UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Výpočty určitých integrálů s podporou Matlabu

Jakub Vácha

Bakalářská práce
2024

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub Vácha**
Osobní číslo:	**I21101**
Studijní program:	**B0714P060001 Aplikovaná elektrotechnika**
Téma práce:	**Výpočty určitých integrálů s podporou Matlabu**
Zadávající katedra:	**Katedra elektrotechniky**

Zásady pro vypracování

Uvést základní definici pojmu určitého (Riemanova) integrálu funkce jedné reálné proměnné. Nastudovat a uvést základní numerické metody výpočtu určitého integrálu. Vypracovat M-skripty pro řešení určitého integrálu jednotlivými metodami v Matlabu. Vytvářet ilustrativní příklady pro řešení určitého integrálu popsaými metodami. Diskutovat přesnost výpočtu jednotlivámi metodami. Vybrat a uvést aplikace určitých integrálů.

Rozsah pracovní zprávy: **30**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Zahrádka, J. *Diskrétní matematika pro SII – diskretizační metody numerické matematiky*. Pardubice, 2014. ISBN 9788073958411.
2. Zahrádka, J. Matematický seminář – MATLAB. Pardubice: Univerzita Pardubice, 2013. ISBN 978-80--7395-691-2.

Vedoucí bakalářské práce: **RNDr. Jaromír Zahrádka, Ph.D.**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

doc. Ing. Jan Pidanič, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 24. ledna 2024

Prohlášení autora

Prohlašuji:
Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.
Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 9.5.2024 Jakub Vácha

Poděkování

Chtěl bych poděkovat vedoucímu bakalářské práce panu RNDr. Jaromírovi Zahrádkovi, Ph.D. za jeho rady a věnovaný čas. Dále bych chtěl poděkovat své rodině za podporu během studia.

Anotace

Tato bakalářská práce se zabývá převážně základními numerickými metodami pro výpočty určitých integrálů. Konkrétně jejich definice, příklady s ilustracemi a realizace v Matlabu pro explicitní i implicitní funkce. K jednotlivým metodám jsou také zahrnuty výpočty odhadů absolutních chyb. Nakonec práce srovnává mezi sebou jednotlivé numerické metody a implementované příkazy v Matlabu určené k numerické integraci.

Klíčová slova

určitý integrál, numerické metody, Matlab, obdélníková metoda, lichoběžníková metoda, Simpsonovo pravidlo, metoda Monte Carlo

Title

Calculations of definite integrals with Matlab support

Annotation

This bachelor thesis deals mainly with basic numerical methods for calculating definite integrals. In particular, their definition, examples with illustrations and implementation in Matlab for explicit and implicit functions. Calculations of absolute error estimates are also included for each method. Finally, the thesis compares the different numerical methods and the implemented commands in Matlab with each other.

Keywords

definite integral, numerical methods, Matlab, rectangle method, trapezoidal rule, Simpson's rule, Monte Carlo method

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Numerické metody pro výpočty určitých integrálů slouží k aproximaci číselných hodnot integrálů v případech, kdy integrovaná funkce/integrand nemůže být vůbec nebo s příliš velkými obtížemi vyjádřena ve tvaru primitivní funkce, anebo když je známý pouze omezený počet funkčních hodnot.

V úvodní kapitole je rámcově popsaná historie, která předcházela zrodům Newtonového určitého integrálu a Riemannova určitého integrálu, ze kterých dále vychází řešení některých příkladů v této bakalářské práci. Oba jsou potom v druhé kapitole podrobně definovány a doplněny o ilustrace. Třetí kapitola je věnována Matlabu, v němž jsou všechny výpočty, algoritmy a zobrazení grafů funkcí pro tuto práci realizované. V této kapitole jsou obecné informace o Matlabu nejen jako softwaru ale i jako programovacímu jazyku. Jsou zde popsány a na příkladech ukázány nezbytné matlabovské příkazy, které nějak souvisí s výpočtem určitých integrálů nebo tvorbou m-funkcí pro numerickou integraci. Čtvrtá hlavní kapitola se zabývá několika numerickými metodami pro výpočet určitých integrálů. Jmenovitě jsou to metody obdélníková, lichoběžníková, Simpsonova a Monte Carlo. K lichoběžníkové a Simpsonové se ještě vážou matlabovské příkazy `trapz` a `quad`, které jsou v práci též zahrnuty. Jednotlivé metody jsou popsány, demonstrovány, graficky interpretovány a hlavně naprogramovány v Matlabu jako m-funkce. Kromě řešení integrace explicitních funkcí je provedena také integrace implicitně zadané funkce. Pro explicitní funkce jsou navrženy dodatečně m-funkce pro výpočet odhadu absolutních chyb. Na závěr bakalářské práce jsou metody aplikovány a mezi sebou porovnány na několika matematických funkcích s různými průběhy.

1. Stručná historie

V první kapitole si zrychleně uvedeme, co předcházelo vzniku Newtonova a Riemannova integrálu. Zdrojem jsou první tři kapitoly z [1].

Jedna z motivací, která vedla ke vzniku integrálu bylo počítání obsahu a objemu. Od 8. tisíciletí před n. l. vznikala opevněná sídliště městského typu a začíná se obdělávat půda a pěstování zemědělských plodin. V 5. až 4. tisíciletí v oblasti velkých toků Eufratu, Tigridu a Nilu se staví zavodňovací díla, při kterých se začíná využívat zeměměřičských pomůcek a s tím spojené vyměřování zaplavených ploch, které vedlo k základům geometrie.

