy“ se připojit na daný repositář GitHubu, pokud spojení proběhne v pořádku, Heroku si načte výše zmíněné konfigurační soubory, připraví si deklarované knihovny a poté ze zdrojových souborů sestaví webovou aplikaci.

5.6 Základní vlastnosti aplikace

Už při otevření je proveden analytický výpočet předem definovaného systému, za předem definovaných parametrů, průběh analytického výpočtu je vykreslen do grafu. Od tohoto okamžiku si uživatel může zvolit v bočním panelu z výběru definovaný systém, aplikace mu dynamicky podle konkrétního výběru nabídne parametry diferenciální rovnice, které lze měnit, jedná se např. o vstupní podmínky a pak v závislosti na fyzikální úloze parametry jako hmotnost, délka pružiny apod.

Uživatel má také možnost zobrazit si základní popis algoritmu jednotlivé numerické metody. Při rozbalení lišty na hlavním panelu, je možnost zvolit si jednotlivou metodu, která je v aplikaci implementována. Algoritmus je v pseudo-matematickém zápisu vygenerován ve formátu LaTex.

Další vlastností je zobrazení vybrané diferenciální rovnice, analytického řešení, definovaných parametrů a počátečních podmínek rovněž ve formátu LaTex.

Dalším parametrem, který lze měnit uživatelem je velikost intervalu, na kterém se daný výpočet provádí. Z toho důvodu, aby bylo uživateli nejprve ukázáno v grafu, jak vypadá průběh analytického řešení, pro který se rozhodne spustit výpočet numerickými metody. Pro provedení výpočtu numerickými metodami je potřeba kliknout na tlačítko „Calculate by numerical methods“, v tu chvíli se zahájí výpočet. Uživatel také může předem změnit počet kroků, který je omezen v rozsahu 5 až 100. Po zvolení počtu kroků se pod tímto vstupem vypíše výsledná velikost kroku.

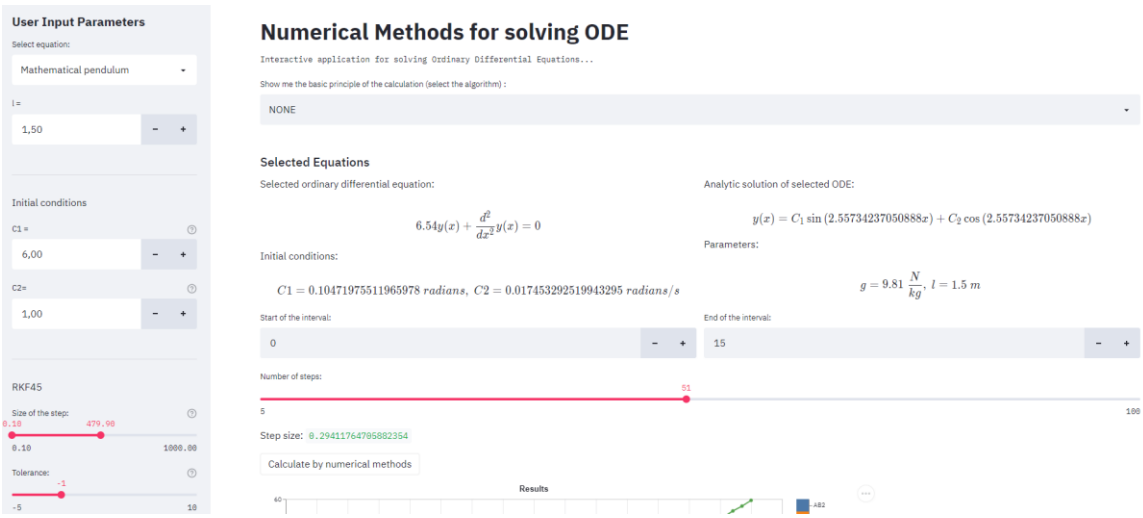
Během výpočtu je zobrazen informační status, který po dokončení zmizí, po úspěšném dokončení se průběhy numerických metod vykreslí do existujícího grafu, odkud je možné

filtrovat pouze tu metodu, která nás zajímá. Dále jsou výsledky pod grafem vepsány do tabulky, která má omezenou velikost (tzn. lze v ní „scrollovat“ až na konec, aniž by bylo potřeba sjet s celou webovou stránkou dolů a pak dlouze se vracet nahoru na graf. Tabulku je možné stáhnout ve formátu csv. Pod touto tabulkou je zobrazen další graf, který udává ve formě sloupcových grafů srovnání přesnosti jednotlivých metod formou sumy kvadratických odchylek jednotlivých metod.

5.7 Struktura aplikace

Na obrázku 4 je vidět uživatelské rozhraní stránky. Stránka je rozčleněna do 5 logických panelů:

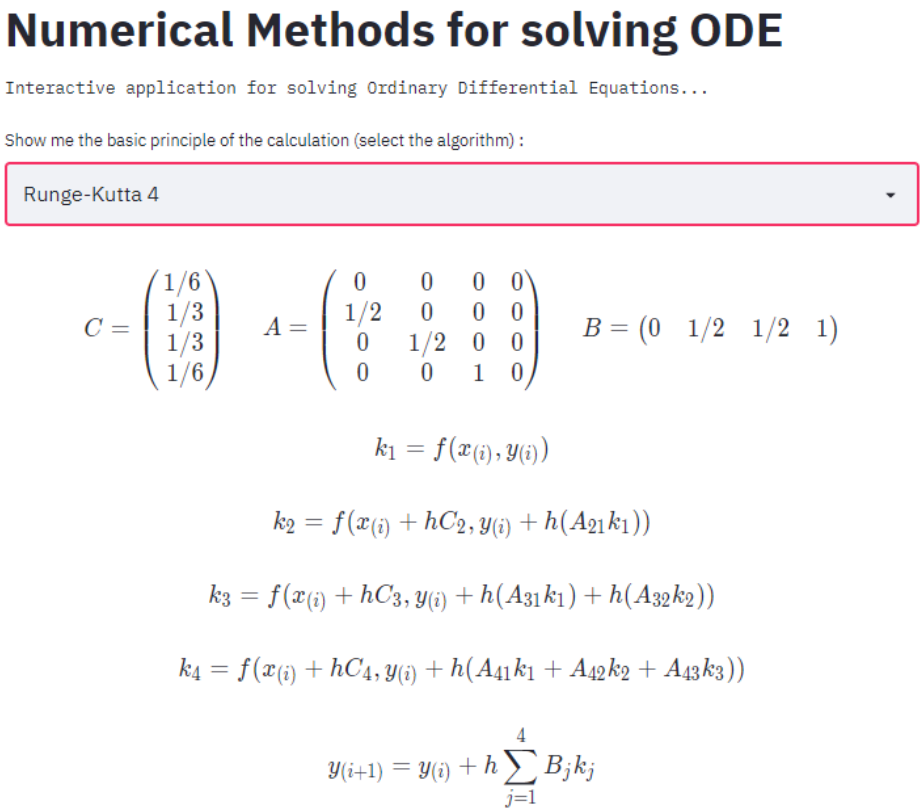
1. Hlavička a informační panel
2. Boční panel
3. Panel pro definování intervalu a počtu kroků
4. Panel výstupu aplikace
5. Rozcestník



Obrázek 4 Uživatelské rozhraní aplikace

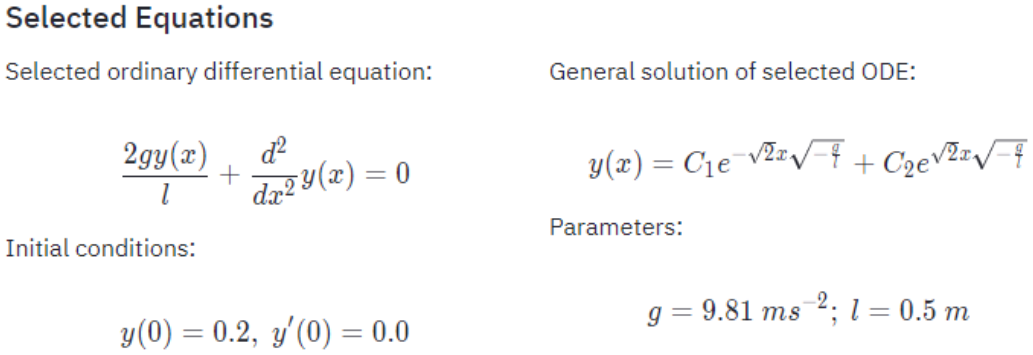
5.7.1 Hlavička a informační panel

V hlavní části hlavičky je nadpis a podnadpis aplikace, hned pod ním je rozbalovací lišta, kde si uživatel může vybrat jednu z použitých metod, která mu zobrazí postup algoritmu v symbolickém zápisu ve formátu LaTex. Na obrázku 5 je pak možné vidět zobrazení postupu pro metodu RK4.



Obrázek 5 Výběr metody RK4 se symbolickým popisem algoritmu

Pod touto rozbalovací lištou se nachází zvolená diferenciální rovnice a její obecné řešení, společně se zvolenými počátečními podmínkami a hodnoty veličin, které jsou v úloze použity, viz obrázek 6.

