VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NUMERICKÉ VÝPOČTY URČITÝCH INTEGRÁLŮ

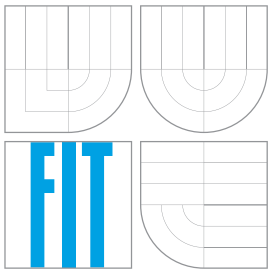
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ MIKULKA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NUMERICKÉ VÝPOČTY URČITÝCH INTEGRÁLŮ

FINITE INTEGRALS NUMERICAL COMPUTATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ MIKULKA

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. JIŘÍ KUNOVSKÝ, CSc.

BRNO 2014

Abstrakt

Aplikace určitého integrálu funkcí více proměnných proniká stále do více průmyslových odvětví a vědeckých disciplín. Požadavky kladené na řešení těchto problémů (např. vysoká přesnost, vysoká rychlost výpočtu, aj.) jsou však často velmi protichůdné. Není tak vždy možné aplikovat analytické postupy řešení, a tak se nabízejí různé numerické metody. Neustále rostoucí komplexita řešených problémů však klade příliš vysoké nároky na mnohé numerické metody, a proto ani mnohé z těchto metod nejsou vhodné pro řešení podobných problémů. Cílem této diplomové práce je návrh a implementace nové numerické metody pro přesný a rychlý výpočet určitých integrálů funkcí více proměnných. Tato nová metoda vhodně kombinuje již existující přístupy v oblasti numerické matematiky.

Abstract

The application of the finite integral of multiple variable functions is penetrating into more and more industries and science disciplines. The demands placed on solutions to these problems (such as high accuracy or high speed) are often quite contradictory. Therefore, it is not always possible to apply analytical approaches to these problems; numerical methods provide a suitable alternative. However, the ever-growing complexity of these problems places too high a demand on many of these numerical methods, and so neither of these methods are useful for solving such problems. The goal of this thesis is to design and implement a new numerical method that provides highly accurate and very fast computation of finite integrals of multiple variable functions. This new method combines pre-existing approaches in the field of numerical mathematics.

Klíčová slova

určitý integrál, integrál jedné a více proměnných, numerické metody, numerická integrace, numerická derivace, Taylorův polynom, vysoce přesné výpočty

Keywords

finite integral, single and multiple variable integral, numerical methods, numerical integration, numerical derivation, Taylor polynomial, high-precision computation

Citace

Jiří Mikulka: Numerické výpočty určitých integrálů, diplomová práce, Brno, FIT VUT v Brně, 2014

Numerické výpočty určitých integrálů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Jiřího Kunovského, CSc. Další konzultace mi poskytli pánové Ing. Ondřej Holub a Ing. Jan Chaloupka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Mikulka
25. května 2014

Poděkování

Panu doc. Ing. Jiřímu Kunovskému, CSc. děkuji za vedení, cenné rady a zkušenosti při zpracování této diplomové práce. Mé díky a uznání patří také mým nejbližším za upřímnou podporu a ustavičné povzbuzování.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Integrální a diferenciální počet se během několika posledních staletí dostával stále více do středu zájmu nejen věd matematických, nýbrž všech, které matematiku jakýmkoli způsobem využívají. Ať už se tedy jedná o samotnou matematiku, fyziku, chemii, statiku, meteorologii či návrh nových materiálů, je pro získání přesných výsledků velmi komplexních výpočtů třeba využít popis diferenciálními či integrálními rovnicemi. Analytické řešení těchto rovnic však zpravidla bývá velmi náročné, ne-li téměř nemožné. Proto vyvstala potřeba přijatelně přesných, ale relativně jednoduše použitelných metod. Těmito metodami jsou *metody numerické*.

Velmi významné postavení má numerické řešení diferenciálních a integrálních rovnic a systémů také v oblasti modelování a simulace, kdy předmětem experimentu je zjednodušený popis reálného světa či systému. Tyto problémy lze řešit analyticky, ale takové řešení existuje pouze pro jednoduché, anebo velmi zjednodušené modely. Ať už se jedná o modelování náhodných procesů, spojitou simulaci elektrických a elektronických obvodů nebo o simulaci fyzikálních, chemických, biologických či astronomických jevů, numerické řešení může poskytnout relativně přesné řešení a tedy přijatelnou představu o skutečném (analytickém) řešení. Přesto však tyto metody mají svá omezení vyplývající z jejich oblasti stability, nepřesnosti, kumulativním chybám výpočtu a zejména diskretizaci řešeného problému. [21, 22]

První zmínky o metodách připomínající numerickou integraci pocházejí z antického Řecka (přibližně 370 př. n. l.). Další významnější zmínky se po dlouhé odmlce objevují až v 16. století, kdy Cavalieri navrhl metodu, s jejíž pomocí byl schopen vypočítat určitý integrál funkce $f(x) = x^n$ až do řádu $n = 9$. Rozmach problematiky integrálního počtu nastal v 17. století, kdy Newton a Leibnitz výrazně rozšířili poznatky v této oblasti. Všechny metody však doposud byly numerické, protože jejich odvození bylo přímočařejší. Formalizace a analytického řešení se integrální počet dočkal až v 19. století, kdy již existoval dostatečně silný matematický aparát, který umožnil formalizaci problematiky integrálního počtu a návrh stabilního analytického řešení. Z historického hlediska je zřejmé, že návrh numerického řešení předchází návrh analytického, neboť jeho odvození je snazší a přímočařejší než formalizace analytického řešení, které vyžaduje poměrně pokročilý a robustní matematický aparát.[1]

V současnosti existuje velmi rozsáhlá znalost analytického řešení určitých integrálů funkcí jedné a více proměnných. Avšak s velmi rychle rostoucí složitostí a komplexitou problémů přestává být analytické řešení použitelné (z hlediska výpočetního času a prostoru).

[1] Volná parafráze `<http://en.wikipedia.org/wiki/Integral#History>`.

Proto se využívají různé numerické metody, které však s rostoucím počtem proměnných produkují výsledek s poměrně velkou chybou. Při velmi komplexních problémech tak tato chyba může být natolik významná, že výsledek je prakticky nepoužitelný. Snahou této diplomové práce je navrhnout novou numerickou metodu, která by tento problém co nejvíce eliminovala při zachování žádoucích vlastností (rychlost a vysoká přesnost výpočtu). Detailní popis této metody a její implementace je obsahem této textové zprávy.

Zpráva je rozdělena na dvě části – teoretickou a praktickou. V teoretické části I jsou zavedeny a připomenuty fundamentální pojmy a znalosti z oblasti integrálního počtu funkcí jedné a více proměnných (kapitoly 2.1 a 2.2). Pro tyto integrály jsou uvedeny základní pravidla analytického řešení (kapitola 2.1.2) a také vysvětleny základní numerické metody (kapitola 2.1.3, 2.1.4 a 2.2.2). Významnou součástí teoretického rozboru problematiky je detailní popis a vysvětlení nové numerické metody v kapitole 3, nejprve pro funkce jedné proměnné, což je později rozšířeno do prostoru obecné dimenze. V praktické části II je podrobně popsán návrh (kapitola 4.1) a implementace (kapitola 4.2) aplikace provádějící numerickou integraci metodou zavedenou v teoretické části. Tato aplikace byla podrobena testování a následnému srovnání s již existujícími implementacemi (viz kapitola 5). Výsledky byly vyhodnoceny a byla provedena diskuze nad výsledky tohoto srovnání.

Část I

Integrální počet

Kapitola 2

Od integrálu funkce jedné proměnné k integrálu v prostoru obecné dimenze

Pro zavedení nové numerické metody pro řešení určitých integrálů funkcí více proměnných je nutné připomenout a objasnit běžně používané pojmy a definice z oblasti integrálního počtu. Nejprve budou uvedeny definice a pojmy z oblasti integrálního počtu funkcí jedné proměnné, včetně jejich analytického řešení a některých často používaných numerických metod, a pak budou tyto definice rozšířeny na funkce více proměnných.

2.1 Integrál funkce jedné proměnné

Integrál funkce jedné proměnné lze považovat za základní pojem celé teorie integrálního počtu, neboť vícenásobné integrály (resp. integrály funkcí více proměnných) jsou rozšířením této teorie z $2-$dimenzionálního do obecného $n-$dimenzionálního prostoru.

Definujme proto základní pojmy integrálního počtu funkcí jedné proměnné, jež budou později zobecněny na funkce více proměnných. Tyto pojmy a definice poslouží k odvození nové numerické metody. Definice zde uvedené jsou převzaty z [11–13, 15, 16, 26, 27] a vhodně upraveny tak, aby všechny používaly jednotnou terminologii.

2.1.1 Definice pojmů

Neurčitý integrál

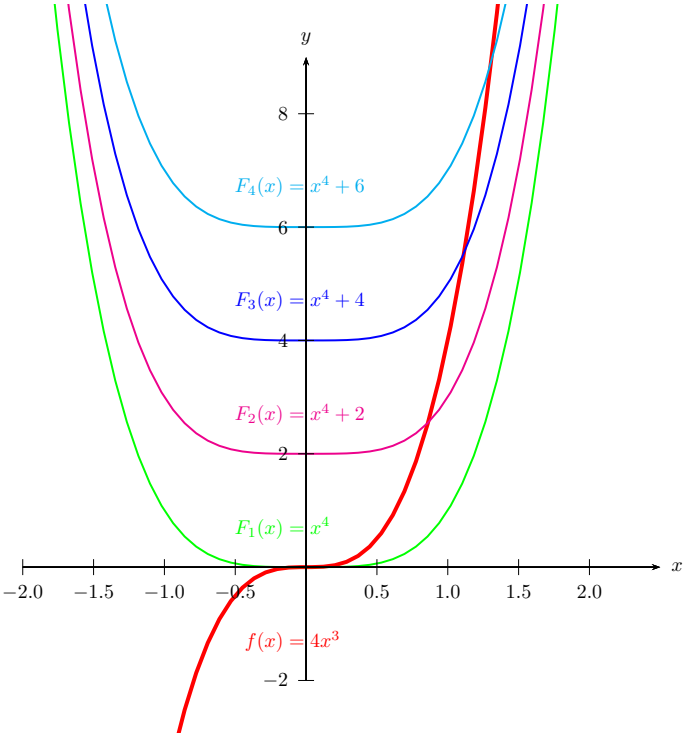
Definice 2.1 (Primitivní funkce)**.** Nechť $\mathcal{I}$ je interval v $\mathbb{R}$ a $f : \mathcal{I} \to \mathbb{R}$ funkce. Funkci F nazveme **primitivní** k funkci f v intervalu $\mathcal{I}$, platí-li pro každé $x \in \mathcal{I}$ vztah

$$F'(x) = f(x). \tag{2.1}$$

(V případě uzavřeného intervalu rozumíme derivací v krajních bodech jednostranné derivace.)

Věta 2.1. *Je-li funkce F primitivní funkcí k nějaké funkci f v intervalu $\mathcal{I}$, pak je funkce F v $\mathcal{I}$ spojitá.*

Věta 2.2. *Je-li funkce F primitivní funkce k funkci f v intervalu $\mathcal{I}$, pak $\{F + c | c \in \mathbb{R}\}$ je množinou všech primitivních funkcí k funkci f.*



Obrázek 2.1: Primitivní funkce $F_1(x), F_2(x), F_3(x)$ a $F_4(x)$ k funkci $f(x) = 4x^3$ (vlastní tvorba podle [26]).

Věta 2.3. *Nechť f je spojitá funkce na intervalu $\mathcal{I}$. Potom k ní na tomto intervalu existuje primitivní funkce.*

Definice 2.2 (Neurčitý integrál)**.** Symbolem $\int f(x)\,\mathrm{d}x$ označujeme systém všech primitivních funkcí k funkci f a nazýváme jej **neurčitý integrál** funkce f. Potom píšeme

$$\int f(x)\,\mathrm{d}x = F(x) + c, \qquad \text{případně jen} \qquad \int f(x)\,\mathrm{d}x = F(x), \tag{2.2}$$

kde F je některá primitivní funkce funkce f.

Funkce f se nazývá **integrand** nebo též **integrovaná funkce**, argument x **integrační proměnná**. Proces nalezení primitivní funkce k dané funkci nazýváme **integrováním** nebo též **integrací**.

Určitý integrál

Definice 2.3 (Dělení intervalu)**. Dělením** D intervalu $\langle a, b\rangle, a, b \in \mathbb{R}, a < b$, nazveme množinu $D = \{x_0, x_1, \ldots, x_n\}, n \in \mathbb{N}$, takovou, že $a = x_0 < x_1 < x_2 < \cdots < x_n = b$.

Čísla $x_0, x_1, \ldots, x_n$ se nazývají **dělicí body**, intervaly $\langle x_0, x_1\rangle, \langle x_1, x_2\rangle, \ldots, \langle x_{n-1}, x_n\rangle$ se nazývají **dělicí intervaly** dělení D.

Číslo $\nu(D) = \max\{x_i - x_{i-1} | i = 1, 2, \ldots, n\} > 0$ nazýváme **normou** dělení D.

Definice 2.4 (Dolní (resp. horní) Riemannův součet)**.** Nechť je funkce $f : \langle a, b\rangle \to \mathbb{R}, a, b \in \mathbb{R}, a < b$, na intervalu $\langle a, b\rangle$ omezená shora i zdola, tzn. existují konstanty m, M takové,

že pro $\forall x \in \langle a, b\rangle$ platí $m \leq f(x) \leq M$. Nechť $D = \{x_0, x_1, \ldots, x_n\}, n \in \mathbb{N}$, je libovolné dělení intervalu $\langle a, b\rangle, a, b \in \mathbb{R}, a < b$, označme pro $i = 1, 2, \ldots, n$:

$$m_i = \inf\{f(x) | x \in \langle x_{i-1}, x_i\rangle\}, \tag{2.3}$$
$$M_i = \sup\{f(x) | x \in \langle x_{i-1}, x_i\rangle\}. \tag{2.4}$$

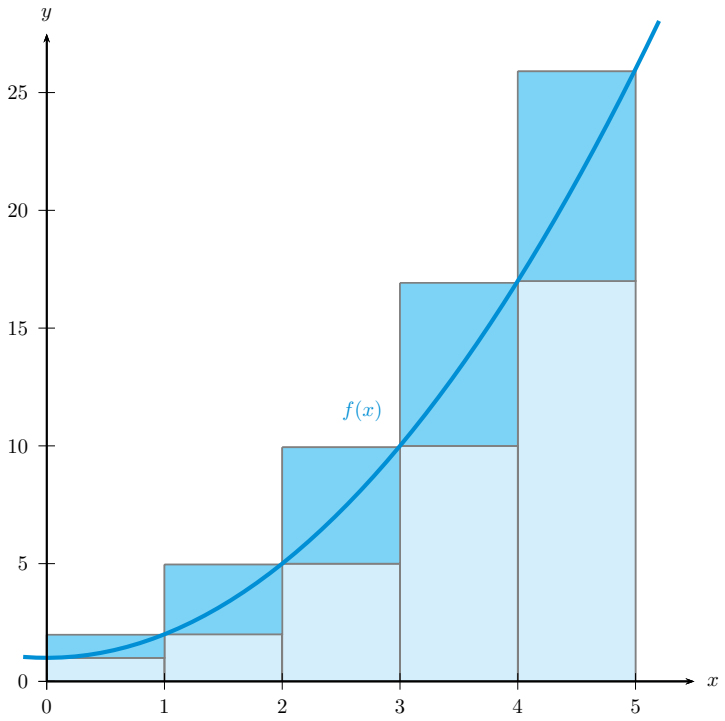
Hodnota

$$s(f, D) = \sum_{i=1}^{n} m_i(x_i - x_{i-1}) \tag{2.5}$$

se nazývá **dolní Riemannův součet** funkce f při dělení D a hodnota

$$\mathcal{S}(f, D) = \sum_{i=1}^{n} M_i(x_i - x_{i-1}) \tag{2.6}$$

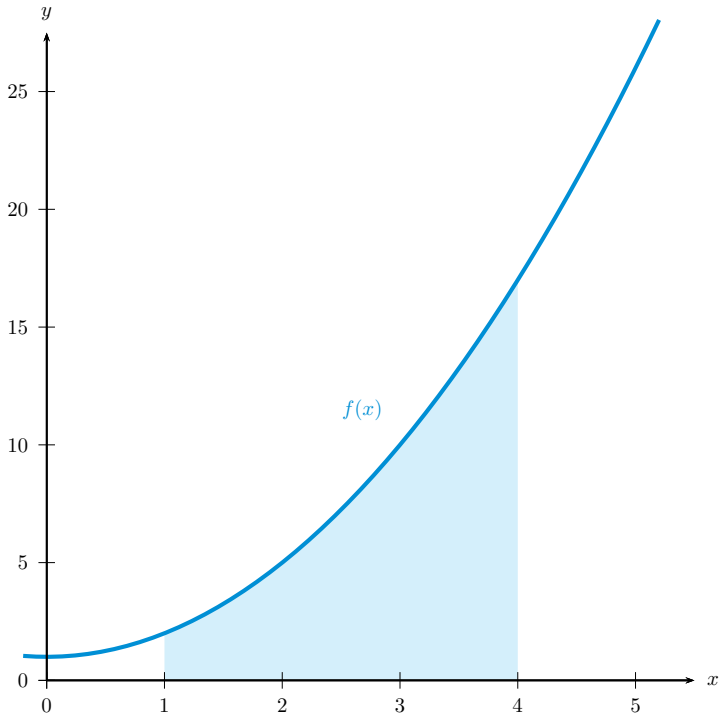
se nazývá **horní Riemannův součet** funkce f při dělení D.



Obrázek 2.2: Dolní (zeleně) a horní (červeně) Riemannův součet pro funkci $f(x) = x^2 + 1$ na intervalu $\langle 0, 5\rangle$ (vlastní tvorba podle [27]).

Definice 2.5 (Určitý (Riemannův) integrál)**.** Nechť $f : \langle a, b\rangle \to \mathbb{R}$ je ohraničená funkce. Řekneme, že f je **integrovatelná** na intervalu $\langle a, b\rangle$, existuje-li číslo $I \in \mathbb{R}$ tak, že ke každému $\varepsilon > 0$ existuje $\delta > 0$ takové, že pro každé dělení D intervalu $\langle a, b\rangle$, jehož norma $\nu(D) < \delta$, platí:

$$|\mathcal{S}(f, D) - I| < \varepsilon. \tag{2.7}$$



Obrázek 2.3: Určitý Riemannův integrál $I = \int_1^4 x^2 + 1 \, \mathrm{d}x$ (vlastní tvorba).

Číslo I nazýváme **určitým (Riemannovým) integrálem** funkce f od a do b a píšeme:

$$I = \int_a^b f(x) \, \mathrm{d}x. \tag{2.8}$$

Věta 2.4 (Newton-Leibnizova věta)**.** *Nechť f je funkce spojitá v $\langle a, b \rangle$. Jestliže v $\langle a, b \rangle$ platí $F'(x) = f(x)$, tj. $\int f(x) \, \mathrm{d}x = F(x) + c$, potom:*

$$\int_a^b f(x) \, \mathrm{d}x = F(b) - F(a) = [F(x)]_a^b \, . \tag{2.9}$$

2.1.2 Analytické řešení

Problém řešení určitých integrálů funkcí jedné proměnné lze díky definici 2.2 o neurčitém integrálu převést na problém hledání primitivní funkce k integrandu a na výslednou primitivní funkci pak aplikovat Newton-Leibnizovu větu 2.4. Pro mnohé funkce existují již vyjádřené primitivní funkce v obecném tvaru, jež se často nazývají **tabulkové integrály**. Při řešení však lze využít i další metody, jako je například **substituční metoda** či **metoda per partes** (neboli metoda částečné integrace). Tento přístup však není možné použít ve všech případech, neboť existují matematické funkce, pro něž neexistuje primitivní funkce a tudíž je není možné analyticky integrovat.

Tabulkové integrály

Problém hledání primitivní funkce se od derivování liší ve dvou důležitých faktech. Za prvé, zatímco derivace elementární funkce je vždy opět elementární funkcí, primitivní funkce k některým elementárním funkcím (např. k e^{x^2}) nejsou elementární. Za druhé, nepatrná změna ve tvaru funkce má za následek nepatrnou změnu v její derivaci, zatímco malá změna ve tvaru funkce může mít za následek podstatnou změnu v její primitivní funkci, např.:

$$\int \frac{1}{1+x^2}\,\mathrm{d}x = \arctan x + c, \quad \text{ale} \quad \int \frac{x}{1+x^2}\,\mathrm{d}x = \frac{1}{2}\ln\left(1+x^2\right) + c. \tag{2.10}$$

Určité integrály lze řešit rozkladem na elementární funkce podle následujících pravidel:

$$\int \left(af(x) \pm bg(x)\right)\mathrm{d}x = a\int f(x)\,\mathrm{d}x \pm b\int g(x)\,\mathrm{d}x, \tag{2.11}$$

$$\int \frac{f'(x)}{f(x)}\,\mathrm{d}x = \ln|f(x)|, \tag{2.12}$$

$$\int_a^b f(x)\,\mathrm{d}x = -\int_b^a f(x)\,\mathrm{d}x, \quad a > b. \tag{2.13}$$

Tento seznam pravidel není úplný, neboť samotný integrální počet není předmětem této práce (cílem této kapitoly je pouze připomenout základní definice, pravidla, přístupy a metody řešení určitých integrálů).