Nejvíce toho víme o egyptské a mezopotámské matematice, protože psali na hliněné destičky a papyrus, které mají relativně velkou trvanlivost. Naproti tomu v Číně a v Indii se psalo na kůru a bambus, které rychleji podléhaly zkáze.

Egypťané počítali obsah plochy tak, že danou plochu rozdělili na trojúhelníky, u kterých spočítali jejich obsahy a ty potom sečetli. V Řecku byla matematika podobná té egyptské, tedy hlavně konkrétní a praktická. Začátkem 6. stol. před n. l. ale začali Řekové rozvíjet teoretickou matematiku, která dávala odpovědi na otázku „proč?“ v kontrastu k matematice praktické, která řeší jen způsob.

Za předchůdkyni infinitezimálních úvah je považovaná *Exhaustivní* (vyčerpávající) *metoda,* za kterou stojí **Eudoxos**. Metoda umožňuje už poměrně přesné výpočty obsahů i objemů a je založena na nekonečném dělení. Na tuhle metodu navázal **Euklides** ve svém díle, které je zároveň nejstarší zcela zachovalé dílo řecké matematiky. Jedná se o Eukleidovy *Základy* složené z 13 knih, ve kterých jsou shrnuta většina poznatků v oblasti matematiky z té doby. Euklides patří mezi nejvýznamnější matematiky. Žil kolem 3. století před n. l. Jeho výklad vychází ze soustavy definic, postulátu a axiomu. Ve 12. knize píše o *Exhaustivní metodě,* kde dokazuje různé vztahy mezi objemy těles. Nikde ale nevypočítává obsahy nebo objemy daných těles, protože v té době nepatřily do teoretické geometrie, nýbrž do geometrie praktické. Šlo například o tato tvrzení:

Obsahy kruhů jsou ve stejném poměru jako kvadráty jejich poloměrů:

$$S_1 : S_2 = r_1^2 : r_2^2$$

Objemy jehlanů o stejné výšce jsou ve stejném poměru jako obsahy jejich podstav:

$$V_1 : V_2 = A_1 : A_2$$

Na *Exhaustivní metodu* navázal další velmi významný matematik **Archimédes** (3. století před n. l.). Proslul nejen matematikou ale i technickými vynálezy.

Exhaustivní metodu rozvíjí v jeho pracích: *Měření kruhu, Kvadratura paraboly, O kouli a válci, O spirálách, O konoidech* a *sféroidech*. Archimedes tuto metodu aplikoval na řadu problémů, které jsou dnes řešeny integrálním počtem. Začal do matematiky více vkládat proměnné. Archimedovy práce a práce jeho nástupce **Apollónia z Pergy** (asi 260–170 před

n. l.), který proslul svým dílem *Kuželosečky*, měly široký vliv na další rozvoj matematiky a v dobách novověku se staly východiskem pro vznik *matematické analýzy*. Starořecká matematika dosáhla vysoké úrovně, ale pozdější období římské nadvlády přineslo stagnaci a úpadek celé vědy. Kolem roku 150 n. l. vzniklo ale obsáhlé dílo **Ptolemaiova** *Velká skladba* (*Almagest*), ve které se zabývá i trigonometrií.

Po rozpadu římské říše, a tedy koncem starověku se matematika téměř nerozvíjela. Situace se zlepšila až po 12. století, kdy došlo k celkovému rozvoji vědy a kultury. Do latiny se přeložilo spoustu vědeckých prací, mezi kterými byly i práce Eukleida, Archiméda a Apollónia, které stály za rozvojem evropské matematiky. Začátkem 16. století matematika překračuje rámec znalostí ze starověkého Řecka. Začínalo období matematiky *proměnných veličin, symbolické algebry, analytické geometrie* a *integro-diferenciálního počtu*. Ke vzniku moderních integračních metod přispěl velmi **Johan Kepler** (1571–1630), který při svých výpočtech rozděloval tělesa na nekonečně malé kusy. Známé je jeho rozdělení kruhu na nekonečno rovnoramenných trojúhelníků s vrcholem ve středu kruhu pro výpočet obsahu. Na rozdíl od Archiméda se ale moc nezaobíral důkazy. Dalším matematikem, ze kterého vycházeli další matematici, byl žák astronoma **Galilea Galileii** – **Bonaventura Cavalieri** (1598–1647). Od něho pochází tzv. „*Cavalieriho princip*“. Pro výpočet obsahu/objemu porovnával mezi sebou dvě tělesa, u kterých porovnával nekonečně malé části těchto těles. Galileo došel například k *neurčitému integrálu* z výpočtu dráhy:

$$x = \frac{1}{2}gt^2 \text{ když } \frac{dx}{dt} = gt$$

Významnějším problémem se stalo určování tečen v daném bodě na křivce. Jedni k tomuto problému přistupovali skrz geometrii – **Torricelli** a **Isaac Barrow**. Druzí na to šli přes algebru – **Fermat**, **Descartes** a **Wallis**. Fermat dovedl integrovat funkce parabolické, hyperbolické a exponenciální. **Pascal** jako první trigonometrické. Nejvýznamnější matematici z druhé poloviny 17. století se stali **Isaac Newton** a **Gottfried Wilhelm Leibniz**, kteří nezávisle na sobě vytvořili *integrální a diferenciální počet*. Sjednotili *infinitezimální počet* a dali mu výpočetní algoritmy. Newton přitom vycházel ze své *teorie fluxí a fluent*, která vychází z toho, že integrování je inverzní operace k diferencování. Vypracoval tabulky integrálu a založil *substituční metodu*. Leibniz zase *metodu per partés*. Leibniz zavedl operační symbol pro integrování, ale samotný název integrál pochází od **Jakoba Bernoulliho**. Leibniz se více snažil tvořit obecné metody a na rozdíl od Newtona dbal na notaci. Vznikly pojmy jako *funkce, proměnná, konstanta* a *parametr*. Z této doby vzešel fundamentální vztah:

$$\int_a^b f(x)dx = F(b) - F(a)$$

Kde F :(a,b)→ $\mathbb{R}$ je funkce primitivní k funkci f na intervalu (a,b).

Problémem bylo, zda infinitezimální veličiny skutečně existují. Mezi matematiky pak propukla silná polemika. Někteří pracovali jen s konečnými přírůstky Δx . Jiní se snažili o definici spojitosti pomocí limit. Krok k vyřešení udělal **Alembert** s touto definicí:

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$$

V 18. století se matematická analýza používala k řešení problému z fyziky a astronomie. Spousta matematiků byla zároveň i fyziky. Dalším velkým matematikem té doby byl **Leonhard Euler** (1707–1783) S obrovským množstvím děl přispěl k řadě odvětví jako třeba *kombinatorika, diferenciální rovnice* a *teorie čísel*. Zavádí známé Eulerovo číslo e vypočteno na 23 des. míst a nekonečnou veličinu jako:

$$\frac{a}{dx} = \infty$$

S další definicí integrálu přišel v roce 1823 **Augustin-Louis Cauchy** jako:

$$S = \sum_{i=1}^{n} f(x_{i-1})(x_i - x_{i-1})$$

pro interval $\langle a, b \rangle$ rozdělen na n částí.

S tím související pojem *levý*, respektive *pravý Cauchyův integrál*.

Nyní se konečně dostáváme k *Riemannovému integrálu*, jehož autorem je **Georg Friedrich Bernhard Riemann** (1826–1866). Výhoda jeho integrálu oproti Cauchyovému je, že integrovaná funkce nemusí být všude spojitá. Oba se vrátili se svými definicemi integrálu ke staré řecké *exhaustivní metodě*.

2. Určitý integrál

Zde se zaměříme na definice a ilustrace Newtonova a Riemannova integrálu a jejich aplikaci na několika příkladech. Kromě těchto dvou existují i další a mladší definice určitých integrálů s širším využitím. Určitý integrál jednoduše řečeno počítá plochu pod křivkou v daném intervalu pro danou funkci. V místě, kde se křivka drží nad horizontální osou x se hodnota plochy přičítá. V místě, kde je křivka pod osou se naopak odečítá. Takže integrál například jedné periody funkce sinus je 0 (stejná plocha nad i pod osou). Matematické značení určitého integrálu funkce f je $\int_a^b f(x)dx$

Kde $\int$ je značka integrálu. a je dolní mez integrálu a b je horní mez integrálu – část osy x po které se bude integrál počítat. $f(x)$ je integrovaná funkce neboli integrand a dx je diferenciál, který zároveň znázorňuje, podle které proměnné se integruje.

2.1 Newtonův integrál

Začneme Newtonovým integrálem, který vzniknul dříve.

2.1.1 Definice Newtonova integrálu na uzavřeném intervalu

Uvažujme funkci f definovanou a spojitou na uzavřeném intervalu $\langle a, b \rangle \in \mathbb{R}$, k níž existuje primitivní funkce F (pro kterou platí $F'(x) = f(x)$) na intervalu $\langle a, b \rangle$. Rozdíl funkčních hodnot primitivní funkce F v počátečním a koncovém bodě intervalu se nazývá Newtonův integrál na intervalu $\langle a, b \rangle$

$$\int_a^b f(x)dx = [F(x)]_a^b = F(b) - F(a)$$

Pro spojité funkce na intervalu $\langle a, b \rangle$ platí, že Newtonův a Riemannův integrál se na tomto intervalu rovnají. Newtonův integrál takto definovaný totiž nelze počítat z funkce, k níž neexistuje primitivní funkce. [2]

2.1.2 Definice Newtonového integrálu na otevřeném intervalu

Pro funkci f spojitou na otevřeném intervalu (a, b), k níž existuje primitivní funkce F a lze funkci dodefinovat na $\langle a, b \rangle$ tak, že $f(a) = \lim_{x \to a^+} f(x) \in \mathbb{R}$ a $f(b) = \lim_{x \to b^-} f(x) \in \mathbb{R}$. Potom Newtonův integrál funkce f na intervalu (a, b) je $\int_a^b f(x)dx = [F(x)]_a^b = \lim_{x \to b^-} F(x) - \lim_{x \to a^+} F(x)$.