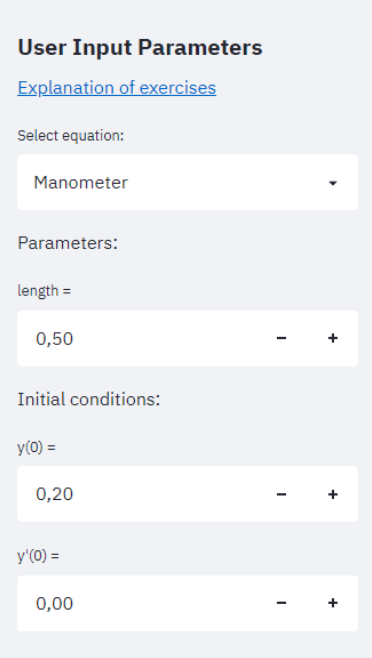


Obrázek 6 Přehled diferenciální rovnice, analytické řešení a parametry

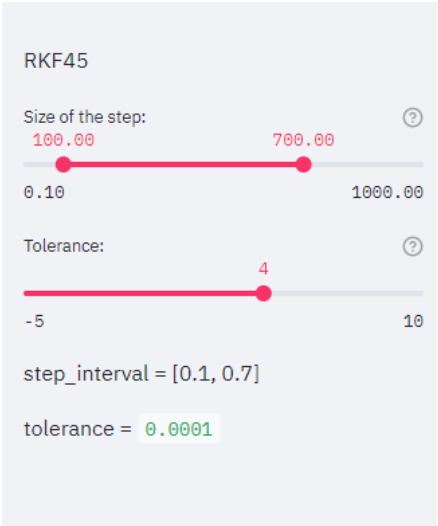
5.7.2 Boční panel

Boční panel obsahuje výběr úlohy, kterou aplikace nabízí. Hned pod tímto výběrem je také odkaz na stručný popis jednotlivých úloh, tento popis je součástí přílohy PII této práce. Po

výběru metody se dynamicky zobrazí vstupy pro parametrizaci úlohy, tedy daný počet počátečních podmínek k vybrané diferenciální rovnici a dále parametry, které daná úloha nabízí, např hmotnost apod. Současně s tím se i mění omezení pro daný vstup. Tento panel je možné vidět na obrázku 8, který tvoří vrch bočního panelu a spodní část na obrázku 7. V části na obrázku 7 je vidět interval velikosti kroku pro algoritmus RKF45, tedy hranici, kterou numerická metoda s adaptivní velikostí kroku nesmí překročit a pod tím přesnost, se kterou algoritmus adaptuje velikost kroku, oba vstupy jsou definovány formou posuvného jezdce, tedy mají svá definovaná omezení.



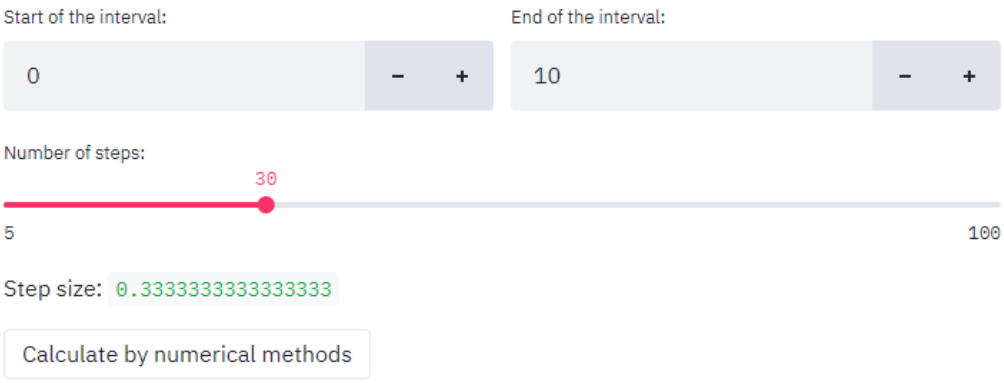
Obrázek 8 boční panel část 1



Obrázek 7 Boční panel část 2

5.7.3 Panel volby zkoumaného intervalu a počtu kroků pro numerické řešení

Pod informačním panelem se v hlavní části nachází možnost výběru intervalu, viz obr. 9, pro řešení zadané úlohy. Tento interval se volí číselně a není možné zadat konec intervalu menší, než je hodnota začátku. Numerické metody se aplikují pro řešení až po stisknutí tlačítka „Calculate by numerical methods“.

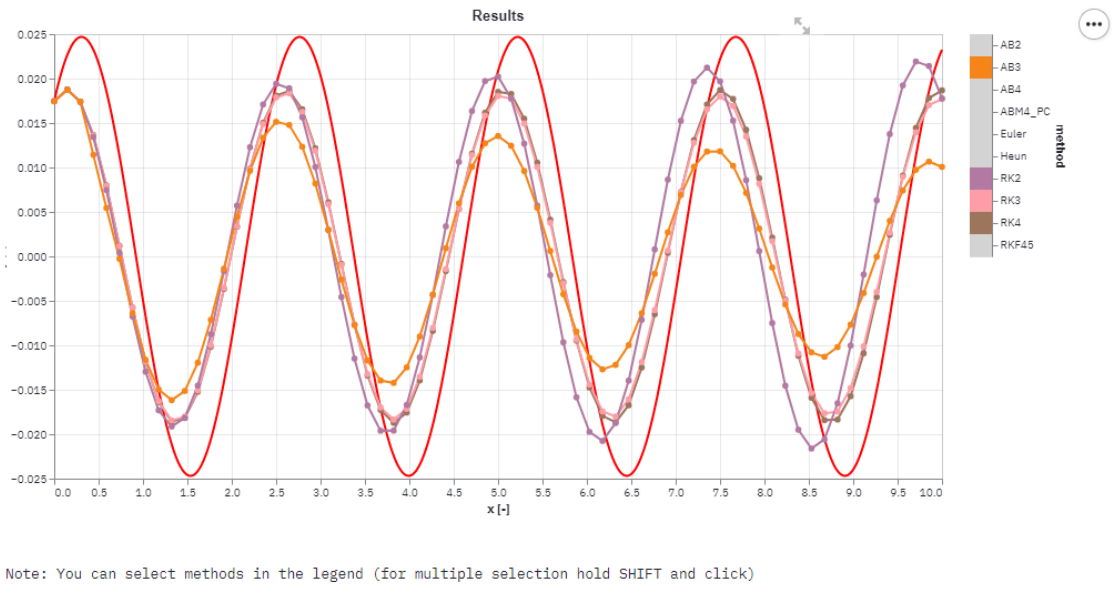


Obrázek 9 Volba intervalu řešení a počet kroků

5.7.4 Panel Výstupu aplikace

Výstup je rozdělen do několika částí:

1. **Grafické zobrazení průběhů** umožňuje vybírat jednotlivé metody a odfiltrovat tak tedy metody, které nebyly schopny při zadaných podmínkách řešit úlohu dostatečně přesně, graf automaticky přizpůsobí velikosti os, aby byl detail co největší. Vybírání je možné poklikem na danou metodu v legendě. Při stisku tlačítka „shift“ a poklikem na metody je pak možné vybrat vícero metod současně. Příklad grafu je na obrázku 10.



Obrázek 10 Grafické zobrazení průběhu hodnot

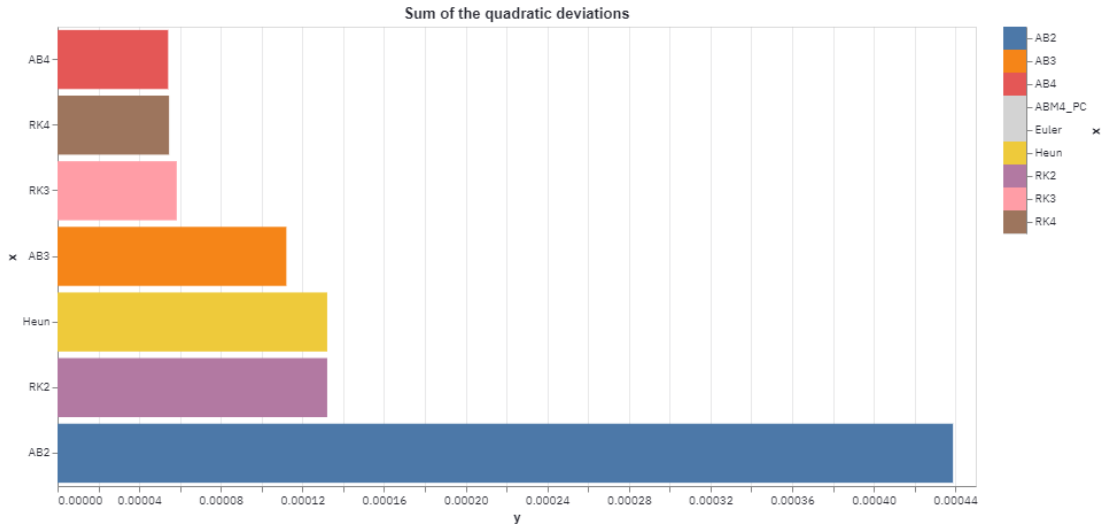
2. **Tabulka výsledných hodnot** se nachází hned pod grafem, je možné ji maximalizovat pro případ, kdy velikost přesáhne omezenou velikost na daném panelu. Dále je možné pak hodnoty exportovat do souboru formátu csv (viz obr. 11).

	Analytical	Euler	Heun	RK2	RK3	RK4	AB2	
0	0.0175	0.0175	0.0175	0.0175	0.0175	0.0175	0.0175	0
0.1471	0.0226	0.0200	0.0188	0.0188	0.0187	0.0187	0.0187	0
0.2941	0.0247	0.0201	0.0174	0.0174	0.0174	0.0174	0.0188	0
0.4412	0.0232	0.0174	0.0134	0.0134	0.0136	0.0136	0.0146	0
0.5882	0.0186	0.0118	0.0075	0.0075	0.0079	0.0080	0.0078	0
0.7353	0.0113	0.0038	0.0004	0.0004	0.0012	0.0012	-0.0003	-0
0.8824	0.0025	-0.0059	-0.0068	-0.0068	-0.0058	-0.0058	-0.0085	-0
1.0294	-0.0067	-0.0162	-0.0130	-0.0130	-0.0119	-0.0119	-0.0154	-0
1.1765	-0.0150	-0.0256	-0.0173	-0.0173	-0.0163	-0.0164	-0.0200	-0
1.3235	-0.0211	-0.0327	-0.0191	-0.0191	-0.0185	-0.0186	-0.0214	-0
1.4706	-0.0243	-0.0362	-0.0182	-0.0182	-0.0180	-0.0182	-0.0195	-0

Download csv file

Obrázek 11 Tabulka s výslednými hodnoty

3. **Graf porovnání metod** je zobrazen jako posledním výstupem aplikace a udává tedy průměrnou přesnost metody. Na pravé straně je opět možnost v legendě si vyselektovat metody, které chceme porovnat, může se totiž stát, že jedna metoda se odchýlí daleko více, což by pak mělo za následek, že ostatní metody by nebyly v grafu vůbec viditelné.