Po aplikaci předchozích pravidel lze určité integrály řešit podle vzorců pro výpočet neurčitých integrálů (převzato z [16]) uvedených v 2.14, kde jsou uvedeny primitivní funkce $F(x)$ k funkcím $f(x)$ takové, aby platilo $F'(x) = f(x)$ a $\int f(x)\,\mathrm{d}x = F(x) + c$:

$$\begin{aligned}
\int 0\,\mathrm{d}x &= c \\
\int 1\,\mathrm{d}x &= x \\
\int x^k\,\mathrm{d}x &= \frac{x^{k+1}}{k+1}, k \neq -1 \\
\int \frac{1}{x}\,\mathrm{d}x &= \ln|x|, x \neq 0 \\
\int \sin x\,\mathrm{d}x &= -\cos x \\
\int \cos x\,\mathrm{d}x &= \sin x \\
\int \frac{1}{\sin^2 x}\,\mathrm{d}x &= -\operatorname{cotan} x \\
\int \frac{1}{\cos^2 x}\,\mathrm{d}x &= \tan x \\
\int e^x\,\mathrm{d}x &= e^x
\end{aligned} \tag{2.14}$$

$$\int a^x \,\mathrm{d}x = \frac{a^x}{\ln a}, a > 0, a \neq 1$$
$$\int \sinh x \,\mathrm{d}x = \cosh x$$
$$\int \cosh x \,\mathrm{d}x = \sinh x$$
$$\int \frac{1}{x^2 + a^2} \,\mathrm{d}x = \frac{1}{a} \arctan \frac{x}{a}, a > 0$$
$$\int \frac{1}{x^2 - a^2} \,\mathrm{d}x = \frac{1}{2a} \ln \left| \frac{x - a}{x + a} \right|, a > 0, |x| \neq a$$
$$\int \frac{1}{\sqrt{x^2 + b}} \,\mathrm{d}x = \ln \left| x + \sqrt{x^2 + b} \right|, b \neq 0$$
$$\int \frac{1}{\sqrt{a^2 - x^2}} \,\mathrm{d}x = \arcsin \frac{x}{a}, |x| < a, a > 0$$
$$\int \sqrt{x^2 + b} \,\mathrm{d}x = \frac{x}{2}\sqrt{x^2 + b} + \frac{b}{2} \ln |x + \sqrt{x^2 + b}|, b \neq 0$$

Substituce

Je-li F primitivní funkce k funkci f na nějakém intervalu $\mathcal{I}$, je možné psát integrál $\int f(t) \,\mathrm{d}t$ ve tvaru

$$\int f(t) \,\mathrm{d}t = \int F'(t) \,\mathrm{d}t = \int \mathrm{d}F(t), \tag{2.15}$$

kde v posledním integrálu vystupuje diferenciál primitivní funkce F. Pokud $t = g(x)$, pak z věty o derivaci složené funkce $(F(g(x)))' = F'(g(x))g'(x)$ vyplývá pro diferenciál $\mathrm{d}F(t)$:

$$\mathrm{d}F(t) = \mathrm{d}F(g(x)) = F'(g(x))g'(x) \,\mathrm{d}x = f(g(x))g'(x) \,\mathrm{d}x, \tag{2.16}$$

z čehož vyplývá

$$\int f(t) \,\mathrm{d}t = \int f(g(x))g'(x) \,\mathrm{d}x, \quad \text{kde} \quad t = g(x). \tag{2.17}$$

Tato metoda se nazývá **substituční metoda** a jedná se o vůbec jednu z nejvýznamnějších integračních metod.

Integrace per partes

Metoda **integrace per partes** (neboli integrace po částech) se využívá pro integrování součinu funkcí. Tato metoda je založena na větě o derivaci součinu $(uv)' = u'v + uv'$. Uplatněním této věty na podmínky pro integrál vyplývá následující:

$$\int (uv)' \,\mathrm{d}x = \int u'v \,\mathrm{d}x + \int uv' \,\mathrm{d}x, \tag{2.18}$$

z čehož vyplývá

$$\int uv' \,\mathrm{d}x = uv - \int u'v \,\mathrm{d}x. \tag{2.19}$$

Při integraci lze tuto metodu použít opakovaně, dokud je to pro řešení problému vhodné.

2.1.3 Numerické řešení

V předchozí části bylo řečeno, že ne pro všechny funkce existuje primitivní funkce. Existují však také funkce, pro které je nalezení primitivní funkce velmi náročné až nemožné. Tyto problémy řeší numerické metody, které si kladou za cíl poměrně snadnou a přímočarou metodou rozkladu plochy pod funkcí na menší části, jejichž plochu dokážeme vypočítat, určit celkovou plochu pod funkcí, resp. určitý integrál dané funkce v určitém intervalu.

Numerická integrace je založena na myšlence nahrazení integrandu interpolačním (např. Lagrangeovým) či aproximačním polynomem (např. Taylorova řada), jež je posléze integrován. Zřejmě platí, že čím je přesnější tento polynom, tím je přesnější i aproximace určitého integrálu.

Definice a věty uvedené v této kapitole jsou inspirovány zejména [2, 5, 25], dále také [1, 3, 6, 7, 30] a následně vhodně upraveny tak, aby byly vzájemně konzistentní, co se týče používané terminologie.

Lagrangeův interpolační polynom

Připomeňme konstrukci Lagrangeova interpolačního polynomu. Předpokládejme, že chceme interpolovat libovolnou funkci na množině bodů $x_0, x_1, \ldots, x_n$. Nejprve definujme systém $n+1$ speciálních polynomů stupně n známých v teorii interpolace jako *kardinální polynomy*. Ty jsou označeny jako $l_0, l_1, \ldots, l_n$ a mají následující vlastnost:

$$l_i(x_j) = \delta_{ij} = \begin{cases} 0 & \text{pokud } i \neq j \\ 1 & \text{pokud } i = j \end{cases} \tag{2.20}$$

Jakmile jsou tyto polynomy k dispozici, je možné interpolovat *libovolnou* funkci f **Lagrangeovým interpolačním polynomem**:

$$L_n(x) = \sum_{i=0}^{n} l_i(x) f(x_i). \tag{2.21}$$

Tato funkce L_n je lineární kombinací polynomů l_i a je samotná polynomem stupně nejvýše n. Dále lze vyjádřit L_n v bodě x_j:

$$L_n(x_j) = \sum_{i=0}^{n} l_i(x_j) f(x_i) = l_j(x_j) f(x_j) = f(x_j). \tag{2.22}$$

Proto je L_n interpolačním polynomem pro funkci f v bodech $x_0, x_1, \ldots, x_n$. Dále zbývá jen vyjádřit rovnici pro **kardinální polynom** l_i:

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right), \quad (0 \leq i \leq n). \tag{2.23}$$

Dále je možné určit chybu interpolace vztahem, který obsahuje derivaci interpolované funkce f stupně $(n+1)$.

Definice 2.6. Pokud L je polynom stupně nejvýše n, který interpoluje funkci f v $n+1$ různých bodech $x_0, x_1, \ldots, x_n$ patřících do intervalu $\langle a, b \rangle$ a pokud $f^{(n+1)}$ je spojitá, pak pro každé x v intervalu $\langle a, b \rangle$ existuje $\xi \in (a, b)$, pro které platí:

$$f(x) - L(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i). \tag{2.24}$$

Newton-Cotesovy kvadraturní vzorce

Newton-Cotesovy vzorce je skupina vztahů pro numerickou integraci, jež je založená na vyhodnocování integrandu na ekvidistantní síti bodů. Předpokládejme, že funkce f je definována na intervalu $\langle a, b\rangle$ ve stejně vzdálených bodech x_i pro $i = 0, \ldots, n$, kde $x_0 = a$ a $x_n = b$. Existují dva typy Newton-Cotesových vzorců – (a) *uzavřené*, které využívají všechny body $x_0, \ldots, x_n$, a (b) *otevřené*, které nevyužívají krajní hodnoty x_0 a x_n.

Uzavřené Newton-Cotesovy vzorce stupně n pak mají tvar:

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=0}^{n} w_i f(x_i), \tag{2.25}$$

kde $x_i = x_0 + hi$ s krokem $h = \frac{x_n - x_0}{n} = \frac{b-a}{n}$ a vahami w_i. Z následujícího vztahu je zřejmé, že tyto váhy w_i jsou odvozeny z *Lagrangeových kardinálních polynomů* (viz vztah 2.23), což znamená, že závisí pouze na hodnotách x_i a ne na funkci f:

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x \approx \int_a^b L(x)\,\mathrm{d}x = \int_a^b \left(\sum_{i=0}^{n} f(x_i) l_i(x)\right)\,\mathrm{d}x = \sum_{i=0}^{n} f(x_i) \int_a^b l_i(x)\,\mathrm{d}x = \\ = \sum_{i=0}^{n} w_i f(x_i).\end{aligned} \tag{2.26}$$

Pro **otevřené Newton-Cotesovy vzorce** platí podobné odvození, kdy však nejsou uvažovány krajní body $x_0 = a$ a $x_n = b$:

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n-1} w_i f(x_i). \tag{2.27}$$

Obdélníková metoda

Mezi nejjednodušší složené Newton-Cotesovy vzorce patří **obdélníková metoda** (tj. Newton-Cotesův vzorec 0. stupně). To znamená, že interval $\langle a, b\rangle$ je rozdělen na n stejných podintervalů a v každém z těchto podintervalů je funkce f nahrazena Lagrangeovým polynomem 0. stupně (tedy konstantní funkcí). Hodnota této konstantní funkce je stejná jako funkční hodnota funkce f v polovině daného podintervalu. Celkový integrál je roven součtu ploch obdélníků nad všemi podintervaly (viz obrázek 2.4).

Na celém intervalu $\langle a, b\rangle$ pak platí:

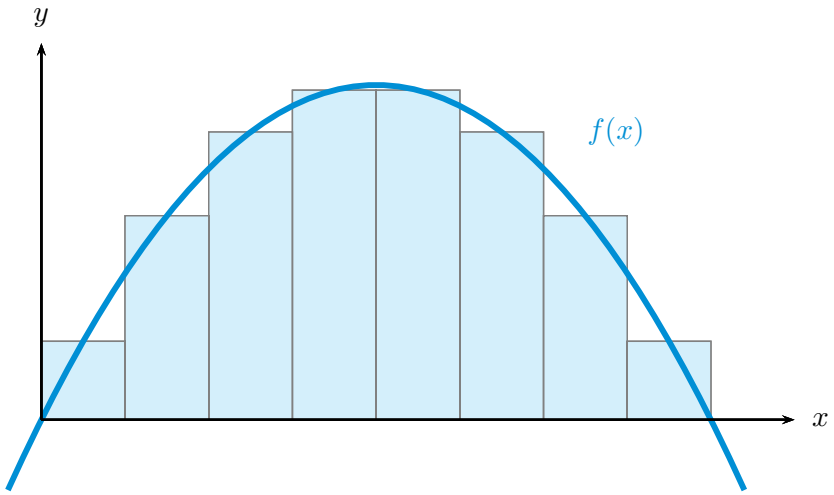
$$\int_a^b f(x)\,\mathrm{d}x \approx h \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right). \tag{2.28}$$

Lichoběžníková metoda

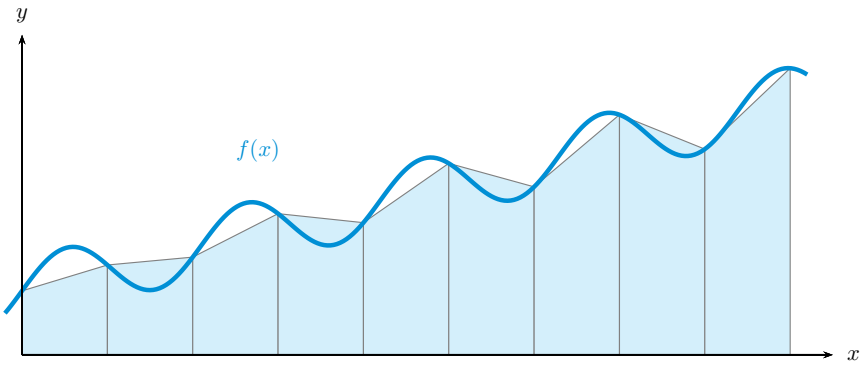
Složená **lichoběžníková metoda** využívá Newton-Cotesovy vzorce 1. stupně, kdy funkci f na každém podintervalu $\langle x_i, x_{i+1}\rangle$ nahradí Lagrangeovým polynomem 1. stupně, tedy lineární funkcí. Nad každým podintervalem tak vznikne lichoběžník, jehož plocha aproximuje plochu pod funkcí f na daném podintervalu (viz obrázek 2.5).

Na celém intervalu $\langle a, b\rangle$ (resp. $\langle x_0, x_n\rangle$) pak platí:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{2} \sum_{i=0}^{n-1} \left(f(x_i) + f(x_{i+1})\right) = \frac{h}{2}\left(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right). \tag{2.29}$$



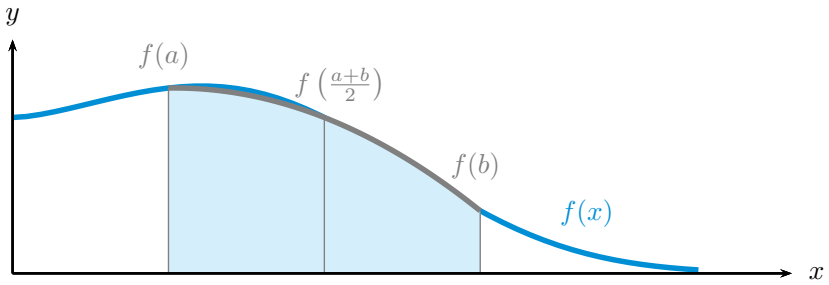
Obrázek 2.4: Složená obdélníková metoda (vlastní tvorba podle [2]).



Obrázek 2.5: Složená lichoběžníková metoda (vlastní tvorba podle [2]).

Simpsonova metoda

Složená **Simpsonova metoda** (neboli *metoda parabol*) je založena na Newton-Cotesových vzorcích 2. stupně a tudíž je integrovaná funkce f nahrazena na každém podintervalu polynomem 2. stupně, tedy parabolou (viz obrázek 2.6).



Obrázek 2.6: Složená Simpsonova metoda (vlastní tvorba podle [2]).

Na celém intervalu $\langle a, b\rangle$ (při dělení na sudý počet $2n$ podintervalů) pak platí:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\left(f(x_0) + 2\sum_{i=1}^{n-1} f(x_{2i}) + 4\sum_{i=1}^{n} f(x_{2i-1}) + f(x_{2n})\right). \tag{2.30}$$

2.1.4 Převod na obyčejnou diferenciální rovnici

Řešení určitého integrálu $F(x) = \int_a^b f(x)\,\mathrm{d}x$ lze podle [8] převést na řešení obyčejné diferenciální rovnice s počáteční podmínkou:

$$F'(x) = f(x), \quad F(a) = 0. \tag{2.31}$$

Definice 2.7 (Diferenciální rovnice)**. Diferenciální rovnicí řádu** n rozumíme rovnici tvaru

$$F(x, y, y', y'', \ldots, y^{(n-1)}, y^{(n)}) = 0, \tag{2.32}$$

kde F je reálná funkce $n + 2$ proměnných.

Definice 2.8 (Řešení diferenciální rovnice)**. Řešením diferenciální rovnice** 2.32 rozumíme funkci y definovanou na nějakém neprázdném otevřeném intervalu I, která má v každém bodě intervalu I vlastní $n-$tou derivaci a jejíž hodnoty spolu s hodnotami derivací splňují rovnici 2.32 v každém bodě intervalu I, tj. pro každé $x \in I$ platí

$$F(x, y(x), y'(x), y''(x), \ldots, y^{(n-1)}(x), y^{(n)}(x)) = 0. \tag{2.33}$$

Je-li funkce y řešením rovnice 2.32 na intervalu I a funkce $\tilde{y}$ řešením rovnice 2.32 na intervalu $\tilde{I}$, kde $I \subset \tilde{I}, I \neq \tilde{I}$, a $y(x) = \tilde{y}(x)$ pro všechna $x \in I$, pak říkáme, že řešení $\tilde{y}$ je **prodloužením řešení** y na interval $\tilde{I}$. Řešení rovnice 2.32, které nemá prodloužení, nazýváme **maximálním řešením** rovnice 2.32.

Definice 2.9 (Obyčejná diferenciální rovnice 1. řádu)**.** Nechť $F(x, y, z)$ je funkce tří proměnných, která je definovaná na otevřené množině $\Omega \subset \mathbb{R}^3$. Pak rovnice

$$F(x, y, y') = 0 \tag{2.34}$$

se nazývá **obyčejná diferenciální rovnice** 1. **řádu v implicitním tvaru** s neznámou $y(x)$.

Při vyšetřování vlastností obyčejných diferenciálních rovnic je důležitý případ, kdy lze z rovnice 2.34 osamostatnit y', tj. získat tzv. **explicitní tvar obyčejné diferenciální rovnice** 1. **řádu**

$$y' = f(x, y), \tag{2.35}$$

kde $f(x, y)$ funkce dvou proměnných definovaná na otevřené množině $\Omega \subset \mathbb{R}^2$.

Definice 2.10 (Cauchyova počáteční úloha)**.** Nechť $(x_0, y_0) \in \Omega$. Pak úloha najít řešení $y(x)$ diferenciální rovnice 2.35, která je definovaná na nějakém intervalu I obsahujícím bod x_0 a které splňuje tzv. počáteční podmínku $y(x_0) = y_0$, se nazývá **Cauchyova počáteční úloha**.

Věta 2.5 (Existenční)**.** *Nechť $f(x, y)$ je spojitá na otevřené množině Ω. Pak pro každé $(x_0, y_0) \in \Omega$ má úloha*

$$y' = f(x, y), \quad y(x_0) = y_0 \tag{2.36}$$

alespoň jedno řešení.

Definice 2.11 (Lipschitzova podmínka)**.** Existují konstanta $K > 0$ a okolí O bodu (x_0, y_0), $O \subset \Omega$, takové, že pro každé dva body $(x, y_1) \in O, (x, y_2) \in O$ je

$$|f(x, y_1) - f(x, y_2)| \leq K \ |y_1 - y_2|. \tag{2.37}$$

Věta 2.6 (O existenci a jednoznačnosti)**.** *Nechť $f(x, y)$ je spojitá na otevřené množině $\Omega \subset \mathbb{R}^2$ a v každém bodě je splněna **Lipschitzova podmínka**. Pak pro libovolný bod $(x_0, y_0) \in \Omega$ má úloha 2.36 právě jedno řešení.*

Existuje několik analytických způsobů řešení obyčejných diferenciálních rovnic 1. řádu. Ty však zde nebudou rozebírány, neboť všechny uváděné definice a metody směřují k zavedení nové numerické metody. Numerickým řešením získáme pouze přibližné řešení obyčejných diferenciálních rovnic, které je však dostačující, pokud by nalezení přesného analytického řešení bylo náročné nebo pokud by analytické řešení neexistovalo.

Společným znakem všech dále uvedených metod je, že řešení nehledáme jako spojitou funkci definovanou na celém zkoumaném intervalu $I = \langle a, b \rangle$, ale hodnoty přibližného řešení počítáme pouze v konečném počtu bodů $a = x_0 < x_1 < \cdots < x_n = b$. Těmto bodům říkáme **uzlové body** (nebo **uzly sítě**) a množině $\{x_0, x_1, \ldots, x_n\}$ říkáme **síť**. Rozdíl $h_i = x_{i+1} - x_i$ se nazývá **krok** sítě v uzlu x_i. Je-li krok h_i stejný pro všechny uzlové body x_i, pak hovoříme o **pravidelné (ekvidistantní) síti**. Přibližné hodnoty řešení v uzlových bodech, vypočtené numerickou metodou, budeme značit $y_0, y_1, \ldots, y_n$, na rozdíl od přesného analytického řešení, které budeme značit $y(x_0), y(x_1), \ldots, y(x_n)$.

Metody, které k vyjádření hodnoty v dalším uzlovém bodě používají pouze informace z jediného předchozího uzlového bodu, se nazývají **jednokrokové metody**. Pokud se pro vyjádření hodnoty v dalším uzlovém bodě používají informace z více předcházejících uzlových bodů, nazývají se tyto metody **vícekrokové**.

Jednokrokové metody

Eulerova metoda Mějme dánu počáteční úlohu 2.36 a ekvidistantní síť $\{x_0, x_1, \ldots, x_n\}$ s krokem h. Ve všech bodech sítě by podle rovnice 2.36 mělo platit

$$y'(x_i) = f(x_i, y(x_i)). \tag{2.38}$$

Derivaci na levé straně této rovnice můžeme nahradit (dopřednou) diferencí:

$$\frac{y(x_{i+1}) - y(x_i)}{h} \doteq f(x_i, y(x_i)). \tag{2.39}$$

Nahradíme-li $y(x_i)$ přibližnou hodnotou y_i, můžeme odtud vyjádřit přibližnou hodnotu $y(x_{i+1})$ jako

$$y_{i+1} = y_i + h f(x_i, y_i). \tag{2.40}$$

Pomocí tohoto vzorce vypočteme přibližnou hodnotu řešení v dalším uzlovém bodě pomocí hodnoty v uzlu předchozím. Hodnotu řešení v bodě x_0 známe z počáteční podmínky; je rovna y_0.

Modifikace Eulerovy metody Existuje několik modifikací Eulerovy metody, které fungují na stejném principu jako metoda Eulerova, avšak výpočet hodnoty v následujícím uzlovém bodě je důmyslnější a používá pomocné hodnoty k_1, k_2.

První modifikace Eulerovy metody (známá též jako *midpoint Euler formula*) je vyjádřena vztahy:

$$\begin{aligned} k_1 &= f\left(x_n, y_n\right) \\ k_2 &= f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right) \\ y_{n+1} &= y_n + hk_2. \end{aligned} \tag{2.41}$$

Druhá modifikace Eulerovy metody (známá též jako *Heunova metoda*) je vyjádřena vztahy:

$$\begin{aligned} k_1 &= f\left(x_n, y_n\right) \\ k_2 &= f\left(x_n + h, y_n + hk_1\right) \\ y_{n+1} &= y_n + \frac{1}{2}h(k_1 + k_2). \end{aligned} \tag{2.42}$$

Obě tyto modifikace jsou Eulerovy metody druhého řádu.

Runge-Kuttovy metody Tyto numerické metody jsou jedna z nejdůležitějších a nejpoužívanějších skupin jednokrokových metod. Výše uvedené dvě modifikace Eulerovy metody jsou jednoduchými příklady této skupiny metod.

Obecný tvar Runge-Kuttovy metody je

$$y_{n+1} = y_n + h(w_1k_1 + \cdots + w_sk_s), \tag{2.43}$$

kde

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_i &= f(x_n + \alpha_i h, y_n + h\sum_{j=1}^{i-1} \beta_{ij}k_j), \quad i = 2, \ldots, s \end{aligned} \tag{2.44}$$

a w_i, α_i a β_{ij} jsou konstanty volené tak, aby metoda měla maximální řád.

U první modifikované Eulerovy metody byly tyto konstanty zvoleny jako $w_1 = 0, w_2 = 1, \alpha_2 = \frac{1}{2}$ a $\beta_{21} = \frac{1}{2}$. U druhé modifikace pak $w_1 = w_2 = \frac{1}{2}, \alpha_2 = 1$ a $\beta_{21} = 1$.