[3]

2.2 Riemannův integrál

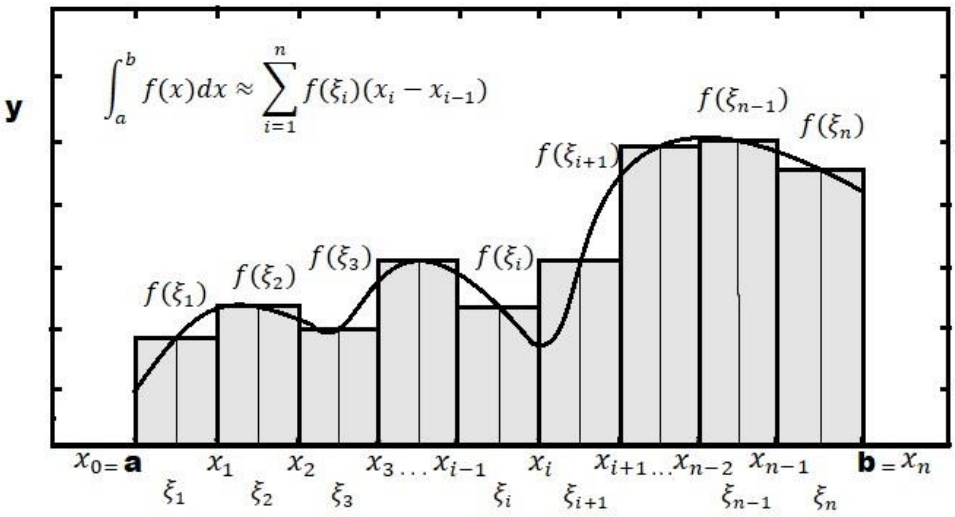
S ekvivalentní definicí k Riemannově definici přišel několik let poté ještě Gaston Darboux (1842–1917) [1]. Uvedeme si tedy obě definice.

2.2.1 Riemannova definice

Uvažujme funkci f definovanou na omezeném intervalu $\langle a, b \rangle \in \mathbb{R}$ a posloupnost dělení $\{D_n\}$ uzavřeného intervalu $\langle a, b \rangle$, kde každé dělení D_n je určeno vektorem $n + 1$ čísel $x_0, x_1, x_2 \dots, x_n$ takových že $a = x_0 < x_1 < x_2 < \dots < x_n = b$. Předpokládejme, že $\{D_n\}$ je posloupnost dělení, jejichž norma $v(D_n) = \max_{i \in \{1,\dots,n\}} (x_i - x_{i-1})$ konverguje k nule, tj. $\lim_{n \to \infty} v(D_n) = 0$. Pro každé dělení $\{D_n\}$ intervalu $\langle a, b \rangle$, je integrálním součtem hodnota $\sigma(D_n) = \sum_{i=1}^n f(\xi_i)(x_i - x_{i-1})$, kde $x_{i-1} \leq \xi_i \leq x_i$. Jestliže existuje limita posloupnosti integrálních součtů $\lim_{n \to \infty} \sigma(D_n)$ nezávisle na volbě posloupnosti dělení $\{D_n\}$ a nezávisle na volbě bodů ξ_i, pak se tato limita nazývá **Riemannův integrál** na intervalu $\langle a, b \rangle$. Píšeme:

$$\int_a^b f(x)\ dx = \lim_{n \to \infty} \sum_{i=1}^n f(\xi_i)(x_i - x_{i-1})$$

[2]