Obrázek 12 Příklad porovnání sumy jednotlivých kvadratických odchylek

5.7.5 Rozcestník

V posledním panelu aplikace nabízí pár zajímavých odkazů s podobnou problematikou a verzi aplikace knihovny streamlit.

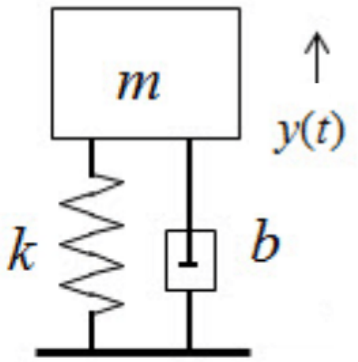
6 SIMULACE A ANALÝZA REÁLNÉHO SYSTÉMU

Pro účel verifikace implementace jednotlivých numerických metod a následného porovnání, je modelována soustava pružina-hmota-tlumič, která může odpovídat systému odpružení např. u motocyklu. Pro motokrosové jezdce jsou systémy odpružení motocyklů velmi důležité. Terénní tratě, na kterých jezdí, často zahrnují skoky a odpružení hraje velice důležitou roli pro pohodlí i bezpečnost jezdce a pro snížení negativního silového namáhání komponent.

6.1 Popis systému

Náš vztažný rámec definujme vzhledem k rámu motocyklu. Předpokládejme, že konec tlumiče je pevně připojený k rámu motocyklu. Měříme pak polohu kola vzhledem k rámu motocyklu. Dále definujeme, že směr vzhůru je kladný. Model našeho systému je ilustrován na obrázku 13.

Když je motocykl zvednut za rám, kolo volně visí a pružina není stlačená. To je přirozená poloha pružiny. Když je motocykl položen na zem a jezdec na něj nasedne, pružina se stlačí a systém je v rovnovážné poloze.



Obrázek 13 Soustava hmota-pružina-tlumič

Náš systém pak můžeme popsat diferenciální rovnicí 4.1.

$$my'' + by' + ky = 0 \tag{4.1}$$

Kde

m ... *hmotnost tělesa*

b ... *koeficient útlumu*

k ... *tuhost pružiny*

6.2 Modelový příklad

Předpokládejme, že hmotnost motocyklu je **101 kg**, hmotnost jezdce **95 kg**. Když jezdec nasedne na motocykl, odpružení se stlačí o **10 cm.** Definujme, že v této pozici je rovnovážný stav. Systém odpružení poskytuje tlumení **420krá**t větší, než je okamžitá vertikální rychlost motocyklu.

6.2.1 Počáteční úloha

Předpokládejme, že motocykl spolu s jezdcem dopadá na zem, cílem je popsat chování při dopadu motocyklu spolu s jezdcem na zem. Definujme tedy, že v čase **t = 0** dochází ke kontaktu kola se zemí. Pro určení počátečních podmínek polohy a rychlosti kola uvažujme, že kolo bylo před dopadem ve vzduchu, a tedy volně viselo tzn. pružina byla nestlačena, takže poloha je 10 cm pod rovnovážnou polohou vzhledem k rámu motocyklu. Vertikální rychlost motocyklu je 3 ms^{-1}, respektive **-3 ms^{-1}** protože se jedná o pohyb směrem k zemi,

$$mg = ks \tag{4.2}$$
$$196 = 0{,}1k$$
$$k = 1960$$
$$\mathrm{c} = 420$$
$$M = mg$$
$$m = \frac{M}{g} = \frac{196}{9{,}8} = 20$$

Po výpočtu rovnic 4.2 a následném dosazení do rovnice 4.1. pak dostáváme diferenciální rovnici 4.3.

$$20y'' + 420y' + 1960(x)y = 0 \tag{4.3}$$

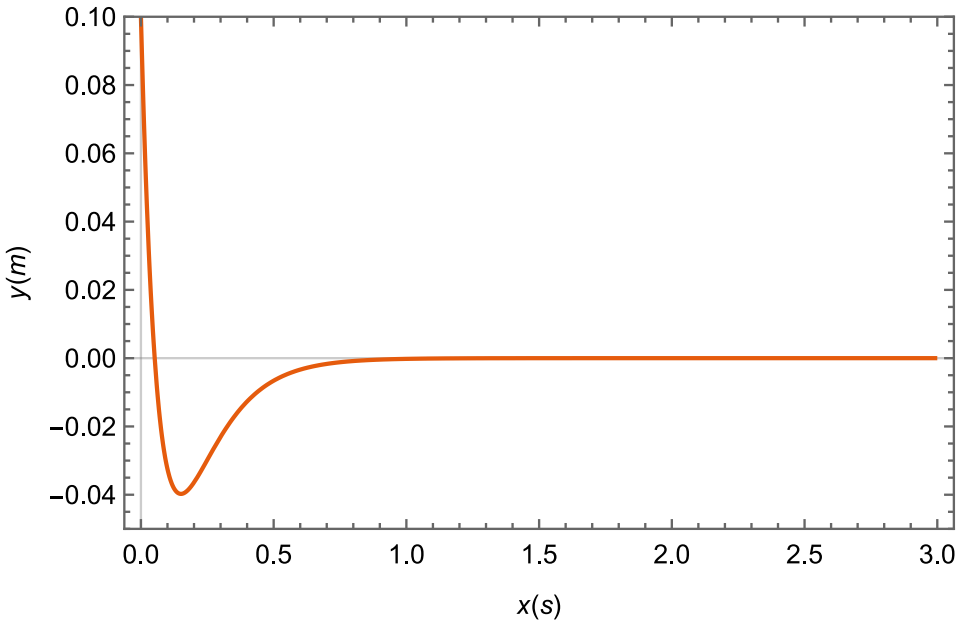
A úpravou pak rovnici 4.4

$$y''(t) + 21y'(t) + 98y(t) = 0 \tag{4.4}$$

Obecné řešení této rovnice pak je popsáno rovnicí 4.5

$$y(t) = -\frac{16}{70}e^{-7t} + \frac{23}{70}e^{-14t} \tag{4.5}$$

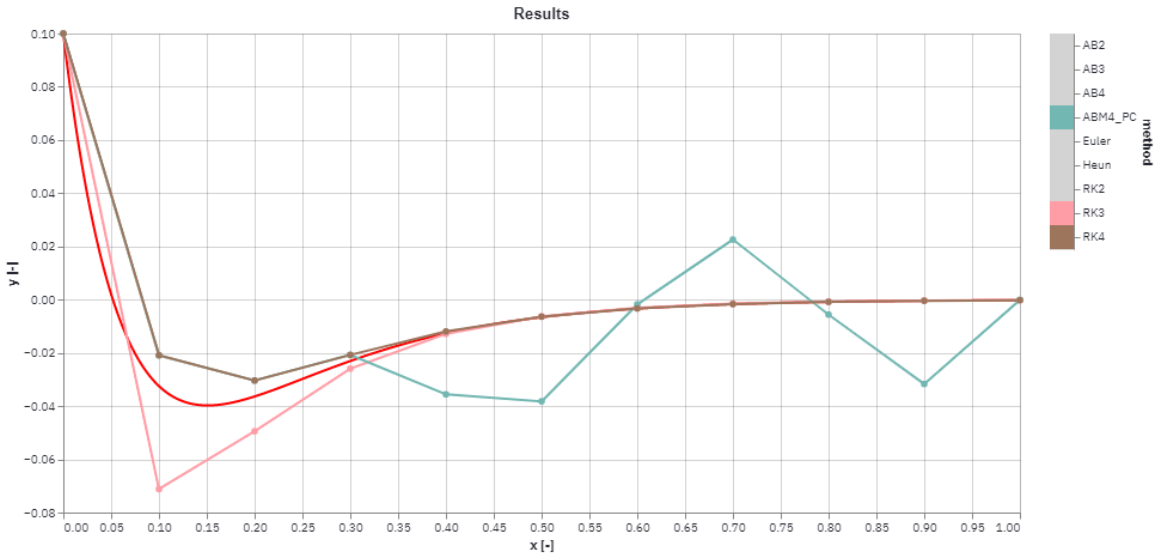
Graf pohybové rovnice y(t) pak vypadá dle obrázku 14.



Obrázek 14 Graf pohybové rovnice v čase

6.2.2 Řešení pomocí numerických metod

K řešení definovaného modelu použijeme vytvořený nástroj pro řešení ODR pomocí numerických metod. Na obrázku 14 je vidět, že hodnota se ustálila někde v čase t = 1 s. Interval, na kterém budeme tedy zkoumat zvolené numerické metody, nechť je x ε [0, 1]. Je zvoleno 10 kroků výpočtu. Velikost kroku tedy bude 0,1.