Runge-Kuttova metoda 4. **řádu** Nejproslulejší a nejpoužívanější metoda ze skupiny Runge-Kuttových metod je právě **metoda Runge-Kutta** 4. **řádu**. Navýšením řádu metody získáme přesnější hodnotu následujícího uzlového bodu, než kdybychom použili Eulerovu metodu či její modifikace. Koeficienty pro tuto metodu jsou nastaveny takto:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= f(x_n, y_n) \\ k_2 &= f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1\right) \\ k_3 &= f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2\right) \\ k_4 &= f\left(x_n + h, y_n + hk_3\right). \end{aligned} \tag{2.45}$$

Vícekrokové metody

U vícekrokových metod počítáme přibližné řešení v dalším uzlovém bodě sítě pomocí několika uzlů předchozích. Protože přitom používáme nejen hodnoty přibližného řešení, ale také hodnoty pravé strany $f(x, y)$ v těchto bodech, budeme kvůli snadnějšímu zápisu používat značení $f_j = f(x_j, y_j)$. **Lineární $k-$krokové metoda** má obecný předpis:

$$y_{n+1} = a_1 y_n + a_2 y_{n-1} + \cdots + a_k y_{n-k+1} + h(b_0 f_{n+1} + b_1 f_n + \cdots + b_k f_{n-k+1}), \qquad (2.46)$$

kde k je přirozené číslo a alespoň jedna z konstant a_k, b_k je různá od nuly.

Zřejmou nevýhodou $k-$krokové metody je, že řešení v prvních k uzlových bodech $x_0, \ldots, x_{k-1}$ musíme získat nějakým jiným způsobem. K tomuto účelu se zpravidla používá jednokroková metoda stejného řádu přesnosti, jaký má dále použitá vícekroková metoda.

Je-li $b_0 = 0$, metoda 2.46 se nazývá **explicitní**. V tomto případě můžeme hodnotu v novém uzlovém bodě přímo vypočítat dosazením do vzorce 2.46. Je-li $b_0 \neq 0$, metoda 2.46 se nazývá **implicitní**. Pak se na pravé straně rovnice 2.46 kromě známých hodnot vyskytuje také $f_{n+1} = f(x_{n+1}, y_{n+1})$, takže y_{n+1} nemůžeme vypočítat přímo, ale v každém kroku musíme řešit rovnici

$$y_{n+1} = hb_0 f(x_{n+1}, y_{n+1}) + g \qquad (2.47)$$

s neznámou y_{n+1}, kde $g = \sum_{j=1}^{k} a_j y_{n-j+1} + h \sum_{j=1}^{k} b_j f_{n-j+1}$ je známé číslo (v každém kroku jiné). V případě některých pravých stran f tuto rovnici vyřešíme přesně, obecně je však potřeba tuto rovnici řešit numericky, většinou metodou prosté iterace. Tato nevýhoda je však vyvážena příznivými vlastnostmi implicitních metod. Tyto metody jsou při daném k přesnější a jsou také stabilnější než explicitní metody.

Vícekrokové metody založené na numerické integraci Řešenou rovnici $y'(x) = f(x, y(x))$ můžeme zintegrovat na intervalu $\langle x_{n+1-s}, x_{n+1} \rangle$:

$$y(x_{n+1}) - y(x_{n+1-s}) = \int_{x_{n+1-s}}^{x_{n+1}} f(x, y(x)) \, \mathrm{d}x. \qquad (2.48)$$

Funkci f nahradíme interpolačním polynomem a ten zintegrujeme. Podle toho, jak zvolíme s a uzly interpolace, získáváme různé metody. Pokud funkci f nahradíme interpolačním polynomem s uzly $x_{n+1-k}, \ldots, x_n$, resp. s uzly $x_{n+1-k}, \ldots, x_{n+1}$, a rovnici zintegrujeme přes interval $\langle x_n, x_{n+1} \rangle$ (tzn. $s = 1$), získáme tím explicitní, resp. implicitní, $k-$krokovou metodu.

Adams-Bashforthovy metody Jsou explicitní lineární $k-$krokové metody získané výše popsaným způsobem.

Nejjednodušším případem této metody, kdy $k = 1$, je Eulerova metoda. V tomto případě funkci f nahradíme konstantou f_n. Integrací přes interval $\langle x_n, x_{n+1} \rangle$ získáme vztah $y_{n+1} = y_n + hf_n$. Obecný tvar Adams-Bashforthových metod je

$$y_{n+1} = y_n + h(b_1 f_n + b_2 f_{n-1} + \cdots + b_k f_{n-k+1}). \qquad (2.49)$$

Přehled koeficientů b_i pro $k = 1, \ldots, 4$ je uveden v následující tabulce:

k	b_1	b_2	b_3	b_4
1	1			
2	$\frac{3}{2}$	$-\frac{1}{2}$		
3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$	
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$

Tabulka 2.1: Koeficienty pro Adams-Bashforthovy metody (převzato z [5]).

Adams-Moultonovy metody Pokud za uzel interpolace vezmeme i x_{n+1}, získáme Adams-Moultonovy metody. Nejjednodušší z těchto metod je implicitní Eulerova metoda $y_{n+1} = y_n + hf_{n+1}$. Obecný tvar těchto metody je

$$y_{n+1} = y_n + h(b_0 f_{n+1} + b_1 f_n + b_2 f_{n-1} + \cdots + b_k f_{n-k+1}). \tag{2.50}$$

Přehled koeficientů b_i pro $k = 0, \ldots, 3$ je uveden v následující tabulce:

k	b_0	b_1	b_2	b_3
0	1			
1	$\frac{1}{2}$	$\frac{1}{2}$		
2	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$	
3	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$

Tabulka 2.2: Koeficienty pro Adams-Moultonovy metody (převzato z [5]).

Metody prediktor-korektor Jak již bylo řečeno, při použití implicitních vícekrokových metod je potřeba v každém kroku vypočítat y_{n+1} jako řešení rovnice

$$y_{n+1} = hb_0 f(x_{n+1}, y_{n+1}) + g, \tag{2.51}$$

kde $g = \sum_{j=1}^{k} a_j y_{n-j+1} + h \sum_{j=1}^{k} b_j f_{n-j+1}$. Tento tvar rovnice je vhodný pro použití metody prosté iterace. K hledané hodnotě se můžeme postupně přibližovat iteračním procesem

$$y_{n+1}^{(r+1)} = hb_0 f(x_{n+1}, y_{n+1}^{(r)}) + g. \tag{2.52}$$

Zbývá tedy otázka, jak získat dobrou počáteční aproximaci $y_{n+1}^{(0)}$. K tomu se nabízí použití explicitní vícekrokové metody.

Princip **metody prediktor-korektor** je tedy následující: V každém kroku nejprve vypočteme počáteční aproximaci $y_{n+1}^{(0)}$ pomocí explicitní vícekrokové metody – **prediktoru**. Tuto hodnotu zpřesníme použitím implicitní vícekrokové metody – **korektoru**, a to dosazením $y_{n+1}^{(0)}$ do 2.52 (s tím, že $r = 0$). Tím dostaneme $y_{n+1}^{(1)}$. Tuto hodnotu bychom opět mohli dosadit do 2.52, ale obvykle se korektor používá v každém kroku jen jednou.

Jako dvojici prediktor-korektor volíme zpravidla explicitní a implicitní metodu téhož řádu.

Taylorův polynom a obyčejná diferenciální rovnice

Taylorův rozvoj je reprezentace funkce f jako nekonečný součet členů, jež jsou vypočteny z hodnot derivací funkce f v daném bodě. Pokud je tento počet konečný, pak hovoříme o **Taylorově polynomu**. Pro funkci $f(x)$ v bodě a pak Taylorův rozvoj lze vyjádřit následovně:

$$f(x) = f(a) + \frac{f^{(1)}(a)}{1!}(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \cdots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n. \tag{2.53}$$

Výše uvedené definice lze podle [8, 17] využít při řešení obyčejných diferenciálních rovnic s počáteční podmínkou. To probíhá tak, pro síť ekvidistantních bodů $a = x_0, \ldots, x_n = b$ na intervalu $\langle a, b\rangle$ hledáme hodnoty funkce $F(x_i)$ (viz vztah 2.31). Jelikož se jedná o ekvidistantní síť bodů, uvažujme konstantní vzdálenost dvou po sobě jdoucích bodů $h = x_{i+1} - x_i$. Jednotlivé body x_i jsou vyjádřeny jako $x_i = a + ih$.

$$\begin{aligned} F(x_1) &= F(x_0) + \frac{h}{1!}F^{(1)}(x_0) + \frac{h^2}{2!}F^{(2)}(x_0) + \cdots \\ F(x_2) &= F(x_1) + \frac{h}{1!}F^{(1)}(x_1) + \frac{h^2}{2!}F^{(2)}(x_1) + \cdots \\ &\vdots \\ F(x_n) &= F(x_{n-1}) + \frac{h}{1!}F^{(1)}(x_{n-1}) + \frac{h^2}{2!}F^{(2)}(x_{n-1}) + \cdots \end{aligned} \tag{2.54}$$

Hodnota určitého integrálu $f(x)$ na intervalu $\langle a, b\rangle$ je pak rovna hodnotě členu $F(x_n)$:

$$\int_a^b f(x)\,\mathrm{d}x = F(x_n) = F(x_{n-1}) + \frac{h}{1!}F^{(1)}(x_{n-1}) + \frac{h^2}{2!}F^{(2)}(x_{n-1}) + \cdots . \tag{2.55}$$

2.2 Integrál v prostoru obecné dimenze

V následující kapitole budou podle [11–13, 15, 16] zobecněny definice a pojmy uvedené v předchozí části na integrály funkcí více proměnných, neboli integrály v prostoru obecné dimenze.

2.2.1 Definice pojmů

Definice 2.12 (Funkce n proměnných)**. Reálná funkce n reálných proměnných** $f : \mathbb{R}^n \to \mathbb{R}$ je zobrazení, které každému $x \in \mathbb{R}^n$ přiřadí nejvýše jedno $f(x) \in \mathbb{R}$. Prvky $x = [x_1, \ldots, x_n] \in \mathbb{R}^n$ se nazývají **body** $n-$rozměrného prostoru $\mathbb{R}^n$. Množina $D(f) = \{x \in \mathbb{R}^n | \exists y \in \mathbb{R} : f(x) = y\}$ se nazývá **definiční obor** funkce f. Množina $H(f) = \{y \in \mathbb{R} | \exists x \in D(f) : f(x) = y\}$ se nazývá **obor hodnot** funkce f. Množina $G(f) = \{[x_1, \ldots, x_n, f(x_1, \ldots, x_n)] \in \mathbb{R}^{n+1} | [x_1, \ldots, x_n] \in D(f)\}$ se nazývá **graf** funkce f.

Definice 2.13. Buď $A = \langle a_1, b_1\rangle \times \cdots \times \langle a_n, b_n\rangle \subset \mathbb{R}^n$ $n-$rozměrný uzavřený interval a $f : \mathbb{R}^n \to \mathbb{R}$ funkce ohraničená na $A \subset D(f)$. Definujme

- $|A| = (b_1 - a_1) \cdot \ldots \cdot (b_n - a_n)$ **objem** A.
- $d(A) = \sqrt{(b_1 - a_1)^2 + \cdots + (b_n - a_n)^2}$ **průměr** A.

- Pro $i = 1, \ldots, n$ buď $D_i : a_i = x_i^{(0)} < x_i^{(1)} < \cdots < x_i^{(m_i)} = b_i$ tzv. **dělení** $\langle a_i, b_i \rangle$.
- $D = [D_1, \ldots, D_n]$ se nazývá **dělení** A.

Dále postupujeme následovně:

1. Dělení D rozloží A na $m = m_1 \cdot \cdots \cdot m_n$ $n-$rozměrných intervalů
$$A_{k_1,\ldots,k_n} = \left\langle x_1^{(k_1-1)}, x_1^{(k_1)} \right\rangle \times \cdots \times \left\langle x_n^{(k_n-1)}, x_n^{(k_n)} \right\rangle, \tag{2.56}$$
kde $1 \leq k_i \leq m_i$ a $i = 1, \ldots, n$. Označme tyto intervaly pro zjednodušení $A^{(1)}, \ldots, A^{(m)}$.
2. V každém intervalu $A^{(j)}$ pro $j = 1, \ldots, m$ zvolme bod (tj. reprezentanta intervalu $A^{(j)}$) $y_j \in A^{(j)}$.
3. Položme $\|D\| = \max\{d(A^{(j)}) | j = 1, \ldots, m\}$. $\|D\|$ je tzv. **norma dělení** D.
4. Nyní každému $k \in \mathbb{N}$ přiřaďme dělení $D(k)$ intervalu A. Posloupnost $\{D(k)\}_{k=1}^{\infty}$ se nazývá **nulová posloupnost**, když $\|D(k)\| \to 0$.
5. Definujme $S_f(D) = \sum_{j=1}^{m} f(y_j)|A^{(j)}|$. Číslo $S_f(D)$ se nazývá **integrální součet** funkce f pro dělení D intervalu A a pro danou volbu reprezentantů y_j.

Definice 2.14. Řekneme, že ohraničená funkce f je Riemannovsky integrovatelná na A a číslo $a \in \mathbb{R}$ nazveme $n-$**rozměrný Riemannův integrál funkce** f **na množině** A, když pro každou nulovou posloupnost $D(k)$ dělení intervalu A a pro každou volbu reprezentantů v těchto děleních platí
$$\lim_{k\to\infty} S_f(D(k)) = a. \tag{2.57}$$

Integraci funkcí více proměnných lze zavést podobně jako pro funkce jedné proměnné. Integrace probíhá na uzavřeném $n-$rozměrném intervalu Ω. Je-li $f(x_1, x_2, \ldots, x_n)$ funkcí n proměnných, pak její integrál na uzavřeném n-rozměrném intervalu Ω označujeme jako **vícerozměrný (n-rozměrný) integrál**, přičemž jej zapíšeme některým z následujících způsob:
$$\begin{aligned} I &= \int \cdots \int_{\Omega} f(x_1, x_2, \ldots, x_n)\, \mathrm{d}\Omega \\ &= \int_{a_n}^{b_n} \cdots \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(x_1, x_2, \ldots, x_n)\, \mathrm{d}x_1\, \mathrm{d}x_2 \ldots\, \mathrm{d}x_n \\ &= \int \cdots \int_{\Omega} f(x_1, x_2, \ldots, x_n)\, \mathrm{d}^n x \\ &= \int_{\Omega} f(x_1, x_2, \ldots, x_n)\, \mathrm{d}\Omega \end{aligned} \tag{2.58}$$

Definice 2.15 (Elementární oblast)**.** Množina $\Omega \subseteq \mathbb{R}^n$ se nazývá **elementární oblast typu** $(x_1, \ldots, x_n)$, když každý bod $[x_1, \ldots, x_n] \in \Omega$ splňuje nerovnosti
$$\begin{aligned} a_1 \leq & x_1 \leq a_2 \\ g_1(x_1) \leq & x_2 \leq h_1(x_1) \\ g_2(x_1, x_2) \leq & x_3 \leq h_2(x_1, x_2) \\ & \vdots \\ g_{n-1}(x_1, \ldots, x_{n-1}) \leq & x_n \leq h_{n-1}(x_1, \ldots, x_{n-1}), \end{aligned} \tag{2.59}$$

kde $a_1, a_2 \in \mathbb{R}, a_1 < a_2$, a pro každé $i = 1, \ldots, n-1$ jsou $g_i, h_i : \mathbb{R}^i \to \mathbb{R}$ spojité funkce splňující podmínku $g_i < h_i$ pro vnitřní body Ω.

Buď σ permutace množiny $\{x_1, \ldots, x_n\}$. Pokud v předchozích nerovnostech píšeme $\sigma(x_i)$ místo x_i, pak Ω se nazývá **elementární oblast typu** $(\sigma(x_1), \ldots, \sigma(x_n))$.

Věta 2.7 (Fubiniova věta)**.** *Buď $\Omega \subseteq \mathbb{R}^n$ elementární oblast typu $(x_1, \ldots, x_n)$ a nechť funkce f je Riemannovsky integrovatelná na Ω. Pak*

$$\int_\Omega f(x_1, \ldots, x_n)\,\mathrm{d}x_1 \ldots \mathrm{d}x_n = \int_{a_1}^{a_2} \left(\int_{g_1(x_1)}^{h_1(x_1)} \left(\cdots \left(\int_{g_{n-1}(x_1,\ldots,x_{n-1})}^{h_{n-1}(x_1,\ldots,x_{n-1})} f(x_1, \ldots, x_n)\,\mathrm{d}x_n \right) \cdots \right) \mathrm{d}x_2 \right) \mathrm{d}x_1. \tag{2.60}$$

Pro typ $(\sigma(x_1), \ldots, \sigma(x_n))$ platí analogické tvrzení.

2.2.2 Numerické řešení

Jak již bylo vzpomenuto při popisu numerických metod řešení integrálů funkcí jedné proměnné, i pro vícenásobné integrály existují varianty těchto metod. Mnohdy je však výpočet pomocí těchto numerických metod natolik komplikovaný, že pro použití nejsou výhodné. Existuje však metoda, která je velmi úspěšně použitelná bez ohledu na počet dimenzí – *metoda Monte Carlo*. Návrh další numerické metody pro řešení integrálů funkcí více proměnných je předmětem následující kapitoly.

Monte Carlo integrace

Pro výpočet určitého integrálu reálné funkce n proměnných na uzavřeném $n-$rozměrném intervalu $\Omega \subseteq \mathbb{R}^n$ lze použít **metodu Monte Carlo**. Zatímco jiné numerické metody vyhodnocují integrand v pravidelné ekvidistantní mřížce, Monte Carlo metoda volí náhodné body z prostoru. Díky tomuto uvolnění tak poskytuje lepší výsledky v prostorech s vyšší dimenzí než klasické kvadraturní metody.

Výpočet určitého integrálu I (viz vztah 2.61) reálné funkce n proměnných f na intervalu $\Omega = \langle a_1, b_1\rangle \times \langle a_2, b_2\rangle \times \cdots \times \langle a_n, b_n\rangle$ metodou Monte Carlo je zaveden v [2, 5, 25] následovně:

$$I = \int_{a_n}^{b_n} \cdots \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(x_1, x_2, \ldots, x_n)\,\mathrm{d}x_1\,\mathrm{d}x_2 \ldots\,\mathrm{d}x_n. \tag{2.61}$$

Pro střední hodnotu této funkce na daném $n-$rozměrném intervalu platí:

$$\overline{f}(x_1, x_2, \ldots, x_n) = \frac{I}{(b_1 - a_1)(b_2 - a_2)\ldots(b_n - a_n)} = \frac{I}{|\Omega|}, \tag{2.62}$$

kde $|\Omega|$ vyjadřuje v objem dané oblasti, nebo-li platí:

$$I = |\Omega|\overline{f}(x_1, x_2, \ldots, x_n). \tag{2.63}$$

Geometrická představa pro reálnou funkci $f(x)$ jedné proměnné x je následující. Chceme spočítat její určitý integrál na intervalu $\langle a, b\rangle$:

$$I = \int_a^b f(x)\,\mathrm{d}x, \tag{2.64}$$

tj. vypočítat obsah plochy pod křivkou funkce $f(x)$ na daném intervalu. Pak existuje obdélník se stejným obsahem, který má stejnou šířku jako daný interval. Výška takového obdélníka je právě $\overline{f}(x)$, tedy

$$I = (b-a)\overline{f}(x). \tag{2.65}$$

Integrace Monte Carlo metodou vychází z potřeby určení střední hodnoty funkce f:

$$\overline{f}(x_1, x_2, \ldots, x_n) \approx \frac{1}{N}\sum_{i=1}^{N} f(\xi_1, \xi_2, \ldots, \xi_n), \tag{2.66}$$

kde N je zvolený počet bodů a $\xi_i = a_i + \gamma_i(b_i - a_i)$ je náhodně zvolené číslo v intervalu $\langle a_i, b_i\rangle$, přičemž γ_i je náhodné číslo z rovnoměrného rozdělení v intervalu $\langle 0, 1\rangle$. Střední hodnota funkce f na daném $n-$rozměrném intervalu Ω je aritmetickým průměrem funkčních hodnot funkce f z N náhodně zvolených bodů v tomto intervalu. Určitý integrál libovolné reálné funkce n proměnných f vypočtený metodou Monte Carlo je tvaru:

$$\begin{aligned} I &= |\Omega|\overline{f}(x_1, x_2, \ldots, x_n) \\ &= |\Omega|\frac{1}{N}\sum_{i=1}^{N} f(\xi_1, \xi_2, \ldots, \xi_n) \\ I &= \frac{(b_1 - a_1)\cdot \ldots \cdot (b_n - a_n)}{N}\sum_{i=1}^{N} f(\xi_1, \xi_2, \ldots, \xi_n), \end{aligned} \tag{2.67}$$

kde význam ξ_i byl popsán výše.

Chyba σ při určování střední hodnoty $\overline{f}(x_1, x_2, \ldots, x_n)$ funkce f je závislá na počtu N náhodně zvolených bodů v intervalu Ω:

$$\sigma_N \sim \frac{1}{\sqrt{N}}, \tag{2.68}$$

což v důsledku znamená, že pro snížení chyby výsledku $10\times$ je nutné použít $100\times$ více náhodně zvolených bodů. Tato metoda je proto klíčová při výpočtu určitých integrálů funkcí více proměnných, neboť stačí pouze vyčíslovat funkční hodnoty integrandu v N náhodných bodech.

Kapitola 3

Numerická integrace v prostoru obecné dimenze

V předchozí kapitole byly připomenuty některé numerické metody pro řešení určitých integrálů, a to jak funkcí jedné proměnné, tak funkcí n proměnných. Tyto metody, jako všechny numerické aproximační metody, mají specifickou přesnost (resp. odchylku numerického od přesného analytického řešení). V této kapitole bude navržena a podrobně popsána nová numerická metoda, jež si klade za cíl být nejen velmi rychlá, ale také přesná.

Hlavní motivací pro návrh této nové numerické metody je využití již existující velmi spolehlivé numerické metody pro řešení obyčejných diferenciálních rovnic – Taylorova řada (resp. polynom) – a její úspěšná implementace v programu `TKSL` (viz [17, 19]).

3.1 Numerické řešení integrálu funkcí dvou proměnných

Při zavádění této nové numerické metody nejprve tuto metodu popíšeme na nejsnazším případu vícenásobných integrálů – tedy na určitém integrálu funkcí dvou proměnných. Po zavedení numerické metody na tento případ, ji rozšíříme a zobecníme na určité integrály funkcí n proměnných.