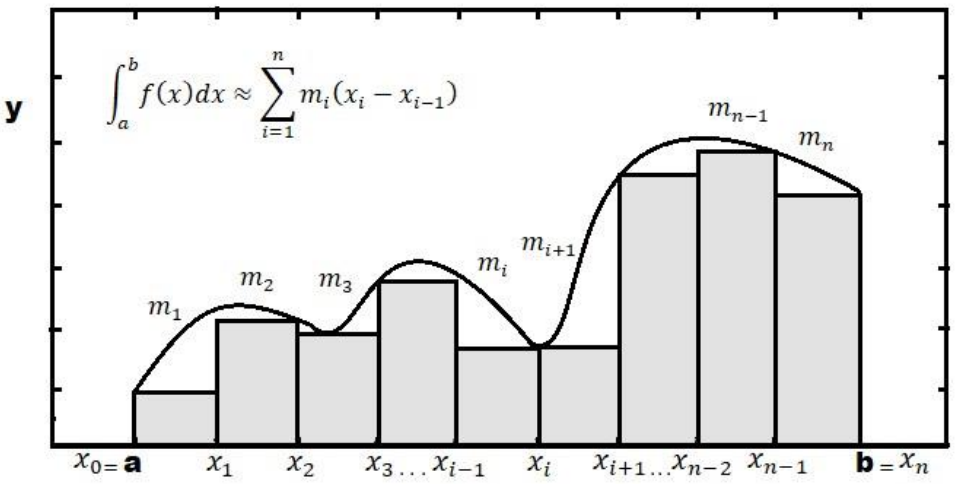


Obrázek 1 – Ilustrace Riemannova integrálu podle Riemannovy definice

2.2.2 Darbouxova definice

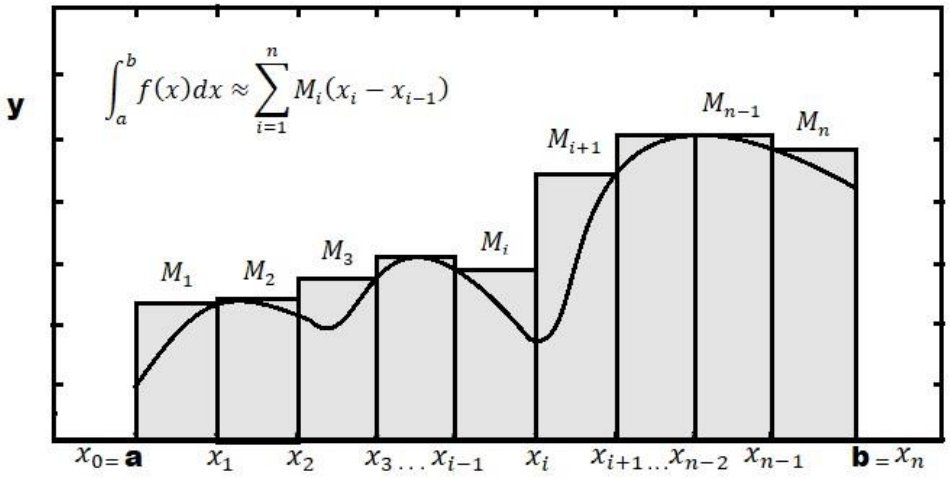
Obdobně jako u Riemannovy definice uvažujme funkci f definovanou na omezeném intervalu $\langle a, b \rangle \in \mathbb{R}$ a posloupnost dělení $\{D_n\}$ uzavřeného intervalu $\langle a, b \rangle$, kde každé dělení D_n je určeno vektorem $n+1$ čísel $x_0, x_1, x_2, \ldots, x_n$ takových, že $a = x_0 < x_1 < x_2 < \ldots < x_n = b$. Pro $\{D_n\}$ mějme normu dělení $v(D_n) = \max_{i \in \{1,\ldots,n\}} (x_i - x_{i-1})$ konvergující k nule.

Pro dolní integrální součet z této funkce f zavedeme $m_i = \inf_{x \in \langle x_{i-1}, x_i \rangle} f(x)$ pro každé i={1,2,…,n} po intervalu $\langle a, b \rangle$. Potom dolní integrální součet na intervalu $\langle a, b \rangle$ při dělení D je $L(f, D) = \sum_{i=1}^{n} m_i(x_i - x_{i-1})$.



Obrázek 2 – Dolní integrální součet

Pro horní integrální součet z této funkce f zavedeme $M_i = \sup_{x \in \langle x_{i-1}, x_i \rangle} f(x)$ pro každé $i = \{1,2,\dots,n\}$ po intervalu $\langle a, b \rangle$. Potom horní integrální součet na intervalu $\langle a, b \rangle$ při dělení D je $U(f,D) = \sum_{i=1}^{n} M_i(x_i - x_{i-1})$.



Obrázek 3 – Horní integrální součet

Platí $L(f,D) \leq U(f,D)$ a lze tedy očekávat, že skutečná hodnota Riemannova integrálu se bude nacházet mezi $L(f,D)$ a $U(f,D)$. Za předpokladu, že Ψ je množina všech možných dělení intervalu $\langle a, b \rangle$ lze definovat:

Dolní Riemannův integrál funkce f na intervalu $\langle a, b \rangle$ je supremum všech možných dolních integrálních součtu z Ψ.

$$\int_{\underline{a}}^{b} f(x)dx = \sup L\,(f,D)_{D \in \psi}$$

Horní Riemannův integrál funkce f na intervalu $\langle a, b \rangle$ je infimum všech možných horních integrálních součtu z Ψ.

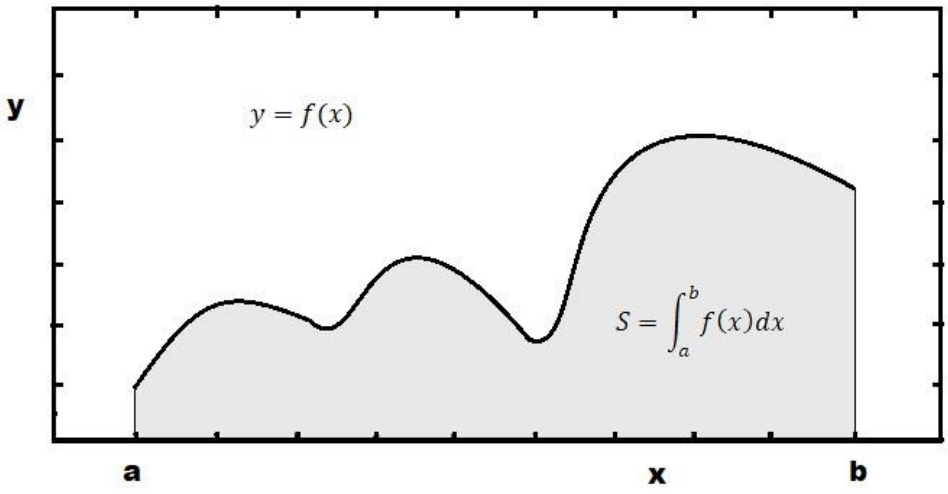
$$\int_{a}^{\overline{b}} f(x)dx = \inf U\,(f,D)_{D \in \Psi}$$

Pokud je funkce f omezená na intervalu $\langle a, b \rangle$ platí: $\int_{\underline{a}}^{b} f(x)dx \leq \int_{a}^{\overline{b}} f(x)dx$

Za předpokladu, že $\int_{\underline{a}}^{b} f(x)dx = \int_{a}^{\overline{b}} f(x)dx$ lze konečně definovat Riemannův integrál funkce f na intervalu $\langle a, b \rangle$ podle Darbouxovy definice takto:

$$\int_{a}^{b} f(x)dx = \int_{\underline{a}}^{b} f(x)dx = \int_{a}^{\overline{b}} f(x)dx$$

[2] [3]



Obrázek 4 – Ilustrace Riemannova integrálu podle Darbouxovy definice

Dodatek: Existují funkce, které jsou integrovatelné pouze v jednom smyslu. Tedy pouze v Newtonově nebo Riemannově smyslu. Později vznikly ještě další definice integrálu – Lebesgueův integrál a Perronův integrál. [1]

2.3 Aplikace určitého integrálu

Určité integrály lze aplikovat v technických oborech, fyzice, chemii, ekonomii, statistice a při výpočtu pravděpodobnosti.