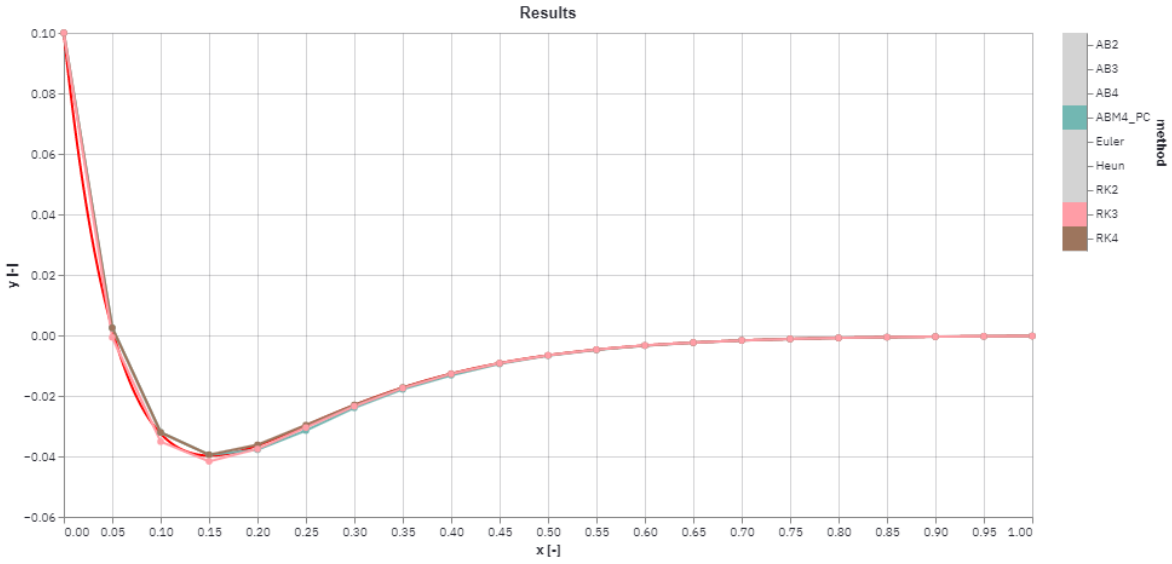
Obrázek 15 Průběh dosažených výsledků numerických metod, krok h = 0,1

S takto definovanými parametry odhady řešení u většiny metod nejsou příliš přesné, tři metody, které byly nejpřesnější jsou RK4, RK3 a ABM4. Výsledek je možno vidět na obr. 15. Ostatní průběhy jsou odfiltrovány pro přehlednost. Celkový výsledek je pak možné vidět na v tabulce 10, metoda AB2, AB3 a AB4 není uvedena, při zadaných parametrech byla nestabilní a nekonvergovala k řešení.

x [s]	Analytické	Euler	Heun	RK2	RK3	RK4	ABM4_PC
0	0,1000	0,1000	0,1000	0,1000	0,1000	0,1000	0,1000
0,1	-0,0325	-0,2000	0,0660	0,0660	-0,0712	-0,0209	-0,0209
0,2	-0,0364	0,0320	0,0426	0,0426	-0,0495	-0,0304	-0,0304
0,3	-0,0231	-0,0272	0,0271	0,0271	-0,0259	-0,0208	-0,0208
0,4	-0,0127	0,0066	0,0170	0,0170	-0,0129	-0,0119	-0,0356
0,5	-0,0066	-0,0039	0,0106	0,0106	-0,0063	-0,0064	-0,0382
0,6	-0,0034	0,0012	0,0065	0,0065	-0,0031	-0,0033	-0,0018
0,7	-0,0017	-0,0006	0,0040	0,0040	-0,0015	-0,0017	0,0226
0,8	-0,0008	0,0002	0,0024	0,0024	-0,0007	-0,0008	-0,0056
0,9	-0,0004	-0,0001	0,0015	0,0015	-0,0004	-0,0004	-0,0317
1	-0,0002	0,0000	0,0009	0,0009	-0,0002	-0,0002	-0,0001

Tabulka 10 Výsledky jednotlivých metod, krok h = 0,1

Nyní budeme pokus opakovat, tentokrát ale zvolíme velikost kroku h = 0,5. Průběhy tří nejlepších metod je pak možno vidět na obrázku 16 (opět RK4, RK3 a ABM4)



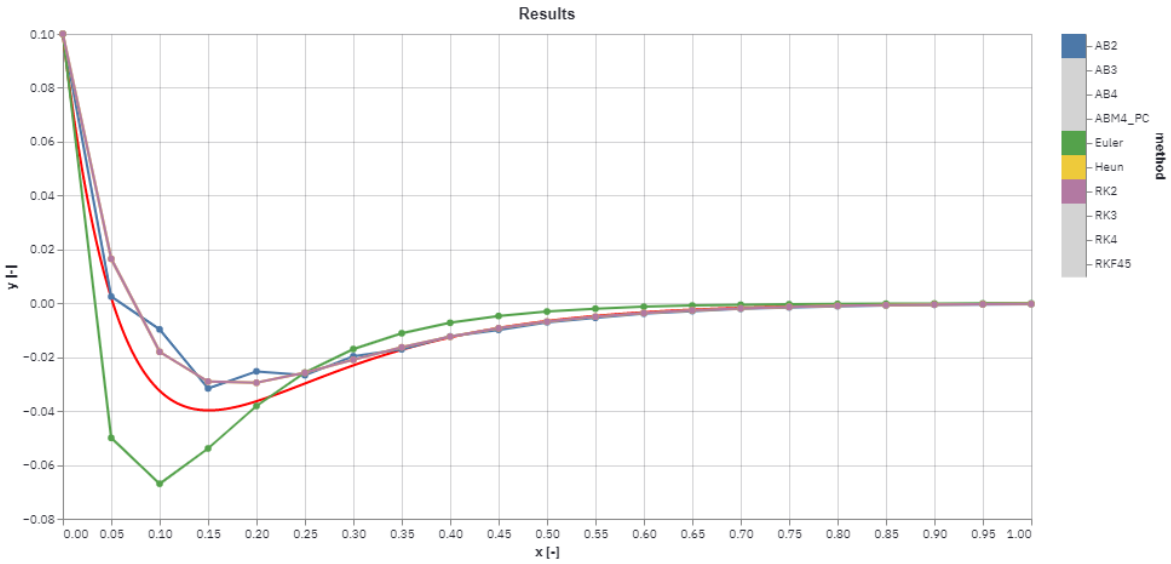
Obrázek 16 Průběh dosažených výsledků numerických metod, krok h = 0,1

V tomto případě již i metoda AB2 konvergovala k řešení, AB3 a AB4 však pořád ne a proto nejsou zahrnuty v tabulce 11 mezi výsledky jednotlivých metod.

x [s]	Analytické	Euler	Heun	RK2	RK3	RK4	AB2
0,00	0,1000	0,1000	0,1000	0,1000	0,1000	0,1000	0,1000
0,05	0,0021	-0,0500	0,0165	0,0165	-0,0006	0,0025	0,0025
0,10	-0,0325	-0,0670	-0,0180	-0,0180	-0,0351	-0,0321	-0,0097
0,15	-0,0398	-0,0539	-0,0291	-0,0291	-0,0416	-0,0395	-0,0316
0,20	-0,0364	-0,0381	-0,0295	-0,0295	-0,0376	-0,0362	-0,0253
0,25	-0,0298	-0,0257	-0,0258	-0,0258	-0,0305	-0,0297	-0,0267
0,30	-0,0231	-0,0170	-0,0210	-0,0210	-0,0234	-0,0230	-0,0197
0,35	-0,0173	-0,0111	-0,0164	-0,0164	-0,0174	-0,0172	-0,0171
0,40	-0,0127	-0,0073	-0,0124	-0,0124	-0,0128	-0,0127	-0,0123
0,45	-0,0092	-0,0047	-0,0093	-0,0093	-0,0092	-0,0092	-0,0099
0,50	-0,0066	-0,0031	-0,0068	-0,0068	-0,0066	-0,0066	-0,0070
0,55	-0,0047	-0,0020	-0,0050	-0,0050	-0,0047	-0,0047	-0,0054
0,60	-0,0034	-0,0013	-0,0036	-0,0036	-0,0033	-0,0034	-0,0039
0,65	-0,0024	-0,0008	-0,0026	-0,0026	-0,0024	-0,0024	-0,0029
0,70	-0,0017	-0,0005	-0,0019	-0,0019	-0,0017	-0,0017	-0,0021
0,75	-0,0012	-0,0004	-0,0013	-0,0013	-0,0012	-0,0012	-0,0015
0,80	-0,0008	-0,0002	-0,0010	-0,0010	-0,0008	-0,0008	-0,0011
0,85	-0,0006	-0,0002	-0,0007	-0,0007	-0,0006	-0,0006	-0,0008
0,90	-0,0004	-0,0001	-0,0005	-0,0005	-0,0004	-0,0004	-0,0006
0,95	-0,0003	-0,0001	-0,0003	-0,0003	-0,0003	-0,0003	-0,0004
1,00	-0,0002	0,0000	-0,0002	-0,0002	-0,0002	-0,0002	-0,0003

Tabulka 11 Výsledky jednotlivých metod, krok h = 0,05

V tomto případě si nevedly příliš špatně ani zbylé metody (Euler, Heun, RK2, AB2), viz obrázek 17 (Pozn.: RK2 a Heun se překrývají).