V každé z následujících sekcí je podrobně popsán jeden krok této numerické metody. Kromě popisu daného kroku budou podrobně popsány i další numerické metody a postupy, které jsou při řešení daného kroku vyžadovány. Kroky jsou uváděny v takovém pořadí, v jakém je nutné je vykonávat pro získání správných výsledků. Celý postup numerické metody lze shrnout v těchto krocích:

1. normalizace integrálu,
2. vzorkování integračního intervalu,
3. numerické řešení obyčejných diferenciální rovnic s počáteční podmínkou,
4. numerický výpočet derivací funkce v daném bodě,
5. vyjádření určitého integrálu na daném intervalu.

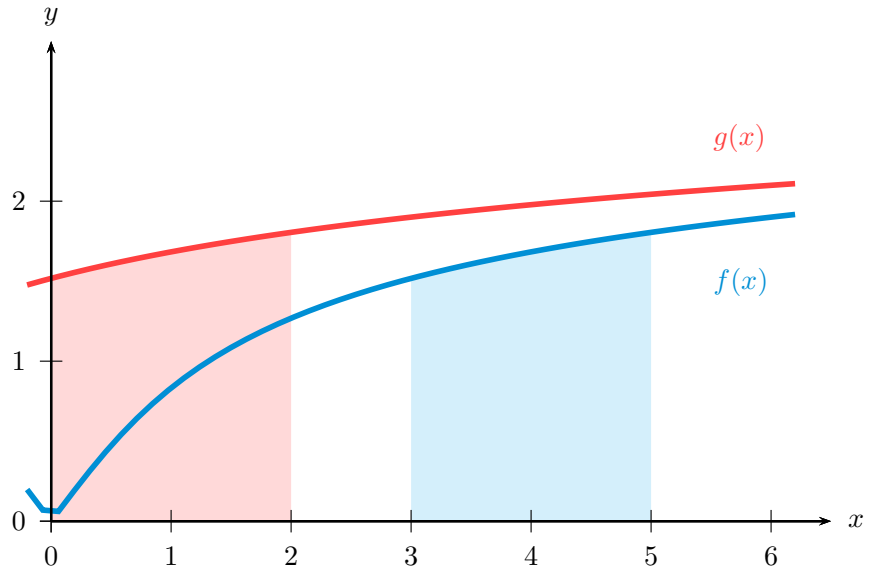
3.1.1 Normalizace integrálu

První úpravou zkoumaného integrálu je kontrola a případná záměna horní a dolní integrační meze pro každou integrační proměnnou i, pro kterou platí $a_i > b_i$. Pro každou integrační

proměnnou nastavíme výchozí hodnotu znaménka na $s_i = 1$, dojde-li k záměně integračních mezí, nastaví se tato hodnota na $s_i = -1$. Všechny tyto hodnoty znamének jsou uloženy ve vektoru s, jež se použije v závěrečné fázi celé metody.

$$\int_{a_2}^{b_2}\int_{a_1}^{b_1} f(x,y)\,\mathrm{d}x\,\mathrm{d}y = \begin{cases} \int_{a_2}^{b_2}\int_{a_1}^{b_1} f(x,y)\,\mathrm{d}x\,\mathrm{d}y, & a_1 \le b_1 \wedge a_2 \le b_2 \\ -\int_{a_2}^{b_2}\int_{b_1}^{a_1} f(x,y)\,\mathrm{d}x\,\mathrm{d}y, & a_1 > b_1 \wedge a_2 \le b_2 \\ -\int_{b_2}^{a_2}\int_{a_1}^{b_1} f(x,y)\,\mathrm{d}x\,\mathrm{d}y, & a_1 \le b_1 \wedge a_2 > b_2 \\ \int_{b_2}^{a_2}\int_{b_1}^{a_1} f(x,y)\,\mathrm{d}x\,\mathrm{d}y, & a_1 > b_1 \wedge a_2 > b_2 \end{cases} \tag{3.1}$$

Dále je provedena **normalizace integrálu**, což je podle [13] **translační** transformace integrálu. Znamená to, že řešený problém se přesune do počátku souřadného systému (jak integrační intervaly, tak samotný integrand). V konečném důsledku pak řešíme jiný problém, jehož výsledek je však totožný s výsledkem původního problému. Tato translace je znázorněna na obrázku 3.1 pro funkci $\int_3^5 \sqrt{\ln(x^2+1)}\,\mathrm{d}x$.



Obrázek 3.1: Integrál $\int_3^5 \sqrt{\ln(x^2+1)}\,\mathrm{d}x$ (modře) a jeho normalizovaná forma $\int_0^2 \sqrt{\ln((x+3)^2+1)}\,\mathrm{d}x$ (červeně) (vlastní tvorba).

Pro obecnou spojitou funkci dvou proměnných $f(x,y)$ je cílem spočítat integrál na 2−rozměrném intervalu $\langle a_1, b_1\rangle \times \langle a_2, b_2\rangle$, a tak pro integrál

$$\int_{a_2}^{b_2}\int_{a_1}^{b_1} f(x,y)\,\mathrm{d}x\,\mathrm{d}y, \tag{3.2}$$

zavedeme substituci $g(x, y) = f(x + a_1, y + a_2)$ a můžeme tedy psát:

$$\int_0^{b_2-a_2} \int_0^{b_1-a_1} g(x, y) \, \mathrm{d}x \, \mathrm{d}y. \tag{3.3}$$

Bez újmy na obecnosti můžeme každý určitý integrál takto transformovat (resp. *normalizovat*), a proto v dalším textu budeme uvažovat jen určité integrály tvaru:

$$\int_0^{b_y} \int_0^{b_x} f(x, y) \, \mathrm{d}x \, \mathrm{d}y, \tag{3.4}$$

kde $b_x = b_1 - a_1$ a $b_y = b_2 - a_2$ a funkce $f(x, y)$ je nahrazením původní funkce respektující translaci, jak již bylo uvedeno výše.

Příklad 1. Demonstrujme výše popisovanou transformaci na konkrétním příkladu. Uvažujme funkci $f(x, y) = e^{x^y}$ a oblast integrace $\langle 1, 2\rangle \times \langle 1, 2\rangle$, cílem je tedy určit integrál této funkce nad danou oblastí:

$$I \; = \int_1^2 \int_1^2 e^{x^y} \, \mathrm{d}x \, \mathrm{d}y. \tag{3.5}$$

Řešení. Normalizace tohoto určitého integrálu pak bude následující:

$$\begin{aligned} I \; &= \int_1^2 \int_1^2 e^{x^y} \, \mathrm{d}x \, \mathrm{d}y \\ &= \int_0^{2-1} \int_0^{2-1} e^{(x+1)^{(y+1)}} \, \mathrm{d}x \, \mathrm{d}y \\ &= \int_0^1 \int_0^1 e^{(x+1)^{(y+1)}} \, \mathrm{d}x \, \mathrm{d}y \end{aligned} \tag{3.6}$$

Výsledky původního a normalizovaného integrálu jsou skutečně ekvivalentní, což je možné ověřit vyjádřením těchto integrálů:

$$\begin{aligned} I_{original} &= \int_1^2 \int_1^2 e^{x^y} \, \mathrm{d}x \, \mathrm{d}y \approx 8.15884 \\ I_{normalized} &= \int_0^1 \int_0^1 e^{(x+1)^{(y+1)}} \, \mathrm{d}x \, \mathrm{d}y \approx 8.15884 \end{aligned} \tag{3.7}$$

3.1.2 Vzorkování integračního intervalu

Stejně jako u všech numerických metod je i pro nově navrhovanou numerickou metodu nutné zvolit dělení integračního intervalu pro každou integrační proměnnou zvlášť. Předpokládejme proto, že integrační interval $\langle 0, b_{x_i}\rangle$ pro proměnnou x_i je rozdělen dělicími body $D_{x_i} = \{a_0, a_1, \ldots, a_n\}$, kde $a_0 = 0$ a $a_n = b_{x_i}$, na dělicí intervaly $\langle a_0, a_1\rangle, \langle a_1, a_2\rangle, \ldots, \langle a_{n-1}, a_n\rangle$ tak, že vzdálenost každých dvou po sobě jdoucích dělicích bodů je vždy stejná. Tedy norma tohoto dělení D_{x_i} je:

$$\nu(D_{x_i}) = a_1 - a_0 = a_2 - a_1 = \cdots = a_n - a_{n-1} = \frac{a_n - a_0}{n} = \frac{b_{x_i}}{n} = h_{x_i}. \tag{3.8}$$

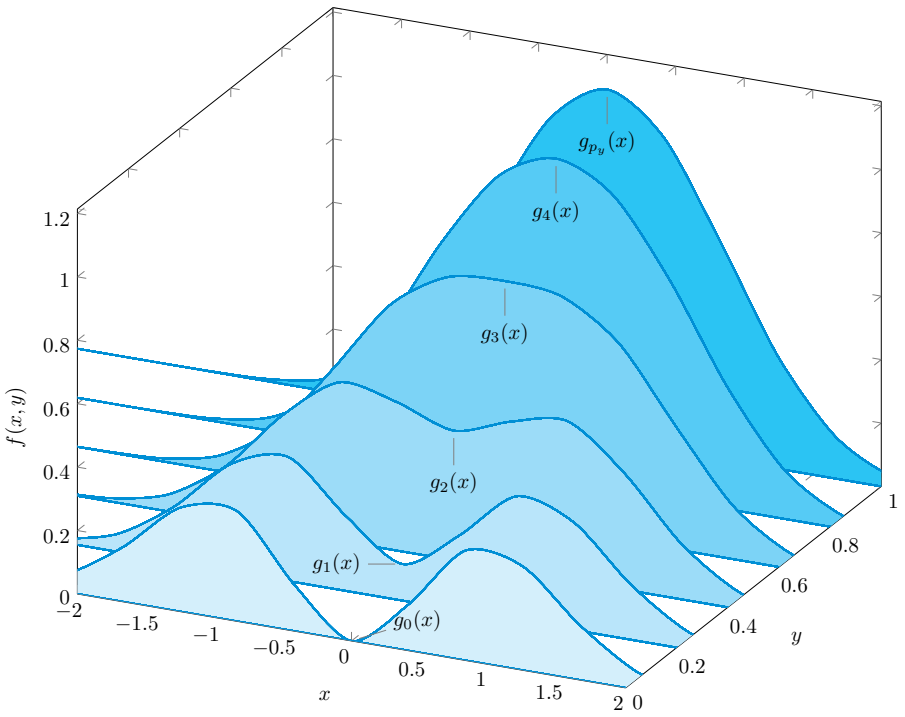
Pro každou integrační proměnnou tak tedy určíme integrační krok (resp. normu dělení integračního intervalu pro danou integrační proměnnou). Pro případ dvojného určitého integrálu můžeme uvažovat zápis vycházející ze zaužívaného značení:

- pro integrační proměnnou x zvolme integrační krok h_x, tedy počet dělicích bodů integračního intervalu $\langle 0, b_x \rangle$ je $p_x = \frac{b_x}{h_x}$,
- pro integrační proměnnou y zvolme integrační krok h_y, tedy počet dělicích bodů integračního intervalu $\langle 0, b_y \rangle$ je $p_y = \frac{b_y}{h_y}$.

Dále je nutné zvolit jednu integrační proměnnou, jež bude fixní (resp. časová proměnná v následujících krocích), a provést **vzorkování** podle dělení integračního intervalu druhé proměnné. Jako fixní proměnnou zvolme např. integrační proměnnou x. **Vzorkováním** podle integrační proměnné y rozumíme vytvoření systému funkcí takových, že pro každý dělicí bod $a_j = jh_y$ integračního intervalu $\langle 0, b_y \rangle$ je vytvořena nová funkce, kde všechny výskyty proměnné y jsou nahrazeny hodnotou dělicího bodu a_j:

$$\begin{aligned} g(x, 0) &= f(x, 0) \\ g(x, h_y) &= f(x, h_y) \\ &\dots \\ g(x, jh_y) &= f(x, jh_y) \\ &\dots \\ g(x, p_y h_y) &= f(x, p_y h_y) \end{aligned} \tag{3.9}$$

Jednotlivé funkce z tohoto systému bude značit s ohledem na dělicí bod a_j jako $g_j(x) = g(x, jh_y)$. Je tedy zřejmé, že vzorkováním podle jedné integrační proměnné se problém výpočtu určitého integrálu redukoval na snazší problém určení integrálu funkce jedné proměnné.



Obrázek 3.2: Funkce $f(x, y) = (x^2 + 3y^2)\mathrm{e}^{(-x^2-y^2)}$ na oblasti $\langle -2, 2 \rangle \times \langle 0, 1 \rangle$ vzorkovaná podle y s krokem $h_y = 0.2$ (vlastní tvorba).

Pro takto navzorkovanou integrační oblast podle jedné integrační proměnné (např. y) je nutné vypočítat určité integrály funkcí jednotlivých vzorků:

$$\begin{aligned} I_0 &= \int_0^{b_x} g_0(x)\,\mathrm{d}x \\ &\vdots \\ I_j &= \int_0^{b_x} g_j(x)\,\mathrm{d}x \\ &\vdots \\ I_{p_y} &= \int_0^{b_x} g_{p_y}(x)\,\mathrm{d}x \end{aligned} \tag{3.10}$$

Každý určitý integrál z tohoto systému lze řešit analyticky, ale vzhledem k celkovému charakteru této metody použijeme jednu z numerických metod popsaných v kapitole 2.1.3 (viz [17]). Uvažujme proto převod každého určitého integrálu $I_j = \int_0^{b_x} g_j(x)\,\mathrm{d}x$ na obyčejnou diferenciální rovnici s počáteční podmínkou $I_j(0) = 0$ (jak bylo popsáno v 2.1.4) a fixní proměnnou x nahraďme časovou proměnnou t:

$$\begin{aligned} I_0'(t) &= g_0(t), \quad I_0(t_{min}) = 0, \quad t_{min} = 0, t_{max} = b_x \\ I_1'(t) &= g_1(t), \quad I_1(t_{min}) = 0, \quad t_{min} = 0, t_{max} = b_x \\ &\vdots \\ I_j'(t) &= g_j(t), \quad I_j(t_{min}) = 0, \quad t_{min} = 0, t_{max} = b_x \\ &\vdots \\ I_{p_y}'(t) &= g_{p_y}(t), \quad I_{p_y}(t_{min}) = 0, \quad t_{min} = 0, t_{max} = b_x \end{aligned} \tag{3.11}$$

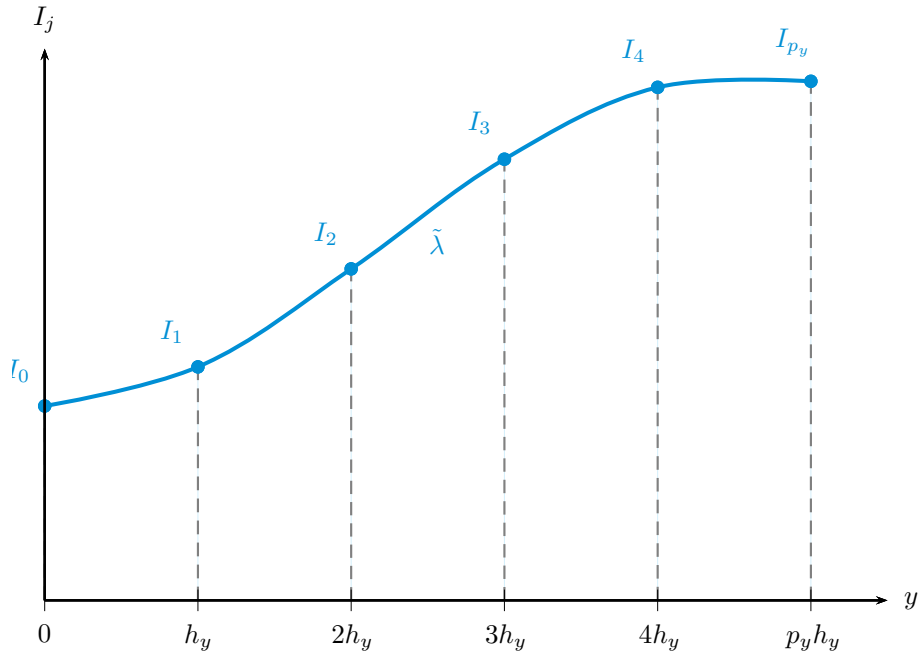
3.1.3 Numerické řešení diferenciální rovnice

Pro aproximaci výsledku každého $I_j(t)$ ze systému rovnic 3.11 můžeme využít Taylorův polynom, jak bylo popsáno v kapitole 2.1.4 (zde je nutno podotknout, že časová proměnná t odpovídá integrační proměnné x):

$$\begin{aligned} I_j(h_x) &= \sum_{m=0}^{n} \frac{h_x^m}{m!} I_j^{(m)}(0) \\ I_j(2h_x) &= \sum_{m=0}^{n} \frac{h_x^m}{m!} I_j^{(m)}(h_x) \\ &\vdots \\ I_j(kh_x) &= \sum_{m=0}^{n} \frac{h_x^m}{m!} I_j^{(m)}((k-1)h_x) \\ &\vdots \\ I_j(p_x h_x) &= \sum_{m=0}^{n} \frac{h_x^m}{m!} I_j^{(m)}((p_x-1)h_x), \end{aligned} \tag{3.12}$$

kde platí $\forall j \in \{0, \ldots, p_y\} : I_j(0) = 0$ (neboť plocha pod libovolnou křivkou v počátečním bodě numerické metody je 0) a kde n je nejvyšší řád derivace $I_j(t)$. Způsob vyjádření jednotlivých derivací libovolné funkce $I_j(t)$ zde nebude diskutován, neboť tato problematika zasahuje do jiné oblasti, než které se tato práce věnuje. Detailní rozbor řešení výše uvedeného systému diferenciálních rovnic 3.11 Taylorovým polynomem je popsán v [9, 17].

Takto vypočtené hodnoty $I_j(p_x h_x)$ pro $j = 0, \ldots, p_y$ zcela jistě interpolují *nějakou* funkci λ (viz obrázek 3.3), která aproximuje průběh určitého integrálu vzorkovaných funkcí jako funkci proměnné x (v kontextu předchozího textu se jedná o fixní časovou proměnnou). Pro výpočet určitého integrálu funkce dvou proměnných, jež je vstupem této metody, by nyní stačilo sestavit z hodnot $I_j(p_x h_x)$ Lagrangeův interpolační polynom (jeho sestavení viz kapitola 2.1.3) $L(x)$ a tento polynom integrovat na integračním intervalu proměnné x. Avšak sestavením interpolačního polynomu bychom do úlohy přinesli značné nepřesnosti a chyby, a proto jeho sestavení a následné integrování není v tomto případě vhodné.



Obrázek 3.3: Funkce $\tilde{\lambda}$ interpolující integrály $I_j(p_x h_x)$ jednotlivých vzorků $g_j(x)$ (vlastní tvorba).

Řešením může být opětovné využití Taylorova polynomu, podobně jako tomu bylo při vyhodnocování integrálů jednotlivých vzorků I_j. Tímto se vyhneme sestavování jakéhokoliv interpolačního či aproximačního polynomu, neboť k výpočtu budeme využívat pouze funkční hodnoty pomyslné funkce λ a jejich derivace.

Uvažujme tedy funkci λ zadanou diskrétními hodnotami následovně:

$$\begin{aligned}
\lambda(0) &= I_0 \\
\lambda(h_y) &= I_1 \\
&\vdots \\
\lambda(jh_y) &= I_j \\
&\vdots \\
\lambda(p_y h_y) &= I_{p_y}
\end{aligned} \tag{3.13}$$

Je tedy zřejmé, že s využitím této funkce λ (resp. její spojité varianty $\tilde{\lambda}$) lze původní dvojnásobný integrál převést na jednoduchý:

$$\int_0^{b_y} \int_0^{b_x} f(x,y)\,\mathrm{d}x\,\mathrm{d}y \approx \int_0^{b_y} \tilde{\lambda}(y)\,\mathrm{d}y. \tag{3.14}$$

Označme tento integrál spojité varianty funkce $\tilde{\lambda}$ jako:

$$J(y) = \int_0^{b_y} \tilde{\lambda}(y)\,\mathrm{d}y. \tag{3.15}$$

Tento určitý integrál můžeme opět vypočítat numericky převodem na diferenciální rovnici s adekvátní počáteční podmínkou, jak tomu bylo výpočtu určitých integrálů jednotlivých vzorkovaných funkcí:

$$J'(t) = \tilde{\lambda}(t), \quad J(t_{min}) = 0, \quad t_{min} = 0, t_{max} = b_y. \tag{3.16}$$

A tuto diferenciální rovnici můžeme řešit převodem na Taylorův polynom:

$$\begin{aligned}
J(h_y) &= \sum_{m=0}^{n} \frac{h_y^m}{m!} J^{(m)}(0) \\
J(2h_y) &= \sum_{m=0}^{n} \frac{h_y^m}{m!} J^{(m)}(h_y) \\
&\vdots \\
J(lh_y) &= \sum_{m=0}^{n} \frac{h_y^m}{m!} J^{(m)}((l-1)h_y) \\
&\vdots \\
J(p_y h_y) &= \sum_{m=0}^{n} \frac{h_y^m}{m!} J^{(m)}((p_y-1)h_y)
\end{aligned} \tag{3.17}$$

Dále však ze vztahu 3.16 vyplývá, že

$$\frac{h_y^m}{m!} J^{(m)}(t) = \frac{h_y^m}{m!} \tilde{\lambda}^{(m-1)}(t), \tag{3.18}$$

a tak lze systém 3.17 (uvažujeme-li diskrétní variantu funkce λ) přepsat ve tvaru:

$$\begin{aligned}
J(h_y) &= J(0) + \sum_{m=1}^{n} \frac{h_y^m}{m!}\lambda^{(m-1)}(0) \\
J(2h_y) &= J(h_y) + \sum_{m=1}^{n} \frac{h_y^m}{m!}\lambda^{(m-1)}(h_y) \\
&\vdots \\
J(lh_y) &= J((l-1)h_y) + \sum_{m=1}^{n} \frac{h_y^m}{m!}\lambda^{(m-1)}((l-1)h_y) \\
&\vdots \\
J(p_y h_y) &= J((p_y-1)h_y) + \sum_{m=1}^{n} \frac{h_y^m}{m!}\lambda^{(m-1)}((p_y-1)h_y)
\end{aligned} \tag{3.19}$$

3.1.4 Numerický výpočet derivací

V systému rovnic 3.19 jsou v Taylorově polynomu na pravé straně zapotřebí jednotlivé derivace funkce $\lambda(t)$ v bodech jh_y, pro $j = 0, \ldots, p_y - 1$, které jsou však v tuto chvíli neznámé. Proto je nutné tyto derivace vyjádřit např. za pomoci numerické metody, která opět vychází z Taylorova rozvoje. Z charakteru tohoto problému vyplývá, že je možné určit veškeré derivace funkce λ v daném bodě jh_y pouze na základě funkčních hodnot a hodnot derivací okolních bodů ih_y, kde $i \neq j$. K tomu je možné použít *dopřednou, zpětnou* nebo *kombinovanou diferenci* (viz [18]).