Pokud známe primitivní funkci F k funkci f, můžeme tímto způsobem v praxi počítat třeba vykonanou práci W po dráze x a sílou f jako $W = \int_a^b f(x)dx$.

Dále přenesený elektrický náboj Q podle proudu I za čas t jako $Q = \int_0^t I(t)dt$.

Díky určitému integrálu a znalosti speciálního vzorce lze počítat i objem rotačního tělesa, který vznikne rotací grafu funkce $f(x)$ kolem osy x $V = \pi \int_a^b [f(x)]^2 \, dx$.

Křivka, která je grafem $f(x)$ se obrazně otočí kolem osy x a tím vytvoří 3D těleso, u kterého zjistíme objem. Podobně lze počítat i obsah rotační plochy $S = 2\pi \int_a^b f(x)\sqrt{1 + [f'(x)]^2} \, dx$.

Délku křivky, která je grafem funkce $f(x)$ pro $x \in a, b$, lze počítat podle vztahu $L = \int_a^b \sqrt{1 + [f'(x)]^2} dx$. [7]

3. Matlab

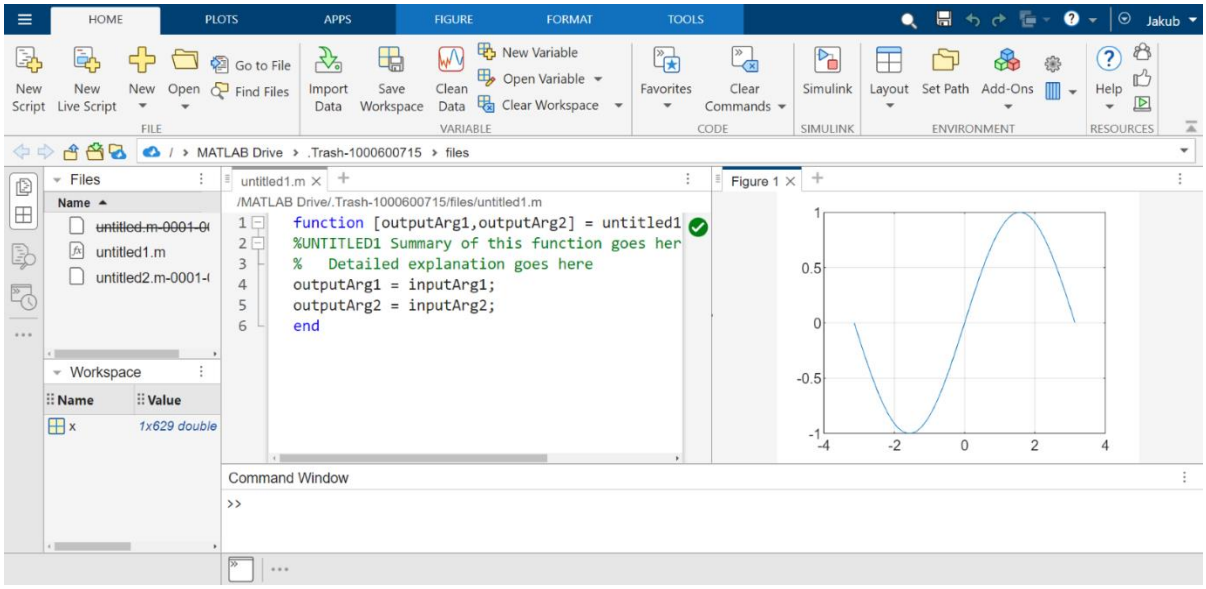
K vytváření našich programů pro výpočty určitých integrálů, budeme využívat online platformu Matlab od firmy Mathworks. V této kapitole shrneme základní informace pro orientaci a práci v Matlabu, které se budou hodit v následující hlavní kapitole.

3.1 Informace k Matlab

Matlab (jako zkratka Matrix Laboratory, přeloženo maticová laboratoř) je programovací jazyk a zároveň programové prostředí od americké firmy MathWorks. Používá se například ke složitým výpočtům, vykreslování 2D nebo 3D grafu funkcí, tvorbě algoritmu a simulacím. Matlab používají především vědci, technici, studenti a zaměstnanci vysokých škol v technických a ekonomických oborech.