Obrázek 17 Průběh odhadů metod Euler, Heun, RK2, AB2, h = 0,05

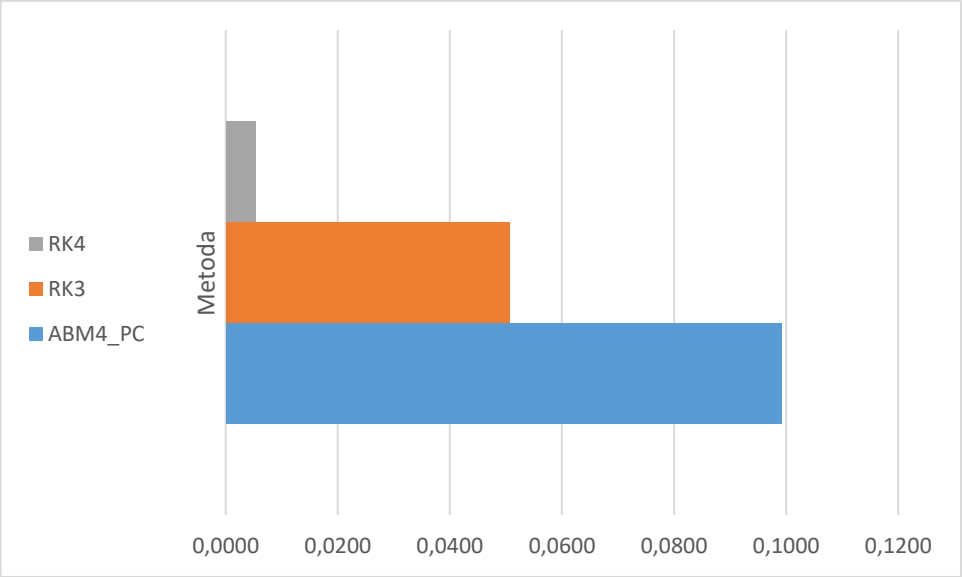
6.2.3 Srovnání výsledků

Pro porovnání metod použijeme dosažených výsledků. Jako ukazatel přesnosti počítejme sumu kvadratických odchylek od analytického řešení. Metody, které nekonvergovaly k řešení vynecháme. V tabulce 12 je pak vidět výsledek, hodnoty jsou normalizované (podělené největší odchylkou).

Euler	Heun	RK2	RK3	RK4	ABM4_PC
1,0000	0,5966	0,5966	0,0506	0,0053	0,0992

Tabulka 12 Srovnání metod, h = 0,1

Při srovnání 3 nejlepších dosažených odhadů (viz obr. 18) je zřetelně vidět, že RK4 si vedla výrazně lépe.



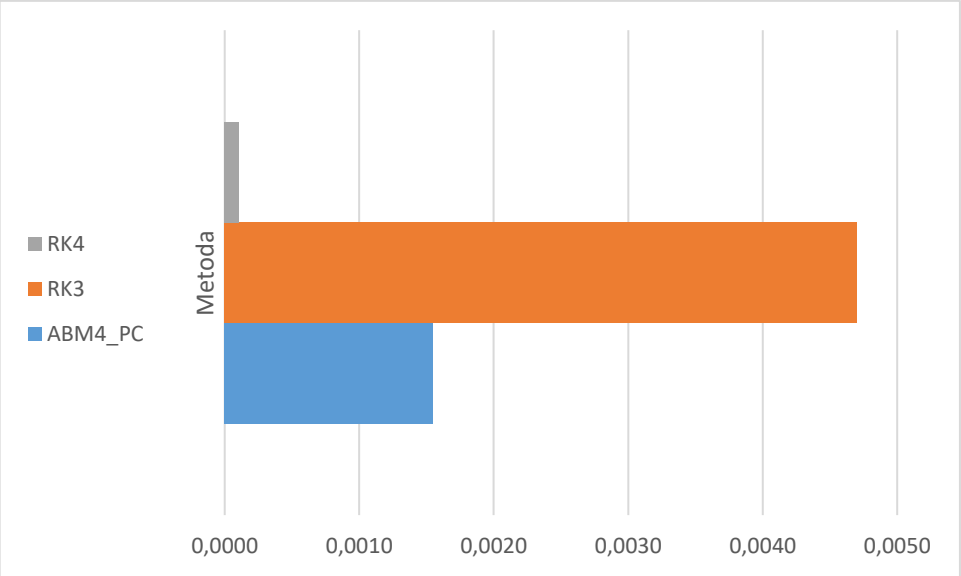
Obrázek 18 Srovnání metod, h = 0.1

V tabulce 13 můžeme vidět srovnání tentokrát při použité velikosti kroku h = 0,01.

Euler	Heun	RK2	RK3	RK4	ABM4_PC
1,0000	0,1401	0,1401	0,0047	0,0001	0,0015

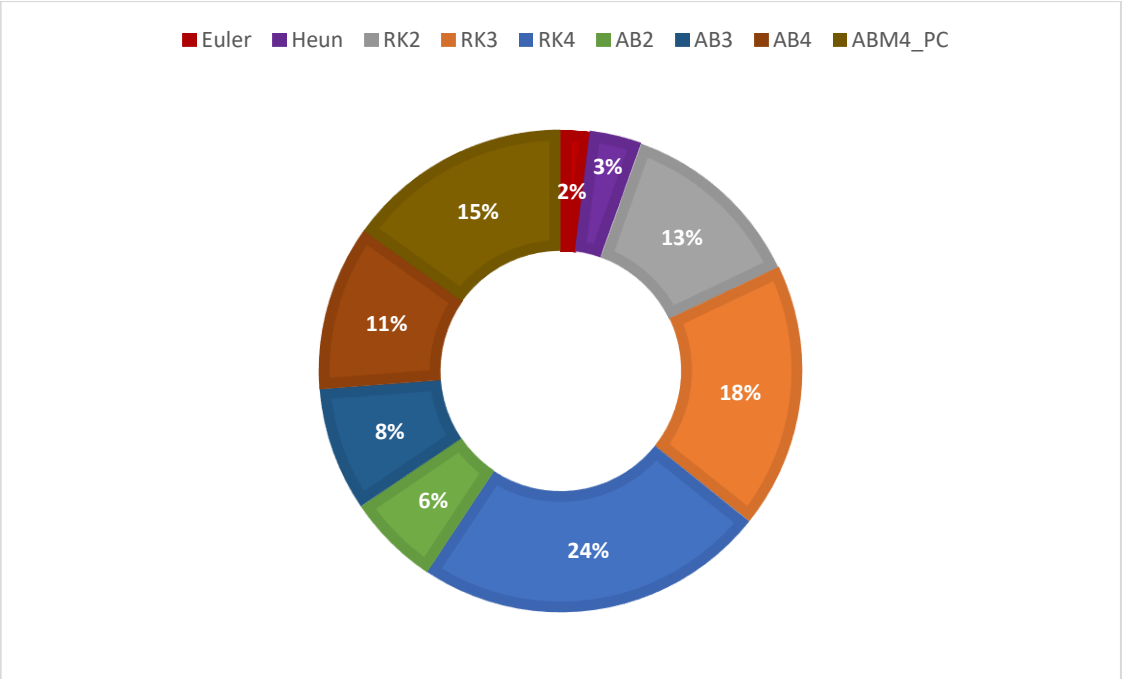
Tabulka 13 Srovnání metod, h = 0,05

Metoda RK4 je opět nejlepší, tentokrát však nutno podotknout, že ABM4 si vedla mnohem lépe než v předchozím případě a poskytla odhady přesnější než RK3, viz obrázek 19.



Obrázek 19 Porovnání metod, suma kvadratických odchylek, krok h = 0,05

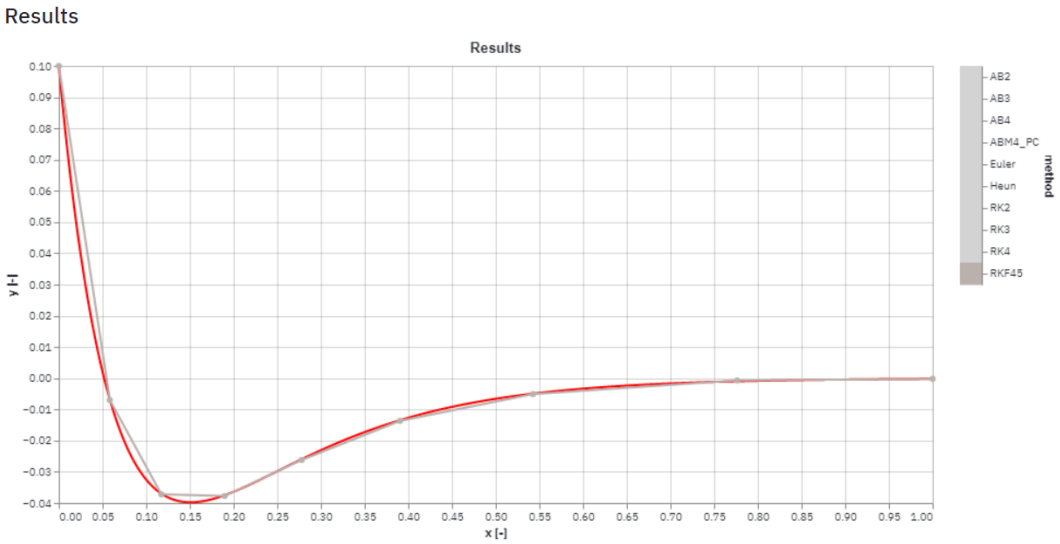
Doposud byla brána v potaz pouze přesnost jako taková, na obrázku 20 je vidět poměrné rozdělení celkového času, který daná metoda potřebovala pro poskytnutí odhadů (Pozn.: pro účel měření byla použita funkce měření systémového času během vykonávání jednotlivých funkcí, pro statistickou validnost bylo měření 30krát opakováno a poté zprůměrováno). Je zřejmé, že RK4 sice poskytuje nejlepší odhad, ale také stojí nejvíce času. Otázkou nyní je, zda bude RK4 pořád nejpřesnější i v případě, kdy např. snížíme velikost kroku pro metodu ABM4 tak, aby měla k dispozici stejné množství výpočetního času.