Uvažujme tedy obecně systém rovnic:

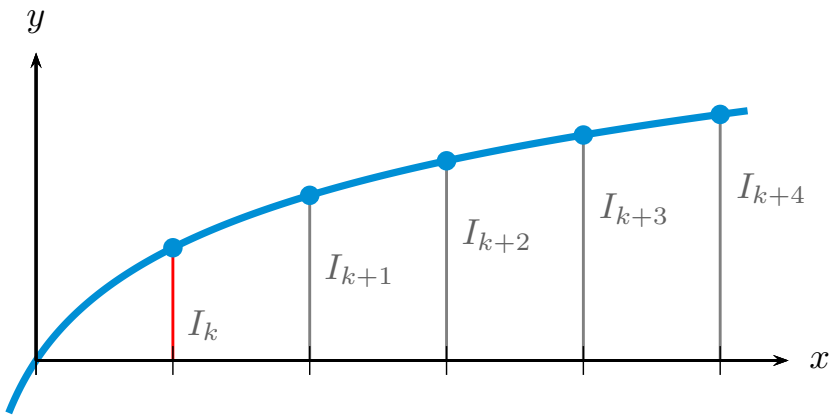
$$\begin{aligned}
\lambda(h_y) &= \sum_{m=0}^{n} 1^m \frac{h_y^m}{m!}\lambda^{(m)}(0) \\
\lambda(2h_y) &= \sum_{m=0}^{n} 2^m \frac{h_y^m}{m!}\lambda^{(m)}(0) \\
&\vdots \\
\lambda(lh_y) &= \sum_{m=0}^{n} l^m \frac{h_y^m}{m!}\lambda^{(m)}(0) \\
&\vdots \\
\lambda(nh_y) &= \sum_{m=0}^{n} n^m \frac{h_y^m}{m!}\lambda^{(m)}(0),
\end{aligned} \tag{3.20}$$

kde však platí $\lambda(jh_y) = I_j$, a proto lze systém rovnic 3.20 přepsat jako:

$$\begin{aligned} I_1 &= \sum_{m=0}^{n} 1^m \frac{h_y^m}{m!} I_0^{(m)} \\ I_2 &= \sum_{m=0}^{n} 2^m \frac{h_y^m}{m!} I_0^{(m)} \\ &\vdots \\ I_l &= \sum_{m=0}^{n} l^m \frac{h_y^m}{m!} I_0^{(m)} \\ &\vdots \\ I_n &= \sum_{m=0}^{n} n^m \frac{h_y^m}{m!} I_0^{(m)} \end{aligned} \tag{3.21}$$

Dopředná diference

Systém rovnic 3.21 využívá k vyjádření n **následujících** funkčních hodnot $I_1, \ldots, I_n$ pouze jednu funkční hodnotu I_0 a její derivace $I_0^{(m)}$ řádu nejvýše n pro $m = 1, \ldots, n$. Jedná se tedy o výpočet **dopředné diference**. Tuto diferenci je vhodné použít pro výpočet derivací minimálního bodu I_0 z bodů I_j pro $j = 0, \ldots, p_y$.



Obrázek 3.4: Dopředná diference (vlastní tvorba podle [18]).

Upravme tedy systém 3.21 tak, ať všechny známé hodnoty jsou odděleny od neznámých, jež je cílem vyjádřit:

$$\begin{aligned} I_1 - I_0 &= \sum_{m=1}^{n} 1^m \frac{h_y^m}{m!} I_0^{(m)} \\ I_2 - I_0 &= \sum_{m=1}^{n} 2^m \frac{h_y^m}{m!} I_0^{(m)} \\ &\vdots \\ I_l - I_0 &= \sum_{m=1}^{n} l^m \frac{h_y^m}{m!} I_0^{(m)} \\ &\vdots \\ I_n - I_0 &= \sum_{m=1}^{n} n^m \frac{h_y^m}{m!} I_0^{(m)} \end{aligned} \tag{3.22}$$

Konečně tento systém lze přepsat do maticového zápisu soustavy rovnic, jejíž řešením a následným vyjádřením získáme jednotlivé derivace $I_0^{(m)}$ pro $m = 1, \ldots, n$:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}, \quad \text{tedy} \quad \mathbf{x} = \mathbf{A}^{-1}\mathbf{b}, \tag{3.23}$$

kde jednotlivé matice $\mathbf{A}, \mathbf{x}$ a $\mathbf{b}$ mají tento význam:

$$\mathbf{A} = \begin{pmatrix} 1 & 1^2 & \ldots & 1^n \\ 2 & 2^2 & \ldots & 2^n \\ & \vdots & & \\ l & l^2 & \ldots & l^n \\ & \vdots & & \\ n & n^2 & \ldots & n^n \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} \frac{h_y^1}{1!} I_0^{(1)} \\ \frac{h_y^2}{2!} I_0^{(2)} \\ \vdots \\ \frac{h_y^l}{l!} I_0^{(l)} \\ \vdots \\ \frac{h_y^n}{n!} I_0^{(n)} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} I_1 - I_0 \\ I_2 - I_0 \\ \vdots \\ I_l - I_0 \\ \vdots \\ I_n - I_0 \end{pmatrix} \tag{3.24}$$

Při řešení této soustavy však narážíme na další problém, který vyvstává v případu, kdy počet využívaných funkčních hodnot, resp. počet vyjadřovaných derivací, je $n \gg 0$ a matice $\mathbf{A}$ je tak čtvercová singulární. Tehdy zřejmě není možné a efektivní analytický výpočet inverzní matice $\mathbf{A}^{-1}$ k matici $\mathbf{A}$, proto vezněme v úvahu **pseudoinverzní matici**, jež je vyjádřena numerickou metodou.

Moore-Penrose pseudoinverzní matice Moore-Penrose pseudoinverzní matice $\mathbf{A}^+$ (viz [4, 31]) nebo též zobecněná inverze se používá ke zobecnění pojmu inverzní matice v případech, kdy matice $\mathbf{A}$ je čtvercová singulární, nebo obdélníková, tedy v případech, kdy klasická inverze neexistuje.

Definice 3.1 (Moore-Penrose pseudoinverzní matice)**.** Moore-Penrose pseudoinverzní matice $\mathbf{A}^+$ matice $\mathbf{A}$ je matice, která je jednoznačným řešením soustavy čtyř (nelineárních) rovnic, tzv. Moore-Penroseových podmínek (kde $\mathbf{A}^H$ znamená *konjugovaná transpozice* matice $\mathbf{A}$):

1. $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$,

2. $\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+$,

3. $(\mathbf{A}\mathbf{A}^+)^H = \mathbf{A}\mathbf{A}^+$,

4. $(\mathbf{A}^+\mathbf{A})^H = \mathbf{A}^+\mathbf{A}$.

Příklad 2. Vypočtěte pseudoinverzní matici $\mathbf{A}^+$ k zadané matici $\mathbf{A}$:

$$\mathbf{A} = \begin{pmatrix} 7 & 3 & 1 \\ 2 & 9 & 0 \\ 1 & 3 & 5 \\ 6 & 7 & 6 \end{pmatrix}. \tag{3.25}$$

Řešení. Nejprve vyjádříme singulární rozklad $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\star$ a poté pomocí jednotlivých složek vyjádříme Moore-Penrose pseudoinverzní matici $\mathbf{A}^+$.

$$\begin{aligned} \mathbf{U} &= \begin{pmatrix} -0.4127 & 0.3202 & 0.7532 & -0.3998 \\ -0.5009 & -0.8558 & 0.0217 & -0.1274 \\ -0.3057 & 0.2617 & -0.6308 & -0.6634 \\ -0.6967 & 0.3108 & -0.1850 & 0.6195 \end{pmatrix}, \\ \mathbf{\Sigma} &= \begin{pmatrix} 15.5177 & 0 & 0 \\ 0 & 5.7927 & 0 \\ 0 & 0 & 5.0641 \\ 0 & 0 & 0 \end{pmatrix}, \\ \mathbf{V} &= \begin{pmatrix} -0.5398 & 0.4585 & 0.7060 \\ -0.7437 & -0.6527 & -0.1447 \\ -0.3945 & 0.6031 & -0.6933 \end{pmatrix}. \end{aligned} \tag{3.26}$$

Pseudoinverzní matice $\mathbf{A}^+$ se vyjádří jako $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^\star$ a tedy po částech vyjádříme:

$$\mathbf{\Sigma}^+ = \begin{pmatrix} 0.0644 & 0 & 0 & 0 \\ 0 & 0.1726 & 0 & 0 \\ 0 & 0 & 0.1975 & 0 \end{pmatrix},$$

$$\begin{aligned} \mathbf{V}\mathbf{\Sigma}^+ &= \begin{pmatrix} -0.5398 & 0.4585 & 0.7060 \\ -0.7437 & -0.6527 & -0.1447 \\ -0.3945 & 0.6031 & -0.6933 \end{pmatrix} \cdot \begin{pmatrix} 0.0644 & 0 & 0 & 0 \\ 0 & 0.1726 & 0 & 0 \\ 0 & 0 & 0.1975 & 0 \end{pmatrix} \\ &= \begin{pmatrix} -0.0348 & 0.0792 & 0.1394 & 0 \\ -0.0479 & -0.1127 & -0.0286 & 0 \\ -0.0254 & 0.1041 & -0.1369 & 0 \end{pmatrix} \end{aligned} \tag{3.27}$$

Nakonec tedy:

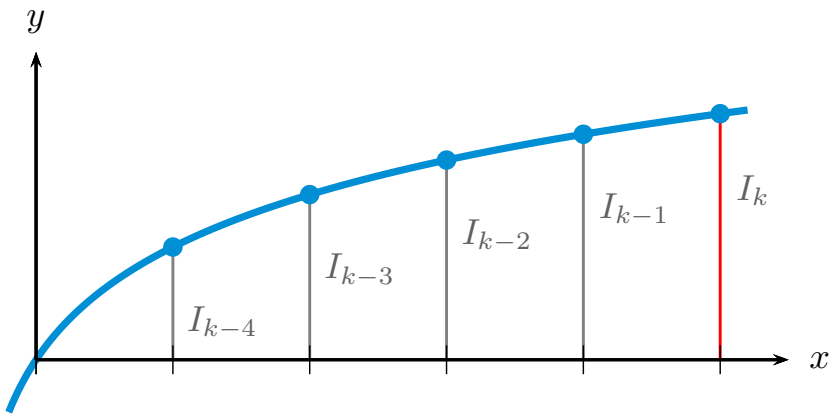
$$\begin{aligned} \mathbf{A}^+ &= (\mathbf{V}\mathbf{\Sigma}^+)\mathbf{U}^\star = \\ &= \begin{pmatrix} -0.0348 & 0.0792 & 0.1394 & 0 \\ -0.0479 & -0.1127 & -0.0286 & 0 \\ -0.0254 & 0.1041 & -0.1369 & 0 \end{pmatrix} \cdot \begin{pmatrix} -0.4127 & -0.5009 & -0.3057 & -0.6967 \\ 0.3202 & -0.8558 & 0.2617 & 0.3108 \\ 0.7532 & 0.0217 & -0.6308 & -0.1850 \\ -0.3998 & -0.1274 & -0.6634 & 0.6195 \end{pmatrix} \\ &= \begin{pmatrix} 0.1447 & -0.0473 & -0.0566 & 0.0230 \\ -0.0378 & 0.1198 & 0.0032 & 0.0036 \\ -0.0593 & -0.0793 & 0.1214 & 0.0754 \end{pmatrix}. \end{aligned} \tag{3.28}$$

Řešením systému rovnic v maticovém zápisu získáme jednotlivé členy $\frac{h_y^l}{l!} I_0^{(l)}$, z nichž pak vyjádříme derivace $I_0^{(l)}$ pro $l = 1, \ldots, n$.

V předchozím výkladu jsme všechny hodnoty I_l vyjadřovali pomocí I_0, avšak platí, že tuto metodu lze obecně použít pro vyjádření n následujících hodnot I_l pomocí hodnoty I_j a tedy $l = j + 1, \ldots, j + n$.

Zpětná diference

Obdobně jako u dopředné diference, tak se u zpětné diference pro vyjádření derivací hodnoty I_l použijí pouze předcházející hodnoty I_j, $j = l - n, l - n + 1, \ldots, l - 1$.



Obrázek 3.5: Zpětná diference (vlastní tvorba podle [18]).

Takže systém rovnic 3.21 lze vyjádřit jako:

$$\begin{aligned}
I_{i-1} - I_i &= \sum_{m=1}^{n} (-1)^m \frac{h_y^m}{m!} I_i^{(m)} \\
I_{i-2} - I_i &= \sum_{m=1}^{n} (-2)^m \frac{h_y^m}{m!} I_i^{(m)} \\
&\vdots \\
I_{i-l} - I_i &= \sum_{m=1}^{n} (-l)^m \frac{h_y^m}{m!} I_i^{(m)} \\
&\vdots \\
I_{i-n} - I_i &= \sum_{m=1}^{n} (-n)^m \frac{h_y^m}{m!} I_i^{(m)}
\end{aligned} \tag{3.29}$$

Což lze přepsat do maticového zápisu jako:

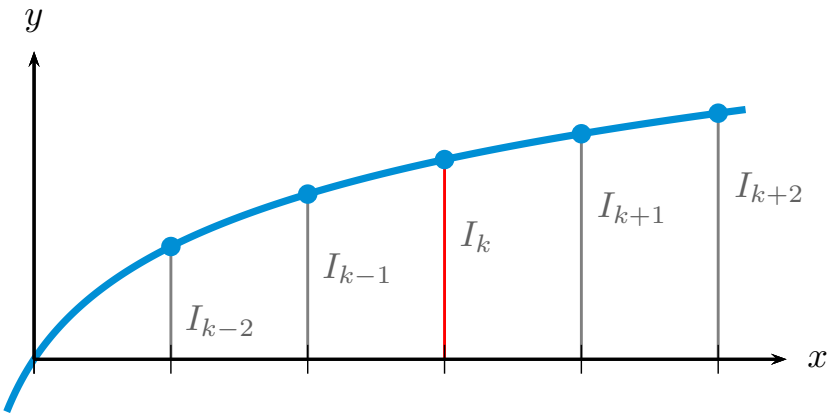
$$\begin{pmatrix} -1 & (-1)^2 & \dots & (-1)^n \\ -2 & (-2)^2 & \dots & (-2)^n \\ & \vdots & & \\ -l & (-l)^2 & \dots & (-l)^n \\ & \vdots & & \\ -n & (-n)^2 & \dots & (-n)^n \end{pmatrix} \cdot \begin{pmatrix} \frac{h_y^1}{1!} I_i^{(1)} \\ \frac{h_y^2}{2!} I_i^{(2)} \\ \vdots \\ \frac{h_y^l}{l!} I_i^{(l)} \\ \vdots \\ \frac{h_y^n}{n!} I_i^{(n)} \end{pmatrix} = \begin{pmatrix} I_{i-1} - I_i \\ I_{i-2} - I_i \\ \vdots \\ I_{i-l} - I_i \\ \vdots \\ I_{i-n} - I_i \end{pmatrix} \tag{3.30}$$

Řešení této soustavy rovnic a následné vyjádření jednotlivých derivací $I_i^{(l)}$ je stejné jako u dopředné diference, proto zde již nebude popisováno. Z charakteru zpětné diference je zřejmé, že je nejvhodnější pro vyjadřování hodnot pomocí maximální krajní hodnoty, ale pochopitelně lze použít i pro libovolnou vnitřní hodnotu.

Kombinovaná diference

Předchozí dvě diference je praktické používat v případech krajních bodů integračního intervalu, kdy při *dopředné diferenci* vyjádříme derivace I_0 pomocí n následujících hodnot; použitím *zpětné diference* je vhodné určit derivace I_{p_y} pomocí n předcházejících hodnot.

Konečně, chceme-li vyjádřit derivace nějaké hodnoty uvnitř integračního intervalu, tedy derivace I_l, $l = 1, \dots, p_y - 1$, je pro vyšší přesnost vhodné využít **kombinovanou diferenci**, kdy derivace jsou počítány na základě několika předcházejících a několika následujících hodnot.



Obrázek 3.6: Kombinovaná diference (vlastní tvorba podle [18]).

Systém rovnic tedy vypadá následovně:

$$\begin{aligned}
I_{i-p} - I_i &= \sum_{m=1}^{n} (-p)^m \frac{h_y^m}{m!} I_i^{(m)} \\
&\vdots \\
I_{i-1} - I_i &= \sum_{m=1}^{n} (-1)^m \frac{h_y^m}{m!} I_i^{(m)} \\
I_{i+1} - I_i &= \sum_{m=1}^{n} 1^m \frac{h_y^m}{m!} I_i^{(m)} \\
&\vdots \\
I_{i+q} - I_i &= \sum_{m=1}^{n} q^m \frac{h_y^m}{m!} I_i^{(m)}
\end{aligned} \tag{3.31}$$

a lze jej převést do maticového zápisu:

$$\begin{pmatrix}
-p & (-p)^2 & \dots & (-p)^n \\
 & \vdots & & \\
-1 & (-1)^2 & \dots & (-1)^n \\
1 & 1^2 & \dots & 1^n \\
 & \vdots & & \\
q & q^2 & \dots & q^n
\end{pmatrix} \cdot \begin{pmatrix}
\frac{h_y^1}{1!} I_i^{(1)} \\
\frac{h_y^2}{2!} I_i^{(2)} \\
\vdots \\
\frac{h_y^l}{l!} I_i^{(l)} \\
\vdots \\
\frac{h_y^n}{n!} I_i^{(n)}
\end{pmatrix} = \begin{pmatrix}
I_{i-p} - I_i \\
\vdots \\
I_{i-1} - I_i \\
I_{i+1} - I_i \\
\vdots \\
I_{i+q} - I_i
\end{pmatrix}$$

Využitím těchto tří různých diferencí (dopředná, zpětná a kombinovaná) můžeme pomocí n hodnot I_l, kde $l \in \{0, \dots, p_y\}$, vyjádřit n derivací každé hodnoty I_i, kde $i \neq l$.

3.1.5 Numerická integrace

Vypočtené derivace jednotlivých vzorků tak můžeme přímo dosadit do vztahu 3.19 (resp. 3.32):

$$\begin{aligned}
J(h_y) &= J(0) + \sum_{m=1}^{n} \frac{h_y^m}{m!} \lambda^{(m-1)}(0) \\
J(2h_y) &= J(h_y) + \sum_{m=1}^{n} \frac{h_y^m}{m!} \lambda^{(m-1)}(h_y) \\
&\vdots \\
J(lh_y) &= J((l-1)h_y) + \sum_{m=1}^{n} \frac{h_y^m}{m!} \lambda^{(m-1)}((l-1)h_y) \\
&\vdots \\
J(p_y h_y) &= J((p_y-1)h_y) + \sum_{m=1}^{n} \frac{h_y^m}{m!} \lambda^{(m-1)}((p_y-1)h_y)
\end{aligned} \tag{3.32}$$

Vyjádřením hodnoty $J(p_y h_y)$ tak získáme celkový dvojný integrál vstupní funkce $f(x, y)$ na integračním intervalu $\langle 0, b_x \rangle \times \langle 0, b_y \rangle$:

$$\int_0^{b_y} \int_0^{b_x} f(x, y) \, \mathrm{d}x \, \mathrm{d}y \approx J(p_y h_y). \tag{3.33}$$

Posledním krokem k získání správného výsledku je nastavení znaménka výsledku, neboť záměnou integračních mezí dochází k otočení znaménka, a proto je nutné tuto změnu reflektovat i na výsledném řešení:

$$\int_{a_2}^{b_1} \int_{a_1}^{b_1} f_{orig}(x, y) \, \mathrm{d}x \, \mathrm{d}y \approx J(p_y h_y) \prod_{i=1}^{n} s_i. \tag{3.34}$$

3.2 Zobecnění na integrál funkce více proměnných

V předchozí kapitole byla podrobně popsána a zavedena nová numerická metoda pro výpočet určitých integrálů funkcí dvou reálných proměnných. Tato metoda spočívá především ve vhodně opakované aplikaci Taylorova rozvoje, resp. polynomu. Výhoda tohoto přístupu spočívá zejména v použitých matematických operacích (základní operace jako je sčítání, odčítání, násobení, dělení a výpočet mocniny), které jsou výpočetně nenáročné. Další výhodou tohoto přístupu je možnost rozšíření domény řešitelných problémů na určité integrály funkcí n reálných proměnných.

V této kapitole budu popsán způsob, jakým je metoda rozšířena z případu určitého integrálu funkcí dvou proměnných na případ určitého integrálu funkcí n proměnných. Nebudou proto popisovány detaily řešení, neboť ty jsou stejné jako v předchozí kapitole, ale pouze odlišnosti a způsob zobecnění metody.

Princip zobecnění spočívá v postupném integrování podle jedné integrační proměnné a následné redukci problému o tuto integrační proměnnou, dokud není řešený problém zredukován pouze na jednu integrační proměnnou. Kroky, které se při řešení budou provádět, jsou obdobné s těmi, jež byly popisovány v předešlé kapitole. Pro účely zobecnění pak budou přidány podmínky pro integraci podle jedné integrační proměnné a následné redukce řešeného problému o jednu dimenzi. Výpočet tedy bude veden následovně:

1. normalizace integrálu,
2. vzorkování integračního intervalu,
3. numerické řešení obyčejných diferenciální rovnic s počáteční podmínkou,
4. seskupení numerických řešení podle dělicích bodů a následný
   (a) numerický výpočet derivací funkce v daném bodě,
   (b) vyjádření určitého integrálu na daném intervalu,
   (c) redukce řešeného problému o jednu dimenzi (resp. eliminace jedné integrační proměnné).