Za počátkem Matlabu stojí matematik a programátor Cleve Moler, který učil matematiku na Univerzitě v Novém Mexiku. Moler vyvíjel Matlab pro své studenty v rámci koníčku. Po letech vývoje byl Matlab otevřen veřejnosti poprvé v roce 1979. [5]

V Matlabu jsou všechny proměnné konstruované jako pole (array). Na rozdíl od některých programovacích jazyků, se v Matlabu pole indexují od 1 místo 0. Při inicializaci většiny proměnných není většinou nutné deklarovat jejich datový typ. Ten může být v průběhu programu přepsán na jiný typ. Nechybí samozřejmě ani podmínkové příkazy jako jsou `if`, `while`, `for`. Matlab má mnoho implementovaných funkcí/příkazů pro operace s maticemi, mocniny/odmocniny, integrace/derivace, goniometrické funkce, logické operace, statistické veličiny nebo Fourierova transformace. Na stránkách Matlabu nebo příkazem *help*, můžeme najít široký obsah návodů, videonávodů a ukázkových kódů. Nechybí ani diskuzní fórum, kde se uživatelé mezi sebou radí.



Obrázek 5 – Uživatelské rozhraní Matlab

V další části zmíníme okna uživatelského rozhraní, která budeme při naši další práci využívat:

Files – seznam vytvořených skript nebo funkcí

Workspace – seznam našich proměnných a jejich parametry (hodnota,velikost a třída)

Command Window – okno pro zadávání přímých příkazu (okamžitě se vyhodnotí)

Figure – již zobrazen graf matematické funkce

Zde budeme moci sledovat grafy našich matematických funkcí.

untitled1.m – nově otevřená matlabovská funkce, do které můžeme psát nový program

Funkce mají vstupní a výstupní parametry.

3.2 Proměnné

Kromě klasických číselných proměnných existují v Matlabu ještě tzv. symbolické proměnné. Je to poměrně neobvyklý datový typ mezi programovacími jazyky. Rozdíl mezi symbolickou proměnnou a běžnou číselnou proměnnou je, že nemá stálou určitou hodnotu, ale zastupuje nějakou množinu reálných nebo komplexních čísel. Používají se obecně tam, kde se pracuje s matematickými funkcemi nebo s rovnicemi. Při vytváření symbolických proměnných je nutné deklarovat jejich typ, na rozdíl od číselných proměnných. S oběma typy lze provádět všechny běžné matematické operace. Proměnné mohou mít různé rozměry. Proměnnou s rozměrem 1×1 nazýváme skalár. Je-li rozměr proměnné $1 \times N$, resp. $N \times 1$ (kde $N > 1$) jedná se o řádkový , resp. sloupcový vektor. Matice typu $M \times N$ má M řádku a N sloupců. [6]

3.2.1 Tvorba proměnných

Ukážeme si tvorbu obou typů proměnných. Veškeré příkazy píšeme do *Command Window*. Pokud za příkazem napíšeme středník, tak se příkaz vyhodnotí bez dalších zpráv. Například při vytváření náhodné matice 100×100 by se nám bez napsání středníků vypsaly hodnoty všech prvků matice do *Command Window*. Všechny vzniklé proměnné se nám ukládají do *Workspace*. Značka >> označuje námi napsaný řádek. Řádek bez této značky potom značí text vygenerovaný samotným Matlabem. Při vytváření číselných proměnných můžeme postupovat následovně.

Vytváření skaláru:

```
>> a=5;
>> bu=8^a;
```

Vytváření vektorů a matic:

```
>> i=[4 7 9];
```

řádkový vektor o třech složkách

```
>> ye=[2;3;4];
```

sloupcový vektor, středník odděluje řádky v matici

```
>> t=0:2:10;
```

vektor od 0 do 10 s krokem 2

```
>> t=linspace(0,10,6);
```

vektor od 0 do 10 s 6 prvky (stejný výsledek jako předchozí příkaz)

```
>> t2=5*t;
```

proměnná t2 bude mít stejný rozměr jako proměnná t

```
>> z=[1 2; 3 4; 5 6];
```

matice se třemi řádky a dvěma sloupci

S maticemi lze provádět všechny běžné matematické operace (sčítání, odečítání, násobení, dělení, transpozice, inverze a řešení soustavy lineárních rovnic).

Vytváření symbolických proměnných:

```
>> x=sym('x');
```

v této formě *x* reprezentuje komplexní čísla

```
>> syms x y;
```

takto lze vytvořit více symbolických proměnných najednou

Pomocí *sym(' ')* můžeme symbolickou proměnnou přiřadit k proměnné s jiným názvem v závorce. Vystačíme si povětšinou s variantou *syms*.

```
>> t=sym('t', 'real');
>> syms t real;
```

oba příkazy vytvoří proměnnou *t*, která reprezentuje jen reálná čísla

```
>> y=sym('y', 'positive');
>> syms y positive;
```

podobně lze vytvořit symbolické proměnné, které reprezentují jen kladná čísla

```
>> syms r [1 3];
```

symbolická proměnná jako vektor

Jakmile máme vytvořenou jednu symbolickou proměnnou, tak další symbolické proměnné můžeme vytvářet i tak, že napíšeme výraz, který obsahuje už vytvořenou symbolickou proměnnou a další proměnná se tak automaticky vytvoří jako symbolická. Například takto:

```
>> q=sqrt(x);
```

3.2.2 Příklady se symbolickými proměnnými

Rovnice a nerovnice

Pomocí příkazu *solve(expr)* lze řešit symbolické rovnice v anulovaném tvaru, tedy *expr=0*. Pokud nemáme anulovanou rovnici, použijeme *solve(exprL==exprR)* kde *exprL* je levá strana a *expR* pravá strana rovnice. Pokud máme rovnici s více symbolickými proměnnými, tak musíme přidat parametr *var*, kterým určíme proměnnou, pro kterou se má rovnice řešit *solve(expr,var)*. Soustavu N anulovaných rovnic řešíme příkazem *solve(expr1,...,exprN,var1,...,varN)*. Parametry *var* jsou symbolické proměnné, pro které se soustava rovnic řeší. Výstupem je vektor o délce N s nalezenými neznámými. V případě nerovnic pouze znak == nahradíme < nebo >. V našem případě příkaz *solve* využijeme dále při řešení a integrací implicitních funkcí. [6]