Obrázek 20 Průměrná proporcionální doba výpočtu jednotlivých metod

Experimentálně pak bylo zjištěno, že při nastavení kroku h = 0,0223 metoda ABM4 trvá přibližně stejné množství času jako RK4 s krokem h = 0,05 a zároveň dosahuje přesnějších výsledků, což z ní dělá efektivnější. Nutno ale podotknout, že tento výsledek platí pouze pro náš případ a nedá se generalizovat, protože jsou tady určité charakteristické vlastnosti ODR jako tuhost a stabilita, které pak mají vliv na výkonnost jednotlivých metod. Neexistuje tedy metoda, o které by se dalo říct, že je nejpřesnější. Za velmi účinné však lze označit v případě metody ABM4 použití kombinace explicitní a implicitní rovnice (popsáno v kapitole 3.3, neboť samotná metoda AB4 nevykazovala příliš přesných řešení.

Pokud do porovnání zapojíme metodu RKF45, pak adaptabilní velikostí kroků můžeme získat v porovnání přesnost/časová náročnost lepší výsledek. Na obrázku 21 je vidět průběh odhadů metodou RKF45.



Obrázek 21 Průběh odhadů metodou RKF45

V tabulce 14 je možné vidět, že metodou RKF45 bylo provedeno celkem 8 odhadů.

x [s]	Analytická	RKF45
0,0000	0,1000	0,1000
0,0568	-0,0052	-0,0054
0,1159	-0,0367	-0,0369
0,1879	-0,0377	-0,0379
0,2756	-0,0263	-0,0265
0,3874	-0,0137	-0,0139
0,5384	-0,0051	-0,0052
0,7689	-0,0010	-0,0008
1,0000	-0,0002	-0,0002

Tabulka 14 Výsledky metodou RKF45

Sumou kvadratických odchylek a podělením maximální získané hodnoty je pak vidět v tabulce 15. Je zřejmé, že s pomocí RKF45 bylo možné získat nejpřesnější odhad. Časová náročnost pak byla ve všech třech případech přibližně stejná.

RK4	ABM4_PC	RKF45
1,0000	0,0029	0,0001

Tabulka 15 Porovnání přenosti metod RK4, ABM4 A RKF45

ZÁVĚR

V rámci této diplomové práce byla vypracována literární rešerše na dané téma a byly představeny fundamentální metody, jakožto zástupci odlišných přístupů, v otázce řešení problémů s počáteční podmínkou. Ačkoliv existuje řada jiných metod, byly představeny právě tyto, protože splňují hned několik kritérií. Jsou dostatečně jednoduché na pochopení k získání vhledu do problematiky. Jsou schopné dosáhnout uspokojivých výsledků. A mají svůj praktický přinos. Zde je však nutné říct, že se v dnešní době prakticky sami o sobě nevyužívají, nicméně většina pokročilých a komplexních metod je založena na podobném postupu nebo kombinací několika postupů zmíněných metod.

Popsané metody byly implementovány a byla vytvořena knihovna v programovacím jazyce Python, která teoreticky umožňuje aplikovat jednotlivé metody pro libovolnou soustavu obyčejných diferenciálních rovnic prvního řádu. Knihovna tvoří základ interaktivní webové aplikace, která jednotlivé metody aplikuje a skrz ni uživatel může měnit definované vstupy. Uživatel této aplikace má tedy možnost se, pomocí demonstrativních příkladů, seznámit s elementární funkčností jednotlivých metod a má možnost srovnání přesnosti jednotlivých metod na základě přehledných grafických výstupů aplikace. Tato aplikace tak má sloužit jako podpora výuky předmětu Simulace systémů na Fakultě Aplikované Informatiky UTB.

SEZNAM POUŽITÉ LITERATURY

[1] YA YAN LU. *Numerical Methods for Differential Equations*. Kowloon, Hong Kong. City University of Hong Kong.

[2] BUTCHER, J. C. *Numerical methods for ordinary differential equations*. 2nd ed. Hoboken, NJ: Wiley, c2008. ISBN 978-0-470-72335-7.

[3] RALSTON, Anthony. *Základy numerické matematiky: příručka pro univerzity ČSR*. 2. čes. vyd. Praha: Academia, 1978.

[4] BURDEN, Richard L. a J. Douglas FAIRES. *Numerical analysis*. 8th ed. Belmont, CA: Thomson Brooks/Cole, c2005. ISBN 05-343-9200-8.

[5] *ODE Laboratories: A Sabbatical Project by Christopher A. Barker* [online]. San Joaquin Delta College, 5151 Pacific Ave., Stockton, CA 95207, USA: San Joaquin Delta College, 2017 [cit. 2019-11-24]. Dostupné z: http://calculuslab.deltacollege.edu/

[6] EPPERSON, James F. *An Introduction to Numerical Methods and Analysis*. 2nd Edition. Hoboken, New Jersey: Wiley, 2013. ISBN 978-1-118-36759-9.

[7] AYYUB, Bilal a Richard H. MCCUEN. *Numerical Analysis for Engineers: Methods and Applications*. Second Edition. 6000 Broken Sound Parkway NW, Suite 300Boca Raton, FL 33487-2742: Chapman and Hall/CRC, 2015. ISBN 978-1482250350.

[8] NASSIF, Nabil a Dolly Khuwayri FAYYAD. *Introduction to Numerical Analysis and Scientific Computing*. 2013: Chapman and Hall/CRC, 2013. ISBN 9781466589483.

[9] ATKINSON, Kendall E., Weimin HAN a David STEWART. *Numerical Solution of Ordinary Differential Equations*. 1th. Iowa City, Iowa: Wiley-Interscience, 2009. ISBN 9781118164495.

[10] HOFFMAN, Joe D. *Numerical Methods for Engineers and Scientists: Second Edition Revised and Expanded*. 2th. NEW YORK, USA: CRC Press, 2001. ISBN 0-8247-0443-6.

[11] SCHÄFER, Michael. *Computational Engineering: Introduction to Numerical Methods*. 1. Technische Universität Darmstadt, Germany: Springer, 2006. ISBN 978-3-540-30686-3.

[12] GILAT, Amos Gilat a Vish SUBRAMANIAM. *Numerical Methods: for Engineers and Scientists*. 3rd Edition. The Ohio State University: Department of Mechanical Engineering, 2013. ISBN 978-1-118-55493-7.

[13] DAHLQUIST, Germund a Åke BJÖRCK. *Numerical Methods in Scientific Computing: Volume I*. 3600 Market Street, 6th Floor, Philadelphia, PA 19104-2688 USA: Society for Industrial and Applied Mathematics, 2008. ISBN 978-0-89871-644-3. Dostupné z: doi:https://doi.org/10.1137/1.9780898717785

[14] Python Software Foundation [online]. Wilmington, Delaware, USA: Python Software Foundation, 2001 [cit. 2021-5-10]. Dostupné z: https://www.python.org/

[15] NumPy [online]. USA: NumPy, 2019 [cit. 2021-5-10]. Dostupné z: https://numpy.org/

[16] Pandas [online]. USA: Pandas, 2008 [cit. 2021-5-10]. Dostupné z: https://pandas.pydata.org/

[17] Altair: Declarative Visualization in Python [online]. 2016 [cit. 2021-5-10]. Dostupné z: https://altair-viz.github.io/

[18] Streamlit [online]. USA: Streamlit, 2019 [cit. 2021-5-10]. Dostupné z: https://streamlit.io/

[19] WolframAlpha [online]. United Kingdom: Wolfram [cit. 2021-5-11]. Dostupné z: https://www.wolframalpha.com/examples/mathematics/applied-mathematics/numerical-analysis/numerical-differential-equation-solving/

[20] CodeSansar [online]. [cit. 2021-5-11]. Dostupné z: https://www.codesansar.com/numerical-methods/runge-kutta-rk-method-online-calculator.htm

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ODR	**O**byčejná **D**iferenciální **R**ovnice
RK	**R**unge-**K**utta
RK2	**R**unge-**K**utta druhého řádu
RK3	**R**unge-**K**utta třetí řádu
RK4	**R**unge-**K**utta čtvrtého řádu
RKF	**R**unge-**K**utta-**F**ehlberg
RKF45	**R**unge-**K**utta-**F**ehlberg s přesností čtvrtého a pátého řádu
AB	**A**dams-**B**ashforth
AB2	**A**dams-**B**ashforth druhého řádu
AB3	**A**dams-**B**ashforth třetího řádu
AB4	**A**dams-**B**ashforth čtvrtého řádu
AM	**A**dams-**M**oulton
AM2	**A**dams-**M**oulton druhého řádu
AM3	**A**dams-**M**oulton třetího řádu
AM4	**A**dams-**M**oulton čtvrtého řádu
ABM4	**A**dams-**B**ashforth-**M**oulton čtvrtého řádu
PC	**P**redikto-Korektor
CSV	**C**omma-**S**eparated **V**alues
PDF	**P**ortable **D**ocument **F**ormat
JSON	**J**ava**S**cript **O**bject **N**otation
PNG	**P**ortable **N**etwork **G**raphics