3.2.1 Normalizace integrálu

Stejně jako u případu určitého integrálu funkcí dvou proměnných i zde je v prvním kroku prováděna normalizace integrálu, což znamená jeho translace do počátku souřadného systému a adekvátní úprava integrandu. Uvažujme, že integrační meze jsou pro všechny integrační proměnné i ve vztahu $a_i \leq b_i$ a znaménko je nastaveno na $s_i = -1$, bylo-li nutné integrační meze prohodit, jinak $s_i = 1$. Obecně tedy můžeme uvažovat tuto rovnost:

$$\begin{aligned}\int_{a_n}^{b_n} \cdots \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(x_1, x_2, \ldots, x_n)\,\mathrm{d}x_1\,\mathrm{d}x_2 \cdots \mathrm{d}x_n = \\ \int_{0}^{b_{x_n}} \cdots \int_{0}^{b_{x_2}} \int_{0}^{b_{x_1}} g(x_1, x_2, \ldots, x_n)\,\mathrm{d}x_1\,\mathrm{d}x_2 \cdots \mathrm{d}x_n,\end{aligned} \tag{3.35}$$

kde $g(x_1, x_2, \ldots, x_n) = f(x_1 + a_1, x_2 + a_2, \ldots, x_n + a_n)$ a horní integrační meze $b_{x_i} = b_i - a_i$ pro $i = 1, \ldots, n$.

Jelikož platí Fubiniova věta 2.7, není nutné integrovat v pořadí $x_1, x_2, \ldots, x_n$, ale je možné uvažovat libovolnou permutaci $\pi : N \to N$ integračních proměnných, kde $N = \{1, 2, \ldots, n\}$. V následujícím textu bude vždy brána v úvahu libovolná permutace integračních proměnných ve tvaru:

$$\int_{0}^{b_{\pi(n)}} \cdots \int_{0}^{b_{\pi(2)}} \int_{0}^{b_{\pi(1)}} g(x_1, x_2, \ldots, x_n)\,\mathrm{d}x_{\pi(1)}\,\mathrm{d}x_{\pi(2)} \cdots \mathrm{d}x_{\pi(n)}. \tag{3.36}$$

3.2.2 Vzorkování integrační oblasti

Dalším nezbytným krokem k získání určitého integrálu funkce na dané integrační oblasti je sestavení sady vzorků této funkce. Pro každou integrační proměnnou vytvoříme na jejím integračním intervalu ekvidistantní síť dělicích bodů. Každá integrační proměnná x_i má tedy svůj integrační interval $\langle 0, b_{x_i} \rangle$ rozdělen $p_{x_i} = \frac{b_{x_i}}{h_{x_i}}$ dělicími body na dělení $D_{x_i} = \{jh_{x_i} | j = 0, 1, \ldots, p_{x_i}\}$, kde $h_{x_i} = \nu(D_{x_i})$ je integrační krok pro proměnnou x_i (resp. norma dělení D_{x_i}.

Dále je nutné zvolit jednu integrační proměnnou x_j, která bude v následujících výpočtech považována za fixní časovou proměnnou. Tato proměnná nebude vzorkována, ale vždy bude nahrazena časovou proměnnýchou t.

Výše uvedené vzorkování je prováděno vždy pro všechny integrační proměnné (až na zvolenou časovou proměnnou), proto pro integrovanou funkci je celkové dělení dáno kartézským součinem dělení jednotlivých integračních intervalů:

$$D = \prod_{i=1}^{n} D_{x_i}. \tag{3.37}$$

Pro každý vzorek $s \in D$ funkce $f(x_1, \ldots, x_n)$ sestavíme *vzorkovanou* funkci $g_s(x_j)$ takovou, že každý výskyt integrační proměnné x_i je ve vzorkované funkci g_s nahrazen hodnotou s_{x_i}. Celkový počet vzorkovaných funkcí je $N = |D|$:

$$N = \prod_{\substack{i=1 \\ i \neq j}}^{n} (p_{x_i} + 1). \tag{3.38}$$

3.2.3 Numerické řešení diferenciální rovnice

Pro všechny vzorkované funkce $g_s, s \in D$, sestavíme určitý integrál jedné proměnné x_j, jež je shodná s fixní časovou proměnnou t, na integračním intervalu této integrační proměnné $\langle 0, b_{x_j} \rangle$:

$$\forall s \ \in D. I_s = \int_0^{b_{x_j}} g_s(x_j) \, \mathrm{d}x_j. \tag{3.39}$$

Těchto N určitých integrálů je řešeno převodem na obyčejnou diferenciální rovnici s nulovou počáteční podmínkou, která je vyhodnocena pomocí Taylorova polynomu, jak již bylo popsáno v předchozí kapitole:

$$\forall s \in D. I'_s(t) = g_s(t), \quad I_s(t_{min}) = 0, \quad t_{min} = 0, t_{max} = b_{x_j}. \tag{3.40}$$

3.2.4 Postupná numerická integrace

Z vypočtených hodnot I_s určitých integrálů vzorkovaných funkcí g_s budou v dalších krocích počítány derivace v každém vzorku integrační oblasti. Dosazením těchto derivací jednotlivých vzorků do Taylorova polynomu budou počítány *plochy* ve směru každé integrační proměnné a po vyjádření této plochy bude problém určení integrálu redukován o dimenzi této integrační proměnné.

Nejprve je nutné vytvořit skupiny vzorků $s|_{x_i} \subset D$, jejichž hodnoty $I_u, u \ \in s|_{x_i}$, budou použity pro výpočet určitého integrálu pro proměnnou x_i. Pro tyto skupiny vzorků $s|_{x_i}$ platí, že jsou shodné ve všech hodnotách $x_j, j \in \{1, \ldots, n\} \backslash \{k\}$, a liší se pouze v hodnotách x_k, kde x_k jsou všechny dělicí body dělení D_{x_k} intervalu $\langle 0, b_{x_k} \rangle$. Z toho vyplývá, že skupiny vzorků jsou sestavovány podle jednotlivých dělicích bodů intervalu $\langle 0, b_{x_i} \rangle$. To si lze představit tak, že vyjadřujeme plochu řezu funkce f pod integrační proměnnou x_i ve směru proměnné x_k.

Numerický výpočet derivací

Z hodnot $I_u, u \ \in s|_{x_i}$, vyjádříme jednotlivé derivace $I_u^{(j)}, j = 1, \ldots, p_{x_k}$, proměnné x_i až do řádu maximálně p_{x_k}. Tyto derivace jsou vypočteny pomocí **dopředné**, **zpětné** či **kombinované** diference s krokem h_{x_i}, jež byla podrobně popsána v předchozí kapitole 3.1.4 a jež využívá Taylorův polynom.

Numerická integrace

Nyní již známe všechny derivace hodnot $I_u, u \ \in s|_{x_i}$, jež potřebujeme k dosazení do Taylorova polynomu pro vyjádření plochy aktuálně zkoumaného *řezu* pro konkrétní vzorek x_i. Učiníme-li tak podle postupu uvedeného v kapitole 3.1.5, získáme plochu *řezu* vedeného ve směru osy proměnné x_k s ohledem na vzorek proměnné x_i.

Dimenzionální redukce problému

Pokud se v předchozích krocích podařilo spočítat integrál pro konkrétní vzorek $s|_{x_i}$, je nutné, aby tento výsledek byl přidán do množiny zkoumaných vzorků a odstraněny vzorky, jež byly pro jeho výpočet použity. Do množiny vzorků je tedy přidán nový vzorek $s|_{x_i}{}'$ takový, jež má stejné hodnoty pro proměnné $x_j, j \in \{1, \ldots, n\} \setminus \{k\}$, a hodnota proměnné x_k je odstraněna. Velikost takového vzorku je o jedna menší než velikost jakéhokoliv vzorku $u \in s|_{x_i}$.

Jakmile jsou všechny vzorky integrační oblasti rozděleny do skupin podle jednotlivých hodnot vzorků integrační proměnné x_i a jsou pro ně vypočteny integrály, jak bylo popsáno výše, obsahuje množina vzorků pouze $s|_{x_i}{}'$, tedy vzorky o jednu dimenzi menší. Tento krok se nazývá **dimenzionální redukce** problému pro proměnnou x_k, protože jsme tím eliminovali (resp. vyřešili) daný problém pro integrační proměnnou x_k.

Výše uvedené kroky jsou prováděny, dokud nejsou eliminovány všechny integrační proměnné a dokud nezůstane pouze jediná (fixní časová) proměnná $x_j \sim t$. Vyjádřením hodnoty integrálu pro tuto proměnnou tak získáme určitý integrál funkce n reálných proměnných $f(x_1, x_2, \ldots, x_n)$ nad integrační oblastí $\Omega = \langle a_1, b_1\rangle \times \langle a_2, b_2\rangle \times \cdots \times \langle a_n, b_n\rangle$.

Závěrečným krokem, po provedení všech předchozích kroků a po redukci problému na integrál funkce jedné proměnné, je nastavení znaménka výsledku, neboť prohozením integračních mezí dochází k otočení znaménka, a proto je nutné tuto změnu reflektovat i na celkovém výpočtu (výsledek je řešením problému pro časovou integrační proměnnou x_j):

$$\int_{a_n}^{b_n} \cdots \int_{a_2}^{b_2} \int_{a_1}^{b_1} f(x_1, x_2, \ldots, x_n)\,\mathrm{d}x_1\,\mathrm{d}x_2 \cdots \mathrm{d}x_n = J(p_j h_j) \prod_{i=1}^{n} s_i. \tag{3.41}$$

3.3 Přesnost metody

Každá numerická metoda poskytuje pouze přibližný výsledek, který je jakousi aproximací přesného výsledku řešeného problému. Tato přesnost metody je značnou měrou ovlivněna metodami a postupy, které se během numerické metody použijí. Tyto *vnitřní* metody mají každá svou vlastní **relativní** i **absolutní chybu**, a proto je celková přesnost (resp. absolutní chyba) kumulativní chybou všech prováděných *vnitřních* numerických metod.

V předchozích kapitolách byla podrobně popsána numerická metoda pro výpočet určitých integrálů funkcí n reálných proměnných. Značné množství kroků této metody je aplikací Taylorova rozvoje (resp. polynomu). Tato aproximační metoda má svoji přesnost, která se v průběhu celého výpočtu kumuluje. Obecně platí, že čím více vzorků má numerická metoda k dispozici, tím přesnější je její výpočet. Tento počet vzorků lze ovlivňovat nastavením integračního kroku h_{x_i}, což se promítne do počtu dělicích bodů p_{x_i} daného integračního intervalu. Přesnost (resp. chyba) metody je proto podobně jako u metody Monte-Carlo nepřímo úměrně závislá na počtu vzorků. Další chyby jsou zaneseny při výpočtu pseudoinverzní matice a při přibližném výpočtu funkce faktoriál (tento přibližný výpočet je vhodný zejména pro velká čísla, která se v této numerické metodě hojně vyskytují). Vzhledem k četnosti použití těchto numerických metod je zřejmé, že Taylorův rozvoj (resp. polynom) je využíván celkově v N případech, což je mnohonásobně častěji než numerický výpočet pseudoinverzní matice či funkce faktoriál, a proto je možné tyto složky zanedbat:

$$\varepsilon \sim \frac{1}{\sqrt{N}}. \tag{3.42}$$

Absolutní chyba každé aplikace $j \in N$ Taylorova polynomu T_j je vyjádřena v Lagrangeově tvaru zbytku:

$$\varepsilon_{T_j} = R_{n+1}^{f,a}(x) = \frac{f^{(n+1)}(c)}{(n+1)!}(x-a)^{n+1}. \tag{3.43}$$

Celkovou chybu proto můžeme uvažovat jako:

$$\varepsilon = \frac{1}{\sqrt{\prod_{i=0}^{N} \varepsilon_{T_j}}}. \tag{3.44}$$

Z toho vyplývá potřeba určení ideálního počtu členů Taylorova polynomu, aby jeho absolutní chyba byla ε_{T_j}. Ty jsou počítány, dokud se přičtením dalšího členu složeného z vyšší derivace změní celková hodnota Taylorova polynomu o více než ε_{T_j}. Jakmile je přírůstek menší než ε_{T_j}, nemá význam pokračovat ve vyjadřování dalších členů Taylorova polynomu. Pokud však při výpočtu překročíme stanovený maximální počet členů Taylorova polynomu, je nutné rozpůlit krok h_{x_i} a provést nový výpočet, abychom získali přesnější Taylorův polynom s větším počtem členů.

Výše uvedené úvahy jsou pouze teoretické, neboť přesné určení absolutní chyby této metody vyžaduje velmi podrobnou analýzu, která je však nad rámec této práce. Je třeba brát v potaz všechny faktory, které ovlivňují přesnost výpočtu – od přesnosti Taylorova polynomu při výpočtu určitého integrálu jednotlivých vzorků přes přesnost numerického výpočtu derivací po absolutní chybu při výpočtu pseudoinverzní matice či faktoriálu velkých čísel. Všechny tyto nepřesnosti se nelineárně kumulují, proto je určení celkové přesnosti výpočtu velmi komplexním problémem.

Část II

Implementace numerické metody

Kapitola 4

Návrh a implementace aplikace

Tato kapitola je věnována problematice návrhu aplikace implementující v předchozích kapitolách zavedenou numerickou metodu pro výpočet určitých integrálů funkcí více proměnných. V následujícím textu jsou stanoveny požadavky na aplikaci a také je navrženo jejich řešení. Dále je nastíněno uživatelské rozhraní aplikace a její možnosti.

V další části kapitoly je podrobně popsána samotná implementace aplikace včetně praktického řešení implementačně specifických problémů, kterých se návrhová část dotýkala jen teoreticky. Je také popsán způsob, jakým lze aplikaci do budoucna rozšířit či modifikovat.

4.1 Návrh aplikace `hpni`

Aplikace `hpni`[1] byla navržena tak, aby zohledňovala několik základních požadavků a nároků na ni kladených. Mezi hlavní požadavky kladené na aplikaci `hpni` patří její **přenositelnost** na úrovni zdrojového kódu a tedy její použitelnost na libovolném operačním systému, **snadná instalace** a **jednoduché** uživatelské rozhraní s **minimálním**, zato však plně **dostačujícím**, počtem nastavení ovlivňujících běh aplikace.

Mezi implementačně specifické požadavky pak patří zejména schopnost aplikace rychle pracovat s **velmi malými a velmi velkými reálnými čísly**, schopnost aplikace pracovat s libovolným **počtem desetinných míst** těchto reálných čísel a v neposlední řadě **spolupráce s** externí aplikací `TKSL`. Aplikace by měla také **ošetřovat** všechny **neočekávané situace** tak, aby chyba nezpůsobila pád aplikace, ale aby aplikace byla korektně ukončena s chybovou hláškou objasňující příčinu nastavšího problému.

4.1.1 Objektově orientované řešení

Z výše uvedených požadavků přímo vyplývá i způsob řešení – využití **objektově orientovaného** paradigmatu při tvorbě aplikace. Tento přístup umožňuje **oddělení** datové od

[1] Zkratka `hpni` vychází z anglického názvu navržené numerické metody – **h**igh-**p**recision **n**umerical **i**ntegration, tj. vysoce přesná numerická integrace.

výpočetní části aplikace[2]. Dále přináší možnost **abstrakce**[3] či **dědičnosti**[4] (vhodné při ošetřování chybových stavů) a dalších specifických znaků objektově orientovaného programování.

Tento přístup při návrhu aplikace umožní, že při volbě *vhodného* objektově orientovaného programovacího jazyka je **přenositelnost aplikace** (na úrovni zdrojového kódu) zaručena a je podmíněna pouze podporou daného jazyka na cílovém operačním systému (v části 4.2 je vedena diskuze o volbě implementačního jazyka).

Dalším požadavkem na aplikaci je její **snadná instalace**. Vzhledem k tomu, že se aplikace striktně zaměřuje na jeden specifický problém, není potřeba vytvářet nadbytečně komplexní aplikaci, jež by bylo nutné instalovat. V diskuzi v části 4.2 o výběru vhodného programovacího jazyka je zdůvodněno, proč se pro účely této aplikace jeví jako vhodný libovolný interpretovaný jazyk.

Při návrhu rozhraní aplikace byly brány v úvahu všechny vstupní informace, z čehož vyplývají i ovládající prvky. Vstupem aplikace je integrand a informace o integrálních proměnných a jejím výstupem je hodnota určitého integrálu tohoto integrandu v prostoru vymezeném jednotlivými integračními intervaly. Vzhledem k povaze vstupů a výstupů aplikace není požadováno grafické rozhraní, ale **konzolová aplikace** se jeví jako vhodné řešení uživatelského rozhraní. Aplikace je navržena jako **konzolová aplikace** ovládaná přes parametry příkazové řádky a nenabízí tedy interaktivní *shell*. Kromě integrandu a mezí integrálních proměnných je nutné zadat krok pro numerickou metodu pro každou integrální proměnnou; dále je volitelně možné zadat počet zobrazovaných desetinných číslic a přesnost numerické metody při práci s Taylorovým polynomem. Tato nastavení tvoří **minimální**, ale funkčně dostačující množinu ovládacích prvků aplikace.

4.1.2 Další požadavky na řešení

Dalším požadavkem kladeným na aplikaci je schopnost aplikace interně pracovat s **velmi malými a velmi velkými čísly s přesným desetinným rozvojem** (řádově v desítkách až stovkách desetinných číslic). Velmi málo univerzálních programovacích jazyků má vestavěnou podporu takto přesných čísel, proto je nutné sáhnout po speciální implementaci. Způsob řešení tohoto problému je diskutován v části 4.2.

Aplikace `hpni` využívá pro výpočet diferenciálních rovnic **externí program** `TKSL`. Aplikace by tedy měla být navržena tak, aby bylo možné program `TKSL` kdykoliv nahradit novější či jinou verzí např. v závislosti na operačním systému či architektuře.

4.1.3 Návrh tříd v aplikaci

Výše uvedené požadavky a návrh řešení zpracování aplikace vyúsťují v objektově orientovanou aplikaci s následující strukturou tříd:

`core` | **HPNI:** Jádro aplikace řídící celý průběh výpočtu (od transformace integrandu přes

[2] V objektově orientovaném programování (dále jen OOP) jsou **objekty** základní jednotkou modularity i struktury programu. Jednotlivé objekty pak mohou plnit různé funkce – některé objekty tak mohou pouze uchovávat data, jiné mohou s daty pouze manipulovat ve formě provádění výpočtů.

[3] **Abstrakce** umožňuje v OOP modelovat pouze vybranou část problému z reálného světa, další detaily jsou zanedbány a ignorovány. V OOP lze této vlastnosti využít tak, že detaily jsou skryty do **černé skříňky**, která je pro své okolí definována pouze svým rozhraním, přes které s ním komunikuje.

[4] **Dědičnost** je způsob, jak implementovat sdílené chování. Nové objekty tak mohou sdílet a rozšiřovat chování již existujících objektů bez nutnosti redundantní implementace téhož chování.

generování skriptu pro `TKSL` a jeho spuštění po provádění numerického výpočtu derivací a numerické integrace).

`tksl` | **TKSL:** Tato třída nabízí rozhraní pro práci s externím programem `TKSL` (tzn. jeho spuštění a sběr dat). Dále nabízí možnost generování skriptu pro program `TKSL` na základě vstupních informací.

`numeric` | **Numeric:** Implementace numerické metody pro výpočet určitých integrálů funkcí více proměnných, jež byla navržena v předchozích kapitolách. Zde je prováděn numerický výpočet derivací dopřednou, kombinovanou či zpětnou diferencí a zejména pak numerická integrace.

`data` | **data:** Jelikož aplikaci pracuje s velkým množstvím dat, je výhodné je všechny uchovávat na jednom místě a pouze k nim přistupovat (a tedy nekopírovat při každém použití).

Aby aplikace splňovala další požadavky dříve stanovené, musí umožňovat zpracování parametrů příkazové řádky a ošetření chybových stavů:

`args` | **Args:** Tato třída zajišťuje kompletní zpracování a kontrolu parametrů příkazové řádky při spuštění, včetně nápovědy k aplikaci.

`error` | **HPNIError:** Obecná třída popisující způsob ošetření neočekávaného chování či chyby při běhu programu.

- `error` | **IntegralError:** Chyba vznikla nesprávným formátem integrandu.
- `error` | **VariableError:** Obecná třída ošetřující nesprávný formát vstupu integrální proměnné.
  - `error` | **VariableNameError:** Nesprávný název integrální proměnné.
  - `error` | **VariableBoundsError:** Nesprávný formát integrálních mezí.
  - `error` | **VariableStepError:** Nesprávný formát kroku pro numerickou metodu.
- `error` | **PrecisionError:** Nesprávný formát přesnosti numerické metody.
- `error` | **DigitsError:** Nesprávný formát počtu zobrazovaných desetinných míst.
- `error` | **NumberError:** Chybný formát aktuálně zpracovávaného čísla.
- `error` | **TKSLError:** Chyba při komunikaci s externím programem `TKSL`.
- `error` | **NumMethodError:** Chyba při provádění numerického výpočtu derivací nebo numerické integrace.
- `error` | **StepTooBigError:** Stav signalizující, že aktuálně používané kroky pro numerickou metodu produkují výsledek, jež nesplňuje požadovanou přesnost. Ošetřením tohoto stavu dojde k rozpůlení kroků všech integrálních proměnných pro numerickou metodu a výpočet je spuštěn znovu.

4.2 Implementace aplikace `hpni`

V této části bude podrobně popsána samotná implementace aplikace `hpni` a představen způsob, jakým lze aplikaci do budoucna rozšířit či upravit.

4.2.1 Implementační jazyk

Objektově orientovaný návrh aplikace vyžaduje použití jazyka, který implementuje objektově orientovaná paradigmata. V dnešní době existuje spousta takových jazyků (např. Java, $C++$, $C\#$, Smalltalk, Object Pascal, Visual Basic, .NET, PHP, Python, Ruby, ...), z nichž pro účely této práce byl vybrán jazyk `Python 3` (ve verzi alespoň `3.3.2`). Tento jazyk se řadí mezi tzv. *čistě objektově orientované*, protože výpočty probíhají výhradně pomocí vzájemného zasílání zpráv mezi objekty.

Mezi hlavní přednosti jazyka `Python 3.3.2` patří především jeho rozšířenost mezi uživateli a velké množství rozšiřujících knihoven a modulů. Právě použitím několika specializovaných modulů je možné naplnit již dříve zmíněné požadavky. Tento jazyk také umožňuje ošetření výjimek, ať už vestavěných nebo uživatelem definovaných. Této vlastnosti je využito při implementaci ošetření výjimečných stavů, které by mohly vést k pádu aplikace. Mezi další přednosti tohoto jazyka bezesporu patří vysoce kvalitní dokumentace a autorova znalost tohoto jazyka.

Vzhledem k tomu, že aplikace `hpni` je implementována v jazyce `Python 3.3.2`, jedná se podle konvencí tohoto jazyka spíše o **balíček**. Proto jsou dodržena pravidla[5], která jsou stanovena pro tvorbu balíčků. Při programování aplikace byly dodrženy rady a doporučení uvedené v *PEP8 – Style Guide for Python Code*[6]. Zdrojový kód byl také analyzován nástrojem `pylint`, který ohodnotil kvalitu zdrojového kódu známkou 7.81 z 10, což značí poměrně vyspělý projekt.

Jelikož jazyk `Python 3.3.2` je velmi modulární, existuje pro něj spoustu rozšíření, knihoven a modulů. Při vývoje aplikace `hpni` byly použity tyto moduly: `enum34`[7], `gmpy2`[8] (viz [10]), `numpy`[9] (viz [20]) a `SymPy`[10] (viz [29]). Využití jednotlivých modulů bude popsáno v následující pasáži o implementaci jednotlivých tříd.

4.2.2 Implementace tříd

Aplikace se sestává z vzájemně komunikujících tříd tak, jak již bylo nastíněno v kapitole 4.1. Na obrázku 4.1 je znázorněn diagram vztahů mezi moduly. Obecně platí, že každý modul implementuje jednu třídu, ale v některých modulech je implementováno více tříd (např. `error` a `misc`).

Dále budou diskutovány jen ty třídy, které mají přímou souvislost s řešeným problémem numerické integrace. Nebudou tedy vysvětlovány žádné pomocné třídy (např. `Singleton`, aj.), které jsou aplikaci definovány, ale nesouvisí přímo s tématem. U každé třídy bude uveden modul, ve kterém se nachází její definice.

Třída `HPNI` (modul `core`)

Třída `HPNI` je jádrem celé aplikace, je jakými *mozkem* celého výpočtu, který zpracovává vstupní informace a nějakým způsobem tyto informace transformuje.

[5] Pravidla pro tvorbu balíčku v jazyce `Python 3.3.2` dostupná online `<https://docs.python.org/3.3/tutorial/modules.html#packages>`.

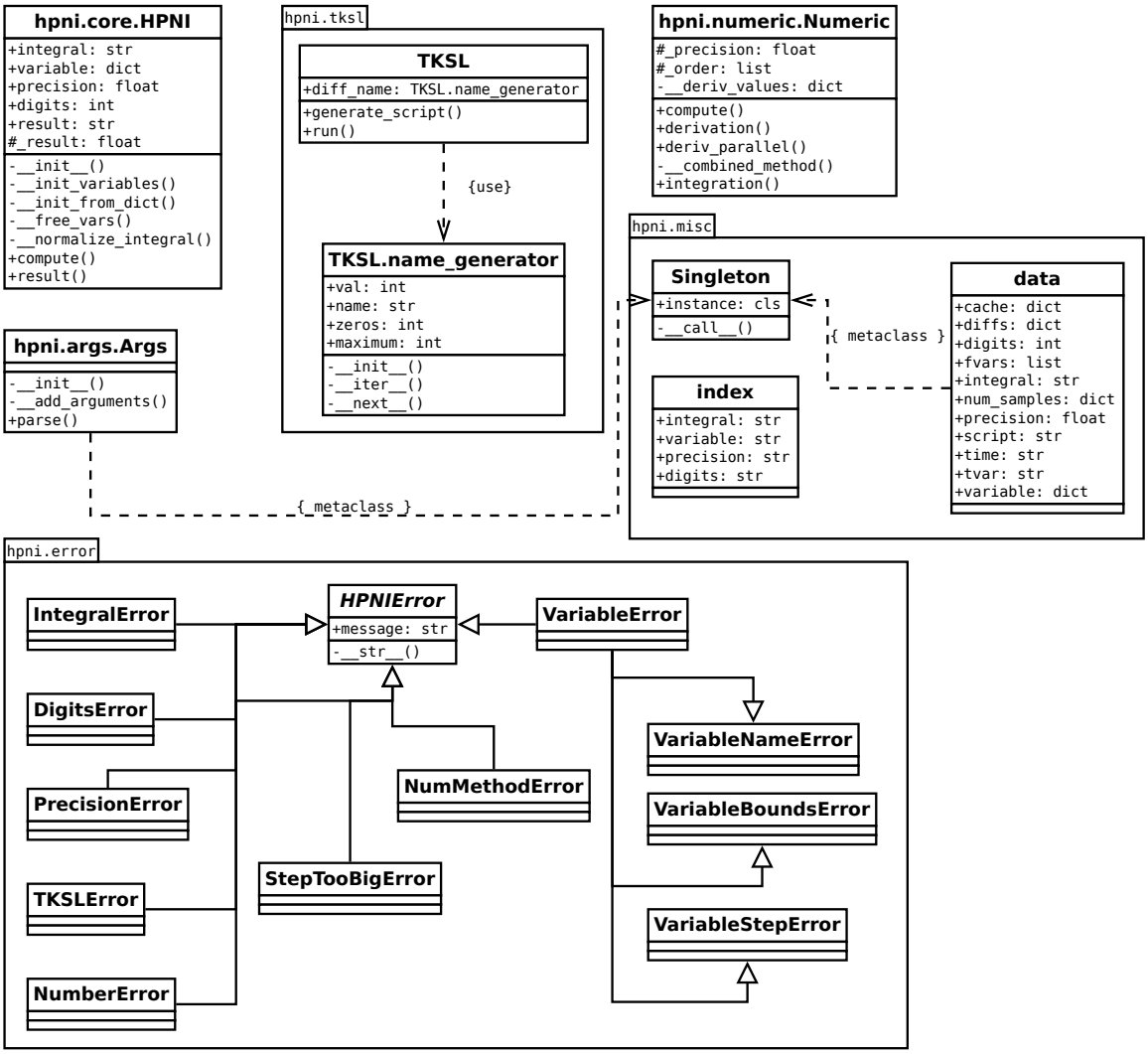
[6] Pravidla pro tvorbu kvalitního kódu dostupná online `<http://legacy.python.org/dev/peps/pep-0008/>`.

[7] Více o modulu `enum34` viz `<https://pypi.python.org/pypi/enum34/0.9.23>`.

[8] Více o modulu `gmpy2` viz `<https://pypi.python.org/pypi/gmpy2/2.0.3>`.

[9] Více o modulu `numpy` viz `<https://pypi.python.org/pypi/numpy/1.8.1>`.

[10] Více o modulu `SymPy` viz `<https://pypi.python.org/pypi/sympy/0.7.5>`.



Obrázek 4.1: Diagram tříd a jejich vzájemných vztahů (vlastní tvorba).

Při inicializaci instance této třídy jsou nejprve zpracovány vstupní informace – tzn. integrand, integrální proměnné včetně jejich dolních a horních mezí a kroku pro numerickou metodu, přesnost výpočtu numerické metody a také počet zobrazovaných desetinných číslic. Poté je provedena analýza integrandu ve smyslu vyhledání všech integrálních proměnných a kontrola, zda pro všechny *skutečné* integrální proměnné byly při spuštění programu poskytnuty potřebné informace. Dále následuje transformace integrandu – tj. translace problému do počátku souřadné soustavy a adekvátní úprava integrandu.

Po provedení těchto základních kroků se začne provádět samotný výpočet určitého integrálu. Nejdříve se vygeneruje skript pro `TKSL`, poté se tento skript předá `TKSL` k vyřešení a výsledné hodnoty jsou sesbírány. Nad těmito hodnotami se pak v cyklu provádí numerický výpočet derivací následovaný numerickou integrací tak, že v každém cyklu eliminuje jedná integrální proměnná a řešený problém se tím redukuje o jednu proměnnou, dokud nezbude jen jedna proměnná.

Třída `TKSL` (modul `tksl`)

Třída `TKSL` implementuje rozhraní po komunikaci s externím programem `TKSL`, který je implementován v jazyce `C++` využívající knihovnu `GMP`. Aby bylo možné zasáhnout do libovolného bodu při běhu aplikace `hpni`, bylo nutné vytvořit rozhraní v jazyce `Python 3` komunikující s programem `TKSL`. Toto rozhraní nastaví všechny potřebné parametry příkazové řádky a vygeneruje skript, který je ve formátu akceptovatelným programem `TKSL`, aby mohl být proveden žádoucí výpočet. Po dokončení výpočtu jsou sesbírány výsledky, které jsou předány dál numerické metodě.

Implementačně je generování skriptu řešeno tak, že je vytvořena sada vzorků pro každou integrální proměnnou a proměnná, která vytváří největší sadu, je určena jako fixní časová proměnná a tudíž celá tato sada je nahrazena časovou proměnnou. Poté je vypočten kartézský součin všech těchto sad vzorků jednotlivých integrálních proměnných. Pro každý prvek tohoto kartézského součinu je pak vygenerována diferenciální rovnice ve tvaru akceptovatelném programem `TKSL` – každá integrální proměnná je v diferenciální rovnici nahrazena aktuální hodnotou ve vzorku. Pro vytvoření sady vzorků pro každou integrální proměnnou byla použita knihovna pro numerickou matematiku `numpy`; kartézský součin pak je vypočten pomocí modulu `itertools`, jež se standardní součástí distribuce `Python 3.3.2`.

Jelikož program `TKSL` je externí aplikace, je nutné zvolit vhodný způsob jeho spuštění a komunikace s ním. K těmto účelům byl zvolen modul `subprocess`, který je rovněž součástí standardní distribuce. Ten umožňuje spustit jakýkoliv externí program včetně parametrů příkazové řádky, předat tomuto programu vstupní skript na `stdin`[11] a číst vypočtené výsledky z `stdout`[12] a chyby z `stderr`[13]. Programem `TKSL` vypočtené diferenciální rovnice jsou pak spárovány se správnými vzorky a ty jsou pak postoupeny dál numerické metodě.

Vzhledem k tomu, že aplikace `hpni` využívá k výpočtu diferenciálních rovnic program `TKSL`, jsou její schopnosti závislé a omezené schopnostmi `TKSL`. To však nijak neovlivňuje funkčnost aplikace `hpni`, neboť `TKSL` je kdykoli možné nahradit novější či jinou verzí.

Třída `Numeric` (modul `numeric`)

V předchozích kapitolách diskutovaná numerická metoda je implementována v rámci třídy `Numeric`. Tato třída zpracovává vstupní data (což jsou výstupy `TKSL`) a střídavě numericky počítá derivace a integrály.

Nejprve jsou integrální proměnné sestupně seřazeny (implementace uvažuje alfanumerické řazení, ale toto je možné nahradit jakýmkoliv jiným řazením), poté je z tohoto seznamu odstraněna fixní časová proměnná a je připojena na konec seznamu. Toto seřazení reprezentuje způsob, jakým bude prováděna redukce problému (resp. v jakém pořadí budou eliminovány jednotlivé dimenze problému). Dále v cyklu probíhají následující operace:

1. Vzorky s výsledky vypočtenými v `TKSL` jsou seskupeny podle *následující* integrální proměnné v sestupně seřazeném seznamu proměnných určených k eliminaci. Tímto získáme skupiny vzorků, které pro konkrétní hodnotu *následující* proměnné obsahují všechny vzorky *aktuální* integrální proměnné.

2. V dalším kroku jsou pro každou konkrétní hodnotu *následující* proměnné numericky vypočteny derivace dopřednou, zpětnou nebo kombinovanou diferencí z hodnot, které

[11] Zkratka `stdin` znamená „standardní vstup“.
[12] Zkratka `stdout` znamená „standardní výstup“.
[13] Zkratka `stderr` znamená „standardní chybový výstup“.

jsou ve skupině vzorků pro danou hodnotu. Takto tedy vypočteme tolik řádů derivací, kolik je hodnot v dané skupině.

3. Z takto vypočtených derivací je sestaven Taylorův polynom vyjadřující numerickou integraci pro konkrétní hodnotu *následující* proměnné. Dále pak je eliminována *aktuální* proměnná a problém je tak redukován o jednu dimenzi.

* Předchozí body se opakují, dokud problém není redukován na jednu dimenzi.

Je zřejmé, že některé z předchozích kroků se budou provádět velmi často a navíc, že některé z nich nejsou na sobě závislé, tudíž jsou paralelizovatelné. Pro zvýšení rychlosti výpočtu proto byly implementovány funkce využívající **LRU** (z anglického **L**east **R**ecently **U**sed) **cache**. Mezi takto modifikované a výsledky výpočtů ukládající funkce patří například výpočet faktoriálu velmi velkých čísel, výpočet koeficientů používaných při numerickém výpočtu derivací a při numerické integraci. Výpočet derivací i numerická integrace je **paralelizovatelná** úloha, což je implementováno i v případu aplikace `hpni`.

Třída `data` (modul `misc`)

Klíčovým požadavkem kladeným na aplikaci `hpni` je efektivní správa paměti a vhodné uložení dat. Aby se předešlo neustálému kopírování dat a mezivýpočtů, je navržena a implementována třída `data` podle návrhového vzoru *Singleton*, která funguje jako globální datové úložiště. Výhoda tohoto přístupu spočívá zejména v tom, že data jsou uloženy jen jednou a lze k nim přistupovat ze kterékoliv části programu. V tomto úložišti je uložen transformovaný integrand, upravené integrální proměnné, výsledky výpočtů prováděných programem `TKSL` a další. Tato implementace měla velmi pozitivní vliv na rychlost a paměťovou náročnost aplikace.

4.2.3 Možnosti rozšíření aplikace

Doposud bylo popsáno, jakým způsobem byla aplikace navržena a tento návrh implementován. Byly také popsány možná úskalí a jejich praktické řešení. Avšak největší omezení aplikace není dáno jí samotnou, nýbrž komunikací s externím programem `TKSL`. Proto se jako velmi vhodné rozšíření aplikace `hpni` jeví vlastní implementace programu `TKSL` v jazyce `Python 3`, aby práce (nejen) tohoto programu byla rychlejší a konfigurovatelná. Stávající implementace programu `TKSL` je natolik rozsáhlá, že jí byla věnována celá diplomová práce (viz [9, 19]), a proto to není součástí této práce.

Kapitola 5

Testování a zhodnocení implementace

V průběhu vývoje byla aplikace `hpni` testována na množině určitých integrálů, jež je uvedena dále v textu. Během tohoto testování byly sledovány zejména dva faktory – rychlost aplikace (resp. numerické metody) a její přesnost (resp. přesnost výsledku vypočteného numerickou metodou). Výsledky aplikace byly srovnávány s jinými implementacemi jako například `SciPy` či `Wolfram|Alpha`; toto srovnání je uvedeno v části 5.2. Testování a zejména následné srovnání s jinými implementacemi poskytuje velmi cennou zpětnou vazbu o reálném výkonu a schopnostech numerické metody a její implementace.

5.1 Metodika testování

Pro testování byla sestavena sada problémů takových, aby byly zastoupeny různé funkce více proměnných (od konstantní funkce až po funkci se čtyřmi proměnnými), aby byla zřejmá závislost výsledku výpočtu na vztahu dolních a horních integrálních mezí a aby byla zřejmá závislost rychlosti výpočtu numerické metody na zadané přesnosti. Během testování byly modifikovány tyto parametry:

- přesnost numerické metody: 1E-10 a 1E-5,
- počet zobrazovaných desetinných míst: 10 a 5,
- výchozí krok numerické metody pro každou proměnnou: 0.1 a 0.01.

V tabulce 5.1 je uvedena testovací sada funkcí a integračních mezí pro výpočet určitých integrálů. Všechny tyto funkce byly předloženy `hpni` k výpočtu; pro ověření správnosti výpočtu byly tyto určitého integrály též vypočteny aplikacemi `SciPy` (modul pro `Python`) a `Wolfram|Alpha`.

počet proměnných	**integrand**	**integrační interval** $\{\min \dots \max\}$
0	$\sin\left(\cos\left(\frac{\pi}{2}\right)\right)$	$t = \{0 \dots 1\}$
1	$\exp(\sin(a))$	$a = \{1 \dots 0\}$
2	$\exp(\sin(a)) \cdot \cos(-b)$	$a = b = \{0 \dots 1\}$
3	$\sin(a) \cdot \cos(b) \cdot c$	$a = b = c = \{0 \dots 1\}$
4	$a - \sin(b) \cdot \cos(c) \cdot d$	$a = b = c = d = \{0 \dots 1\}$

Tabulka 5.1: Testovací sada integrálů.

V tabulkách 5.2 až 5.6 jsou shrnuty výsledky výpočtů jednotlivých určitých integrálů z tabulky 5.1 při různém nastavení aplikace `hpni` – přesnost výpočtu (sloupec *přesnost*), počet zobrazovaných desetinných míst (sloupec *rozsah*) a krok numerické metody pro jednotlivé integrační proměnné (sloupce t, a až e). Nejpřesnější a nejrychlejší výpočet `hpni` je vždy srovnán s výpočtem `SciPy` a `Wolfram|Alpha`.

přesnost	rozsah	t	výsledek	čas $[s]$
1E-10	10	0.1	0	3.312
1E-10	10	0.01	0	3.342
1E-10	5	0.1	0	3.334
1E-10	5	0.01	0	3.393
1E-5	10	0.1	0	3.304
1E-5	10	0.01	0	3.327
1E-5	5	0.1	0	3.302
1E-5	5	0.01	0	3.321

implementace	výsledek	čas $[s]$
`hpni`	0	3.302
`SciPy`	6.123 233 995 736 766E-17	0.490
`Wolfram|Alpha`	0	4.400

Tabulka 5.2: Výsledky testování `hpni` na výpočtu určitého integrálu konstantní funkce.

přesnost	rozsah	t	výsledek	čas $[s]$
1E-10	10	0.1	−1.6318696084	3.290
1E-10	10	0.01	−1.6318696084	3.334
1E-10	5	0.1	−1.63186	3.294
1E-10	5	0.01	−1.63186	3.370
1E-5	10	0.1	−1.6318696084	3.332
1E-5	10	0.01	−1.6318696084	3.318
1E-5	5	0.1	−1.63186	3.304
1E-5	5	0.01	−1.63186	3.338

implementace	výsledek	čas $[s]$
`hpni`	−1.6318696084	3.290
`SciPy`	−1.6318696084180513	0.500
`Wolfram|Alpha`	−1.63187	6.540

Tabulka 5.3: Výsledky testování `hpni` na výpočtu určitého integrálu funkce 1 proměnné.

přesnost	rozsah	a	b	výsledek	čas $[s]$
1E-10	10	0.1	0.1	1.43572415	3.967
1E-10	10	0.1	0.01	0.1322528665	3.628
1E-10	10	0.01	0.1	0.1406432674	3.633
1E-10	10	0.01	0.01	1.376910317	11.682
1E-10	5	0.1	0.1	1.37514	3.541
1E-10	5	0.1	0.01	0.13225	3.667
1E-10	5	0.01	0.1	0.14064	3.672
1E-10	5	0.01	0.01	1.37691	11.695
1E-5	10	0.1	0.1	1.409518306	3.554
1E-5	10	0.1	0.01	0.1317821379	3.606
1E-5	10	0.01	0.1	0.1409534753	3.629
1E-5	10	0.01	0.01	1.376910317	11.469
1E-5	5	0.1	0.1	1.40951	3.501
1E-5	5	0.1	0.01	0.13178	3.586
1E-5	5	0.01	0.1	0.14095	3.628
1E-5	5	0.01	0.01	1.37691	11.424

implementace	výsledek	čas $[s]$
`hpni`	1.37514	3.541
`SciPy`	1.3731709264736138	0.490
`Wolfram|Alpha`	1.37317	6.620

Tabulka 5.4: Výsledky testování `hpni` na výpočtu určitého integrálu funkce 2 proměnných.

přesnost	rozsah	a	b	c	výsledek	čas $[s]$
1E-10	10	0.1	0.1	0.1	0.2123328456	13.390
1E-10	10	0.1	0.1	0.01	0.0211433738	16.223
1E-10	10	0.1	0.01	0.1	0.0212332846	20.064
1E-10	10	0.1	0.01	0.01	0.1752759561	91.905
1E-10	10	0.01	0.1	0.1	0.0212332846	19.116
1E-10	10	0.01	0.1	0.01	0.1961150925	92.530
1E-10	10	0.01	0.01	0.1	0.0176156709	33.664
1E-10	10	0.01	0.01	0.01	N/A	N/A
1E-10	5	0.1	0.1	0.1	0.1917	5.340
1E-10	5	0.1	0.1	0.01	0.01773	5.609
1E-10	5	0.1	0.01	0.1	0.01753	16.180
1E-10	5	0.1	0.01	0.01	0.17528	91.881
1E-10	5	0.01	0.1	0.1	0.01961	6.002
1E-10	5	0.01	0.1	0.01	0.19611	92.051
1E-10	5	0.01	0.01	0.1	0.01762	33.475
1E-10	5	0.01	0.01	0.01	N/A	N/A
1E-5	10	0.1	0.1	0.1	0.1786811614	5.309
1E-5	10	0.1	0.1	0.01	0.0180194799	5.578
1E-5	10	0.1	0.01	0.1	0.015799644	5.923
1E-5	10	0.1	0.01	0.01	0.1737926177	90.959
1E-5	10	0.01	0.1	0.1	0.0178683246	5.945
1E-5	10	0.01	0.1	0.01	0.1965476511	90.301
1E-5	10	0.01	0.01	0.1	0.0174544048	32.723
1E-5	10	0.01	0.01	0.01	N/A	N/A
1E-5	5	0.1	0.1	0.1	0.17868	5.311
1E-5	5	0.1	0.1	0.01	0.01802	5.606
1E-5	5	0.1	0.01	0.1	0.0158	5.913
1E-5	5	0.1	0.01	0.01	0.17379	89.611
1E-5	5	0.01	0.1	0.1	0.01787	5.923
1E-5	5	0.01	0.1	0.01	0.19655	90.791
1E-5	5	0.01	0.01	0.1	0.01745	32.784
1E-5	5	0.01	0.01	0.01	N/A	N/A

implementace	výsledek	čas $[s]$
`hpni`	0.1917	5.340
`SciPy`	0.19341113569752783	0.500
`Wolfram|Alpha`	0.193411	6.540

Tabulka 5.5: Výsledky testování `hpni` na výpočtu určitého integrálu funkce 3 proměnných.

přesnost	rozsah	a	b	c	d	výsledek	čas $[s]$
				...			
1E-5	10	0.1	0.1	0.1	0.1	0.3568070755	217.728
				...			

implementace	výsledek	čas $[s]$
`hpni`	0.3568070755	217.728
`SciPy`	0.3065888643024722	3.110
`Wolfram|Alpha`	0.306589	8.750

Tabulka 5.6: Výsledky testování `hpni` na výpočtu určitého integrálu funkce 4 proměnných.