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: ZDROJOVÝ KÓD KNIHOVNY FUNKCÍ VYBRANÝCH NUMERICKÝCH METOD, PRO ŘEŠENÍ SOUSTAVY DIFERENCIÁLNÍCH ROVNIC

```
import numpy as np
#   TABLEAU FOR RK FAMILY METHODS    #
def get_tableau_rkx(x):
    if 2 == x:
        A = (0, 1 / 2)
        B = [(0, 0), (1 / 2, 0)]
        C = (0, 1)
    elif 3 == x:
        A = (0, 1 / 2, 1)
        B = [(0, 0, 0), (1 / 2, 0, 0), (-1, 2, 0)]
        C = (1 / 6, 2 / 3, 1 / 6)
    elif 4 == x:
        A = (0, 1 / 2, 1 / 2, 1)
        B = [(0, 0, 0, 0), (1 / 2, 0, 0, 0), (0, 1 / 2, 0, 0), (0, 0, 1, 0)]
        C = (1 / 6, 1 / 3, 1 / 3, 1 / 6)
    return A, B, C


#   TABLEAU FOR RKF45 METHODS    #
def get_tableau_rkf45():
    # Coefficients used to compute the independent variable argument of f
    A = (0, 1 / 4, 3 / 8, 12 / 13, 1, 1 / 2)
    # Coefficients used to compute the dependent variable argument of f
    B = [(0, 0, 0, 0, 0, 0),
         (1 / 4, 0, 0, 0, 0, 0),
         (3 / 32, 9 / 32, 0, 0, 0, 0),
         (1932 / 2197, -7200 / 2197, 7296 / 2197, 0, 0, 0),
         (439 / 216, -8, 3680 / 513, -845 / 4104, 0, 0),
         (-8 / 27, 2, -3544 / 2565, 1859 / 4104, -11 / 40, 0)]
    # Coefficients used to compute 4th order RK estimate
    C = (25 / 216, 0, 1408 / 2565, 2197 / 4104, -1 / 5)
    # Coefficients used to compute local truncation error estimate.  These
    # come from subtracting a 4th order RK estimate from a 5th order RK
    # estimate.
    CT = (1 / 360, 0, -128 / 4275, -2197 / 75240, 1 / 50, 2 / 55)
    return A, B, C, CT


#   TABLEAU FOR ADAM'S FAMILY METHODS    #
def get_tableau_abx(x):
    if 2 == x:
        A = 1 / 2
        B = [3, -1]
    elif 3 == x:
        A = 1 / 12
        B = [23, -16, 5]
    elif 4 == x:
        A = 1 / 24
        B = [55, -59, 37, -9]
    return A, B

def get_tableau_abc4():
```

```
    A = 1 / 24
    B_PRED = [55, -59, 37, -9]
    B_COR = [9, 19, -5, 1]
    return A, B_PRED, B_COR

#   SINGLE-STEP METHODS     #
def euler(f, y0, x, h):
    #     """Euler's method to solve system of ODE
    #     USAGE:
    #         y = euler(f, y0, x, h)
    #     INPUT:
    #         f     - function equal to dy/dx = f(y,x)
    #         y0    - Array of initial conditions
    #         x     - Array of independent variable values
    #         h     - Step size
    #     OUTPUT:
    #         y     - NumPy array of corresponding solution function values
    #     NOTES:
    #         This function implements singlestep Euler's method
    #         to solve the initial value problem

    ode_system = len(y0)  # if more then one init condition -> system of ODE's
    n = len(x)  # number of required steps

    # init variables for calc
    y = [[0] * ode_system for i in range(n)]
    y[0] = y0

    for i in range(0, n - 1):
        y[i + 1][:] = np.add(y[i], np.multiply(f(x[i], y[i]), h))
    return y


def heun(f, y0, x, h):
    #     """Heun's method to solve system of ODE
    #     USAGE:
    #         y = heun(f, y0, x, h)
    #     INPUT:
    #         f     - function equal to dy/dx = f(y,x)
    #         y0    - Array of initial conditions
    #         x     - Array of independent variable values
    #         h     - Step size
    #     OUTPUT:
    #         y     - NumPy array of corresponding solution function values
    #     NOTES:
    #         This function implements singlestep Heun's method
    #         to solve the initial value problem

    # initialization of variables
    ode_system = len(y0)  # if more then one init condition -> system of ODE's
    n = len(x)  # number of required steps
    y = [[0] * ode_system for i in range(n)]
    y[0] = y0

    for i in range(0, n - 1):
        # Explicit Euler's formula - PREDICTOR
        y_pred = np.add(y[i], np.multiply(f(x[i], y[i]), h))
        # Implicit Euler's formula and usage of the trapezoidal rule - CORRECTOR
        # Correction of the approximation from explicit formula
```

```
        y[i + 1][:] = np.add(y[i], np.multiply(np.add(f(x[i], y[i]), f(x[i+1],
y_pred)), h/2))
    return y


def rkx(f, y0, x, h, order):
    #     """RUNGE-KUTTA method to solve system of ODE
    #     USAGE:
    #         y = rkx(f, y0, x, h, order)
    #     INPUT:
    #         f     - function equal to dy/dx = f(y,x)
    #         y0    - Array of initial conditions
    #         x     - Array of independent variable values
    #         h     - Step size
    #       order - Order of the method - states the accuracy of the approximation
& how many coefs will be needed
    #     OUTPUT:
    #         y     - NumPy array of corresponding solution function values
    #     NOTES:
    #         This function implements singlestep RK method of nth order
    #         to solve the initial value problem

    # initialization of variables
    ode_system = len(y0)  # if more then one init condition -> system of ODE's
    n = len(x)  # number of required steps
    y = [[0] * ode_system for i in range(n)]
    k = [[0] * ode_system for i in range(order)]
    kB = np.empty(ode_system, float)
    y[0] = y0

    # get coefficients for calculation
    A, B, C = get_tableau_rkx(order)

    # calculation, until the end point b is reached
    for i in range(0, n - 1):
        for o in range(0, order):
            # calculate of coefficients k
            kB = np.multiply([*zip(*k)], B[o])
            sumkB = sum(np.array([*zip(*kB)]))
            y_curr = np.add(sumkB, y[i])
            x_curr = x[i] + h * A[o]
            if 1 == ode_system:
                y_curr = [y_curr]
            k[o][:] = np.multiply(f(x_curr, y_curr), h)

        y[i + 1][:] = np.add(y[i], sum(np.multiply([*zip(*k)], C).T))
    return y


#   MULTI-STEP METHODS     #

def abx(f, y0, x, h, order):
    #     """ADAMS-BASFORTH method to solve system of ODE
    #     USAGE:
    #         y = abmx(f, y0, x, h, order)
    #     INPUT:
    #         f     - function equal to dy/dx = f(y,x)
    #         y0    - Array of initial conditions
    #         x     - Array of independent variable values
```

```
    #         h     - Step size
    #          order - Order of the method - states how many previous solutions
y(i) are needed for approximation
    #     OUTPUT:
    #         y     - NumPy array of corresponding solution function values
    #     NOTES:
    #         This function implements multistep AB method of nth order
    #         to solve the initial value problem

    # initialization of variables
    ode_system = len(y0)  # if more then one init condition -> system of ODE's
    n = len(x)  # number of required steps
    tmp = np.empty(ode_system, float)
    y_m = [[0] * ode_system for i in range(n - order)]
    # Initialization of calculation by RK4 method
    y_s = rkx(f, y0, x[:order], h, 4)
    y = np.vstack((y_s, y_m))
    # get coefficients for calculation
    A, B = get_tableau_abx(order)

    # calculation, until the end point b is reached
    for i in range(order - 1, n - 1):
        for j in range(0, order):
            tmp = np.add(tmp, np.multiply(f(x[i - j], y[i - j]), B[j]))
        y[i + 1][:] = np.add(y[i], np.multiply(np.multiply(A, h), tmp))
        # Clear support variable for next iteration
        tmp = 0
    return y


#   PREDICTOR-CORRECTOR METHODS  #

def abm4_pc(f, y0, x, h):
    #     """ADAMS-BASFORTH-MOULTON method to solve system of ODE
    #     USAGE:
    #         y = abm4_pc(f, y0, x, h)
    #     INPUT:
    #         f     - function equal to dy/dx = f(y,x)
    #         y0    - Array of initial conditions
    #         x     - Array of independent variable values
    #         h     - Step size
    #     OUTPUT:
    #         y     - NumPy array of corresponding solution function values
    #     NOTES:
    #          This function implements multistep Predictor-Corector method ABM
of 4th order
    #         to solve the initial value problem

    # initialization of variables
    ode_system = len(y0)  # if more then one init condition -> system of ODE's
is expected
    n = len(x)            # number of required steps
    order = 4
    tmp = np.empty(ode_system, float)
    y_m = [[0] * ode_system for i in range(n - order)]
    tmp_pred = 0
    tmp_kor = 0
    # Initialization of calculation by RK4 method
    y_s = rkx(f, y0, x[:order], h, 4)
```

```
    y = np.vstack((y_s, y_m))
    # get coefficients for calculation
    A, B_PRED, B_COR = get_tableau_abc4()

    # calculation, until end the point b is reached
    for i in range(order - 1, n - 1):
        # Calculation of explicit AB formula of 4th order - PREDICTOR
        for j in range(0, order):
            tmp_pred = np.add(tmp_pred, np.multiply(f(x[i - j], y[i - j]),
B_PRED[j]))
        y[i + 1][:] = np.add(y[i], np.multiply(np.multiply(A, h), tmp_pred))
        # Calculation of implicit AM formula of 4th order - CORRECTOR
        # Correction of the approximation from explicit formula
        for k in range(0, order):
            tmp_kor = np.add(tmp_kor, np.multiply(f(x[i - k + 1], y[i - k + 1]),
B_COR[k]))
        y[i + 1] = np.add(y[i], np.multiply(np.multiply(A, h), tmp_kor))
        # Clear support variables for next iteration
        tmp_pred = 0
        tmp_kor = 0
    return y


#   ADAPTIVE-STEP METHODS     #

def rkf45(f, y0, a, b, tol, h_max, h_min):
    #     """Runge-Kutta-Fehlberg 45 method to solve system of ODE
    #     USAGE:
    #         t, x, e = rkf45(f, y0, a, b, tol, h_max, h_min)
    #     INPUT:
    #         f     - function equal to dy/dx = f(x,t)
    #         y0    - Array of  of initial conditions
    #         a     - left-hand endpoint of interval (initial condition is here)
    #         b     - right-hand endpoint of interval
    #         tol   - maximum value of local truncation error estimate
    #         h_max  - maximum step size
    #         h_min  - minimum step size
    #     OUTPUT:
    #         x     - NumPy array of independent variable values
    #         y     - NumPy array of corresponding solution function values
    #         e     - Return status ( 0 == successful, -1 == failure => tolerance
not possible to reach)
    #     NOTES:
    #         This function implements 4th-5th order Runge-Kutta-Fehlberg method
    #         to solve the initial value problem

    # initialization of variables
    order = 6
    e = 0
    ode_system = len(y0)  # if more then one init condition -> system of ODE's
is expected
    kB = np.empty(ode_system, float)
    # Set x and y according to initial condition and assume that h starts with
h_max
    x_cur = a
    y_cur = np.array(y0)
    h = h_max
    # Initialize arrays that will be returned
    x = np.array([x_cur])
```

```
    y = np.array([y_cur])
    k = [[0] * ode_system for i in range(order)]
    # get coefficients for calculation
    A, B, C, CT = get_tableau_rkf45()

    # calculation, until the end point b is reached
    while x_cur < b:
        # Adjust step size when we get to last interval
        if x_cur + h > b:
            h = b - x_cur
        # calculation of coefficients k and their sum with B
        for o in range(0, order):
            kB = np.multiply([*zip(*k)], B[o])
            sumkB = sum(np.array([*zip(*kB)]))
            if 1 == ode_system:
                k[o][:] = np.multiply(f(x_cur + h * A[o], np.add(sumkB,
[y_cur])), h)
            else:
                k[o][:] = np.multiply(f(x_cur + h * A[o], np.add(sumkB, y_cur)),
h)
        # Calculate the estimate of the local truncation error
        r = max(abs(sum(np.multiply([*zip(*k)], CT).T)) / h)
        # Null check to avoid division by zero
        if 0 == r:
            e = -1
            break
        # If local truncation error fulfil expected tolerance, 4th estimate of
solution and step h are accepted
        if r <= tol:
            x_cur = x_cur + h
            y_cur = np.add(y_cur, sum(np.multiply([*zip(*k[:5])], C).T))
            x = np.append(x, x_cur)
            y = np.vstack((y, y_cur))
        # adjust size of the step
        h = h * min(max(0.84 * (tol / r) ** (1 / 4), 0.1), 4.0)
        # check if required constraints are not exceeded
        if h > h_max:
            h = h_max
        elif h < h_min:
            # Stop algorithm with failure
            # Could not converge to the required tolerance with chose minimum
step size
            e = -1
            break
    return x, y, e
```

PŘÍLOHA P II: DOPROVODNÝ MATERIÁL OBSAHUJÍCÍ ÚLOHY ŘEŠENÉ V APLIKACI

Populační model

Růst populace je dynamický proces, který lze jednoduše popsat diferenciální rovnicí. Jedním z modelů, které existují pro tento jev je popsán rovnicí 1. Tento model je někdy též nazýván jako „logisticky model“. Při růstu populace dochází v určitém bodě ke zpomalení, resp. k saturaci, např. z důvodu nedostatků zdrojů, a tak se v určité fázi růst zpomaluje.

$$y'' = g - \frac{\mathrm{c}}{\mathrm{m}} y'^2 \tag{1}$$

Obecné řešení pak je pak dáno rovnici 2.

$$N = \frac{CMe^{at}}{M + Ce^{at}} \tag{2}$$

Matematické kyvadlo

Diferenciální rovnice jednoduchého matematického kyvadla (viz obr. 1) je popsána rovnicí 3. Jedná se pouze o teoretického kyvadlo, kde vliv tření není brán v potaz:

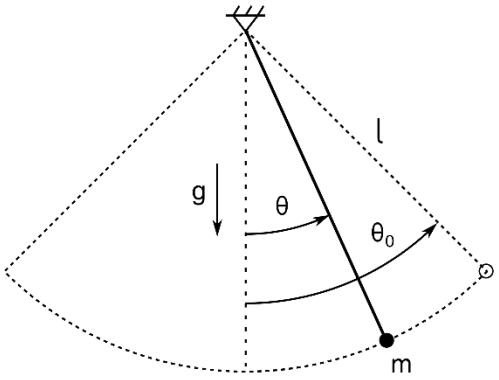
$$y'' = -\frac{g}{L} \sin(y) \tag{3}$$

Obecné řešení je pak dáno rovnici 4:

$$y = C_1 sin\sqrt{\frac{g}{l}}t + C_2 \cos\sqrt{\frac{g}{l}}t \tag{4}$$

Kde:

θ … *úhlové posuní z klidového stavu*
g … *gravitační zrychlení*
l … *délka kyvadla*



Obrázek 22 Matematické kyvadlo

Parašutista

Parašutista o hmotnosti m padá volným vertikálním pádem a pociťuje aerodynamický odpor, y udává vzdálenost uraženou ve směru nahoru. Diferenciální rovnice popisující pád je dána rovnici 5.

$$y'' = \frac{\mathrm{c}}{\mathrm{m}} y'^2 - g \tag{5}$$

Kde:

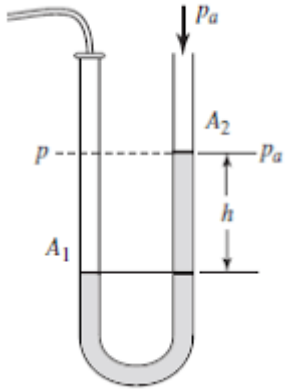
$c \ldots součinitel\ odporu$

$g \ldots gravitační\ zrychlení$

$m \ldots hmotnost$

Kapalinového manometr

Mějme kapalinový manometr, který pracuje pod tlakem p. Sloupec kapaliny manometru získává vlivem působení tlaku polohu zobrazenou na obr 2. Celková délka sloupce kapaliny v manometru je l.



Obrázek 23 Model manometru

Pohybová rovnice je dána rovnicí 6.

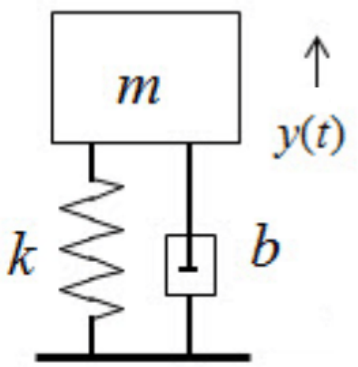
$$y'' + \frac{2g}{l} y = 0 \tag{6}$$

Obecné řešení pak je dáno rovnicí 7:

$$y = C_1 cos \sqrt{\frac{2g}{l}} t + C_2 cos \sqrt{\frac{2g}{l}} t \tag{7}$$

Soustava pružina-hmota-tlumič

Mějme soustavu pružina-hmota-tlumič, kde je pohyb hmotného bodu brzděn další silou. Protože toto tlumení je primárně třecí silou, předpokládejme, že je úměrná rychlosti hmotného bodu.



Obrázek 24 Model manometru

Tento systém lze pak popsat diferenciální rovnicí 8.

$$my'' + bx' + kx = 0 \tag{5}$$

Charakteristická rovnice pak je:

$$m\lambda^2 + b\lambda + k = 0 \tag{5}$$

$$\lambda = -b \pm \frac{\sqrt{b^2 - 4mk}}{2m}$$

Zde rozlišujeme tři případy (viz obr4):

1. Nadkritický útlum – je-li $b^2 > 4mk$ a rovnice tedy má 2 reálné kořeny. Obecné řešení pak je:

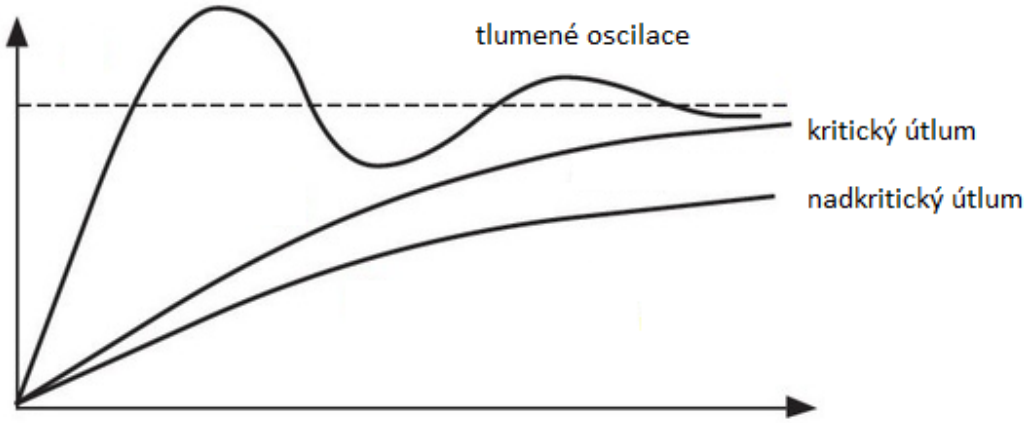
$$y = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t} \tag{5}$$

2. Kritický útlum – je-li $b^2 = 4mk$ a rovnice má tedy jeden dvojnásobný kořen. Obecné řešení pak je:

$$y = e^{\lambda t}(C_1 + C_2 t) \tag{5}$$

3. Tlumený oscilátor je-li $b^2 < 4mk$ a rovnice pak má tedy komplexně sdružený kořen. Obecné řešení pak je:

$$y = e^{-t}(C_1 \cos(\beta t) + C_2 \sin(\beta t)) \quad (5)$$



Obrázek 25 Model manometru