Z výsledků testovaní uvedených v tabulkách 5.2 až 5.6 je zřejmé, že numerická metoda poskytuje velmi nepřesné výsledky, pokud není integrační oblast dělena rovnoměrně; podobně jako kvadraturní vzorce, tak i tato numerická metoda **vyžaduje ekvidistantní**

síť dělicích bodůsw. Dále je zřejmé, že příliš jemné dělení integrační oblasti nepřináší přesnější výsledek. Bylo také zjištěno a potvrzeno, že při práci s méně desetinnými místy anebo s menší přesností je numerický výpočet v průměru rychlejší. Nastavení těchto dvou parametrů výrazně ovlivňuje funkčnost aplikace (např. při nastavení velmi malé hodnoty přesnosti dochází k problémům při výpočtech v programu `TKSL`).

Ze srovnání jednotlivých implementací (`hpni`, `SciPy` a `Wolfram|Alpha`) je zřejmé, že `hpni` není co do rychlosti tou nejlepší variantou. Přestože aplikace `hpni` je implementována jako paralelní aplikace využívající globální sdílené úložiště dat, zvolený implementační jazyk a jeho režie způsobují, že výpočty jsou několikanásobně pomalejší než v případě jiných implementací. Tento nedostatek by bylo možné odstranit nebo alespoň výrazně zlepšit volbou jiného (více nízkoúrovňového) jazyka (např. `C`). Proto byly zkoumány zdrojové kódy modulu `SciPy`, který je *téměř* celý implementován v jazyce `Python`, ale části týkající se numerických výpočtů využívají původní implementaci numerických metod `QUADPACK` v jazyce `Fortran`. Tato implementace je natolik efektivní, že její přepis do *modernějšího* jazyka nepřináší žádné výrazné zlepšení, a proto spousta numerických knihoven a modulů pro různé programovací jazyky využívá právě tuto původní implementaci v jazyce `Fortran`.

5.2 Zhodnocení implementace

Při pohledu na implementaci aplikace `hpni` a její srovnání s jinými dostupnými implementacemi (viz část 5) je zřejmé, že tato implementace není nejrychlejší ani nejpřesnější, stále však v rozumném čase poskytuje relativně přesné výsledky. Tyto základní nedostatky lze silně ovlivnit volbou jiného implementačního jazyka a zejména společnou implementací programu `TKSL` a `hpni`. Efektivitu aplikace `hpni` by bylo možné zvýšit optimalizací ukládání velkého množství dat a přístupu k nim, provést revizi paralelního zpracování nebo také úpravou části algoritmu, ve které se počítají jednotlivé derivace ze všech dostupných okolních hodnot, tak, že by byl staven maximální počet okolních hodnot, z nichž se jednotlivé derivace počítají.

Kapitola 6

Závěr

Integrální a diferenciální počet je v dnešní době aktuální nejen z hlediska teoretického, ale také z hlediska praktického využití v mnoha průmyslových odvětvích. Ať už se jedná o aplikovanou matematiku, fyziku či chemii ve stavebnictví, meteorologii, konstruktérství či návrhu speciálních materiálů, je pro vyhodnocování často velmi komplexních modelů a výpočtů potřeba využít diferenciálních a integrálních rovnic. Avšak jejich řešení není vždy zcela jednoduché a přímočaré – analytické řešení je úspěšně použitelné pouze pro poměrně omezenou podmnožinu úloh, a tak se nabízí využití metod numerických. Tyto metody jsou mnohdy několikanásobně rychlejší, je-li k dispozici např. specializovaný hardware, a zpravidla snazší na pochopení a implementaci. Tyto metody však mají svá omezení, díky kterým nejsou výpočty zcela přesné. Může se však stát, že zvolená numerická metoda není pro řešení daného problému použitelná vůbec.

V této diplomové práci byla připomenuta problematika integrálního počtu funkcí jedné a více proměnných, uvedeny základní definice vymezující tuto oblast matematické analýzy, nastíněny obecně využívané analytické metody. Dále byly připomenuty základní numerické metody využívané pro řešení určitých integrálů. Stěžejní částí této práce byl návrh nové numerické metody pro výpočet určitých integrálu jedné a více proměnných.

Tato nová metoda nejprve provádí transformaci řešeného určitého integrálu, pak vytváří množinu vzorkovaných funkcí integrandu, které jsou převedeny na diferenciální rovnice. Tyto rovnice jsou řešeny aplikací Taylorova polynomu. Taylorův polynom je v této metodě využíván také později při numerické integraci. Aby byl tento krok proveditelný, je nutné vyjádřit velké množství derivací integrované funkce ve vybraných bodech, což je také prováděno numerickou metodou (dopřednou, zpětnou nebo kombinovanou diferencí). V konečném důsledku je tato numerická metoda poměrně robustní a přijatelně přesná a rychlá.

Tato numerická metoda byla implementována v jazyce `Python 3` s využitím knihovny `GMPY2` pro práci s velkými čísly. Aplikace `hpni` využívá proprietární software `TKSL` pro řešení diferenciálních rovnic, takže její funkcionalita také závisí na funkcionalitě tohoto externího programu. Aplikace byla testována a výsledky testování byly porovnány s jinými dostupnými implementacemi (WolframAlpha, SciPy). Návrhem a stávající implementací však problematika numerické integrace funkcí více proměnných není uzavřena. Nabízí se hned několik rozšíření a zdokonalení metody a implementace – automatická transformace integrační oblasti, pokud se jedná o funkcemi zadané meze, nebo přidání grafické nástavby animující průběh numerické integrace. Dále by bylo vhodné sjednotit implementaci programu `hpni` a externího programu `TKSL` ve smyslu implementačního jazyka, tak vzájemné provázanosti. Pro zvýšení efektivity implementace by bylo vhodné provést optimalizaci

ukládání velkého množství dat a přístupu k nim, zavést ještě více paralelismu do výpočtu a v neposlední řadě zvolit méně obecný a vysokoúrovňový implementační jazyk (např. `C/C++`, `Fortran`, aj.). Definice nové numerické metody a výsledky její implementace však naplnilo požadavky a očekávání.

Literatura

[1] Burden, R. L.; Faires, J. D.: *Numerical Analysis*. Cengage Learning, 2010, ISBN 978-0-538-73351-9.
URL <http://ins.sjtu.edu.cn/people/mtang/textbook.pdf>

[2] Cheney, E. W.; Kincaid, D. R.: *Numerical Mathematics and Computing*. Cengage Learning, 7 vydání, 2013, 704 s.

[3] Collins, P. J.: *Differential and Integral Equations*. Oxford University Press, 2006, ISBN 978-0-19-929789-4, 386 s.
URL <http://uqu.edu.sa/files2/tiny_mce/plugins/filemanager/files/4282396/DIFFERENTIAL%20AND%20INTEGRAL%20EQUATIONS.pdf>

[4] Courrieu, P.: Fast Computation of Moore–Penrose Inverse Matrices. *Neural Information Processing – Letters and Reviews*, ročník 8, č. 2, 2005.
URL <http://bsrc.kaist.ac.kr/nip-lr/V08N02/V08N02P2-25-29.pdf>

[5] Fajmon, B.; Růžičková, I.: *Matematika 3*. Brno: Vysoké učení technické v Brně, 2007.
URL <http://www.umat.feec.vutbr.cz/~fuchsp/Matematika3.pdf>

[6] Grasselli, M.; Pelinovsky, D.: *Numerical Mathematics*. Jones & Bartlett Learning, 2008, ISBN 9780763737672, 668 s.

[7] Hammer, P. C.; Stroud, A. H.: Numerical Evaluation of Multiple Integrals II. *Mathematics of Computation*, ročník 12, č. 64, 1958: s. 272–280, ISSN 0025-5718, doi:10.1090/s0025-5718-1958-0102176-6.

[8] Hirayama, H.: Fast Numerical Integration Method Using Taylor Series. In *Integral Methods in Science and Engineering*, editace C. Constanda; S. Potapenko, Birkhäuser Boston, 2008, ISBN 978-0-8176-4670-7, s. 135–140, doi:10.1007/978-0-8176-4671-4˙16.

[9] Holub, O.: *Numerické řešení rozsáhlých soustav diferenciálních a algebraických rovnic*. Diplomová práce, Vysoké učení technické v Brně, Brno, 1999.

[10] Horsen, C. V.: gmpy: Multiple–precision arithmetic for Python. Květen 2014.
URL <https://code.google.com/p/gmpy/>

[11] Jarník, V.: *Integrální počet I.* Praha: Academia, 6 vydání, 1984, 243 s.

[12] Jarník, V.: *Integrální počet II.* Praha: Academia, třetí vydání, 1984, 763 s.

[13] Kalas, J.; Kuben, J.: *Integrální počet funkcí více proměnných*. Brno: Masarykova univerzita, 2009, ISBN 978-80-210-4975-8.

[14] Kiusalaas, J.: *Numerical Methods in Engineering with Python 3*. Cambridge University Press, 2013, ISBN 9781107033856.

[15] Kopáček, J.: *Matematická analýza nejen pro fyziky I.* Praha: MatfyzPress, čtvrté vydání, 2004, ISBN 80-86732-25-8, 192 s.

[16] Krupková, V.; Fuchs, P.: *Matematická analýza pro FIT*. Brno: Vysoké učení technické v Brně, 2014.
URL <http://www.umat.feec.vutbr.cz/~krupkova/IMA2014.pdf>

[17] Kunovský, J.: *Modern Taylor Series Method*. Habilitační práce, Vysoké učení technické v Brně, Brno, 1995.
URL <http://www.fit.vutbr.cz/research/view_pub.php.cs?id=6069>

[18] Kunovský, J.: Real–Time Applications of the Taylor Series. In *Proceedings of ADIUS 2000, ADI*, Boston, 2000, s. 67–75.
URL <http://www.fit.vutbr.cz/research/view_pub.php.cs?id=6086>

[19] Kunovský, J.: Vysoce náročné výpočty. Květen 2014.
URL <http://www.fit.vutbr.cz/~kunovsky/TKSL/index.html.cs>

[20] Numpy developers: NumPy. Květen 2014.
URL <http://www.numpy.org/>

[21] Peringer, P.: *Simulační nástroje a techniky*. Brno: Vysoké učení technické v Brně, 2014.
URL
<http://www.fit.vutbr.cz/study/courses/SNT/public/Prednasky/SNT.pdf>

[22] Peringer, P.; Hrubý, M.: *Modelování a simulace*. Brno: Vysoké učení technické v Brně, 2013.
URL
<http://www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf>

[23] Piessens, R.; deDoncker Kapenga, E.; Überhuber, C. W.; aj.: *QUADPACK: A subroutine package for automatic integration*. Berlin: Springer, 1983, ISBN 978-3-642-61786-7.
URL <http://nines.cs.kuleuven.be/software/QUADPACK/>

[24] Python Software Foundation: multiprocessing: Process–based parallelism. Květen 2014.
URL <https://docs.python.org/3/library/multiprocessing.html>

[25] Quarteroni, A.; Sacco, R.; Saleri, F.: *Numerical Mathematics*. Berlin: Springer, druhé vydání, 2007, ISBN 978-3-540-49809-4.

[26] Schlesingerová, E.: *Web k tématu: Integrální počet v* $\mathbb{R}$. Bakalářská práce, Masarykova univerzita, Brno, 2006.
URL <http://www.math.muni.cz/~xschlesi/bakalarka/>

[27] Schlesingerová, E.: *Web k tématu: Integrální počet*. Diplomová práce, Masarykova univerzita, Brno, 2009.
URL <http://www.math.muni.cz/~xschlesi/dp/>

[28] SciPy developers: SciPy. Květen 2014.
URL <http://scipy.org/>

[29] SymPy Development Team: SymPy. Květen 2014.
URL <http://sympy.org/>

[30] Taylor, M. E.: Introduction to Calculus in Several Variables. 2012.
URL <http://www.unc.edu/math/Faculty/met/mvcalc.pdf>

[31] Vodrážková, E.: *Singulární rozklad matice a jeho použití*. Diplomová práce, Masarykova univerzita, 2012.
URL <http://is.muni.cz/th/211561/prif_m/Vodrazkova_DP.pdf>

Část III

Přílohy

Příloha A

Instalace a návod na použití

Vytvořená knihovna a aplikace `hpni` jsou spustitelné na libovolném operačním systému, který podporuje `Python 3` a pro nějž existují `GMP` knihovny, jež jsou nezbytné pro běh proprietárního software `TKSL`. Vzhledem k uzavřenosti zdrojového kódu tohoto software je na přiloženém CD distribuována pouze jedna verze, která je přeložena pro 64 bitový Linux. Další nezbytné knihovny a moduly jsou součástí přiloženého CD.

A.1 Obsah CD

Soubory jsou na CD uloženy v následující adresářové struktuře:

bin/	Přeložené binární soubory (`TKSL`) a vyžadované `Python 3` moduly.
doc/	Zdrojové soubory textové části práce.
	img/ Grafické soubory použité v textové části práce.
	style/ LaTeX šablona pro textovou část práce a pravidla pro citace.
hpni/	Zdrojové soubory programové části práce.
hpni.py	Spustitelný skript.
LICENSE	Celkové znění `GPL v3` licence.
README.md	Stručný popis programové části práce a její použití.

Tabulka A.1: Obsah CD.

A.2 Instalace

Program `hpni` vyžaduje pro svůj běh interpret jazyka `Python 3` ve verzi `3.3.2` nebo vyšší. Dále pak vyžaduje některé moduly jazyka `Python 3` – modul `enum34` zavádějící výčtový typ pro jazyk `Python`, modul `gmpy2` umožňující pracovat s knihovnou `GMP` z prostředí jazyka `Python`, moduly `numpy` a `sympy` pro numerické a symbolické výpočty v prostředí jazyka `Python`. Všechny tyto programy a moduly jsou součástí přiloženého CD a jsou umístěny v adresáři `bin/`.

Některé moduly mohou mít další závislosti (např. modul `gmpy2` má závislost na knihovnu `GMP`), proto je nutné tyto závislosti vyřešit manuálně. Podobná situace nastává u proprietárního software `TKSL`, který je závislý na knihovně `GMP` a knihovnách jazyka `C++`.

Pro instalaci těchto programů a modulů je vhodné postupovat v níže uvedeném pořadí.

Archívy rozbalte a nainstalujte do adresářů k tomu určených podle zvyklostí cílového operačního systému.

1. `Python 3.3.2`:
   - Linux: `bin/Python-3.3.2.tar.xz`
   - Fedora: `$ yum install -y python3 python3-pip`
2. `enum34 0.9.23`:
   - Linux: `bin/enum34-0.9.23.tar.gz`
   - Fedora: `$ pip-python3 install --upgrade enum34`
3. `gmpy2 2.0.3`:
   - Linux: `bin/gmpy2-2.0.3.zip`
   - Fedora: `$ pip-python3 install --upgrade gmpy2`
4. `numpy 1.8.1`:
   - Linux: `bin/numpy-1.8.1.tar.gz`
   - Fedora: `$ pip-python3 install --upgrade numpy`
5. `sympy 0.7.5`:
   - Linux: `bin/sympy-0.7.5.tar.gz`
   - Fedora: `$ pip-python3 install --upgrade sympy`

A.3 Návod na použití

Aplikace byla navržena tak, aby poskytovala minimální a uživatelsky přívětivé rozhraní. Všechny informace jsou aplikaci předávány jako parametry pro spuštění.

Nápověda k aplikaci

Výčet všech povinných a volitelných parametrů je uveden v nápovědě k aplikaci:

```
$ ./hpni.py -h
usage: hpni [-h] -i INTEGRAL -v VARIABLE [VARIABLE ...] [-p PRECISION]
            [-d DIGITS] [--verbose]

High-precision numerical integration method using Taylor series and numerical
method for derivations' computation.

optional arguments:
  -h, --help            show this help message and exit
  -i INTEGRAL, --integral INTEGRAL
                        General n-dimensional integral to be solved (each
                        variable 'x' must be written in form of 'var(x)').
  -v VARIABLE [VARIABLE ...], --variable VARIABLE [VARIABLE ...]
                        Integral boundaries and step for each variable in form
                        of 'x=lower:step:upper'. The lower and upper
                        boundaries and the step can be either constant or
                        constant function. Due to some limitations of the
                        underlaying program, the minimal step for the
                        numerical method is '1e-7'.
  -p PRECISION, --precision PRECISION
```

```
                      Precision of the numerical method (default 1e-20).
  -d DIGITS, --digits DIGITS
                      Number of displayed decimal digits (default 20).
  --verbose           Enable verbose output (disabled by default).

Copyright 2014 @ Jiri Mikulka <jiri.mikulka@gmail.com>
```

Parametr `-i` a `--integral`

Z nápovědy k aplikaci je zřejmé, že parametr `-i` (nebo `--integral`) je povinný. Tento parametr přijímá řetězec obsahující integrand jako argument. Pro přehlednost musí být každý výskyt integrální proměnné x uveden ve formátu `var(x)`. V integrandu je možné použít všechny operátory a funkce, které jsou podporovány aktuální verzí `TKSL` – základní matematické operátory $+, -, \cdot, /$, funkce ln(), exp() a goniometrické funkce sin(), cos(), tan(), cotan() včetně jejich cyklometrických, hyperbolických a hyperbolometrických variant. Stávající verze `TKSL` nepodporuje operátor `'^'`, proto je mocnina nahrazena zápisem `pow[základ, exponent]`; budoucí verze `TKSL` by však operátor mocniny měla podporovat.

význam	notace
součet	op + op
rozdíl	op − op
součin	op ∗ op
podíl	op/op
mocnina	pow[základ, exponent]
sinus	sin(arg)
kosinus	cos(arg)
tangens	tan(arg)
kotangens	cotan(arg)
arkus sinus	asin(arg)
arkus kosinus	acos(arg)
arkus tangens	atan(arg)
arkus kotangens	acotan(arg)
hyperbolický sinus	sinh(arg)
hyperbolický kosinus	cosh(arg)
hyperbolický tangens	tanh(arg)
hyperbolický kotangens	cotanh(arg)
argument hyperbolického sinu	asinh(arg)
argument hyperbolického kosinu	acosh(arg)
argument hyperbolického tangens	atanh(arg)
argument hyperbolického kotangens	acotanh(arg)
přirozený logaritmus	ln(arg)
exponenciální funkce	exp(arg)

Tabulka A.2: Přehled podporovaných operátorů a funkcí včetně jejich notací (převzato z [9]).

Parametr `-v` a `--variable`

Dalším povinným parametrem je `-v` (nebo `--variable`). Tento parametr má povinně alespoň jeden argument ve tvaru `"var=min:step:max"`, kde `var` musí být integrální proměnná a `min`, `step`, `max` jsou konstantní funkce (např. horní mez lze zadat ve tvaru $1 + \sin(1)$). V těchto výrazech je možné použít konstantu π (notace `pi, PI, pI, Pi`). Nemá-li integrand ani jednu integrální proměnnou (tzn. jedná se o konstantní funkci), pak je nutné zadat pravě jeden argument, u kterého nezáleží na názvu proměnné. Dále platí, že pro každou integrální proměnnou musejí být zadány dolní i horní meze a krok numerické metody.

Parametr `-p` a `--precision`

Tímto parametrem je možné aplikaci `hpni` zadat přesnost, se kterou má numerická metoda při výpočtu Taylorova polynomu a program `TKSL` počítat. Výchozí hodnota je nastavena na 10^{-20}, ale tuto hodnotu lze při spuštění přenastavit na libovolné reálné číslo.

Parametr `-d` a `--digits`

Parametrem `-d` (nebo `--digits`) lze nastavit počet desetinných míst, které se mají brát v úvahu při výpočtech a zobrazovat při tisku. Argument tohoto parametru musí být kladné celé číslo.

Parametr `--verbose`

Informační zprávy o průběhu výpočtu lze povolit parametrem `--verbose`, který je ve výchozím stavu vypnutý. Tyto zprávy umožní uživateli podrobně sledovat jednotlivé kroky výpočtu, provedené modifikace a restart výpočtu, pokud při stávajících nastaveních nebyl výpočet dostatečně přesný.

Příloha B

Příklad použití

Na několika příkladech bude demonstrován způsob spuštění a použití aplikace `hpni`.

Příklad 3. Vypočtěte určitý integrál

$$\int_0^1 \sin(1)\,\mathrm{d}t. \tag{B.1}$$

Řešení. V `hpni` bude tento příklad zapsán následovně (uvažujme 6 desetinných míst):

```
$ ./hpni.py -d 6 -i "sin(1)" -v "t=0:0.1:1"
Integrate(sin(1), (t, 0.000000, 1.000000)) = 0.841471
```

Příklad 4. Vypočtěte určitý integrál

$$\int_0^1 \sin(x)\,\mathrm{d}x. \tag{B.2}$$

Řešení. V `hpni` bude tento příklad zapsán následovně (uvažujme 6 desetinných míst):

```
$ ./hpni.py -d 6 -i "sin(var(x))" -v "x=0:0.1:1"
Integrate(sin(x), (x, 0.000000, 1.000000)) = 0.459698
```

Příklad 5. Vypočtěte určitý integrál

$$\int_0^1 \int_2^1 \sin(x) \cdot \sin(y)\,\mathrm{d}x\,\mathrm{d}y. \tag{B.3}$$

Řešení. V `hpni` bude tento příklad zapsán následovně (uvažujme 6 desetinných míst):

```
$ ./hpni.py -d 6 -i "sin(var(x))*sin(var(y))" -v "x=2:0.1:1" "y=0:0.1:1"
Integrate(sin(x)*sin(y), (x, 2.000000, 1.000000), (y, 0.000000, 1.000000)) = -0.429594
```

Příklad 6. Vypočtěte určitý integrál

$$\int_0^1 \int_1^2 \int_0^2 \sin(x) \cdot \sin(y) \cdot \cos(z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z. \tag{B.4}$$

Řešení. V `hpni` bude tento příklad zapsán následovně (uvažujme 6 desetinných míst):

```
$ ./hpni.py -d 6 -i "sin(var(x))*sin(var(y))*cos(var(z))" -v "x=0:0.1:2" "y=1:0.1:2" "z=0:0.1:1"
Integrate(sin(x)*sin(y)*cos(z), (x, 0.000000, 2.000000), (y, 1.000000, 2.000000),
(z, 0.000000, 1.000000)) = 1.146396
```