Řešení kvadratické rovnice $x^2 - 4x + 13 = 0$ v oboru komplexních čísel.

```
>> syms x
>> solve(x^2-4*x+13)
ans=
2 - 3i
2 + 3i
```

V případě, že by *x* reprezentovalo reálná čísla (*>> syms x real*) tak by rovnice neměla řešení.

Ukázka řešení soustavy dvou lineárních rovnic $x + 2 * y = 5$ a $2 * x - y = 5$ o dvou neznámých.

```
>> syms x y
>> solve(x+2*y == 5, 2*x-y == 5, x, y)
ans = struct with fields:
x: 3
y: 1
```

Limita funkce

Limita funkce jedné reálné proměnné se v Matlabu řeší příkazem *limit(expr,x,a,'left/right')*. Kde *expr* je vyšetřovaná funkce se symbolickou proměnnou *x*, která se blíží k *a*. Jako poslední argument *'left/right'* můžeme ještě uvést z

jaké strany se má limita počítat. Pro výpočet limity zprava $\lim_{x\to a^+} f(x)$ zadáme argument `'right'`, nebo zleva $\lim_{x\to a^-} f(x)$ argument `'left'`.

Pro následující funkci `f` vypočítáme limitu třemi zmíněnými způsoby.

```
>> syms x
>> f=(x^2+4*x+5)/(x^3+8);
>> limit(f, x, -2)
ans = NaN
```

Tedy $\lim_{x\to -2} \frac{x^2+4x+5}{x^3+8}$ neexistuje.

```
>> limit(f, x, -2, 'right')
ans =Inf
```

Tedy $\lim_{x\to -2^+} \frac{x^2+4x+5}{x^3+8} = +\infty$.

```
>> limit(f, x, -2, 'left')
ans =-Inf
```

Tedy $\lim_{x\to -2^-} \frac{x^2+4x+5}{x^3+8} = -\infty$.

[6]

Derivování funkce

Pomocí derivací můžeme u funkcí vyšetřovat monotonii, tedy jestli funkce je na daném intervalu rostoucí, klesající, nerostoucí nebo neklesající. Dále pak z druhé derivace konkávnost, konvexnost a extrémy funkce. Vyšší derivace pak k vytváření Taylorových polynomů, které nám aproximují funkci kolem daného bodu. Derivace bude dále použita v hlavní kapitole 4., kde derivaci budeme potřebovat pro výpočet odhadů chyb jednotlivých numerických metod. K derivování nám poslouží příkaz `diff(expr,x,n)`, kde parametr `expr` je derivovaná funkce, `x` je proměnná podle které derivujeme a `n` je řád derivace. V případě první derivace můžeme parametr `n` vynechat. Ještě si uvedeme jeden důležitý příkaz, který se často používá v kombinaci s příkazem `diff`. Tím je tzv. symbolická substituce, která symbolickou proměnnou ve funkci nahradí novou proměnnou nebo výrazem. Syntax je `subs(expr,old,new)`, kde `expr` je funkce ve které se symbolická proměnná `old` nahradí proměnnou nebo výrazem `new`.

U následující funkce *f* budeme zjišťovat monotónnost v bodech $xa = -2, xb = 1$ *a* $xc = 6$.

```
>> syms x
>> f=cos(sqrt(x+8)-sqrt(10-x));
>> fd=diff(f,x)
fd =-sin((x + 8)^(1/2) - (10 - x)^(1/2))*(1/(2*(x + 8)^(1/2)) + 1/(2*(10 -
x)^(1/2)))
>> xa=-2; xb=1; xc=6;
>> fda=subs(fd,x,xa)
```

Dosadíme *xa* za *x* do již derivované funkce *fd*.

```
fda = -sin(6^(1/2) - 12^(1/2))*(6^(1/2)/12 + 12^(1/2)/24)
>> double(fda)
```

Hodnotu *fda* takto převedeme na desetinné číslo.

```
ans = 0.2959
```

Funkce je v bodě *xa* stoupající.

```
>> fdb=subs(fd,x,xb)
fdb = 0
```

Funkce je v bodě *xb* stacionární.

```
>> fdc=subs(fd,x,xc)
fdc = -sin(14^(1/2) - 2)*(14^(1/2)/28 + 1/4)
>> double(fdc)
ans = -0.3780
```

Funkce je v bodě *xc* klesající.
[6]

Integrování funkce

Matlab již obsahuje několik příkazů pro výpočet vlastních i nevlastních integrálů. Dokonce pro dvojné a trojné integrály. V této části si uvedeme příkaz, který počítá integrály pomocí primitivních funkcí. Jedná se o příkaz *int* [2]. Ve 4. kapitole již budeme řešit výpočty integrálů numerickými metodami. Nejkomplexnější tvar tohoto příkazu je *int(expr,var,a,b)*, kde *expr* je integrovaná funkce/integrand a *var* je integrovaná proměnná – u neparametrických funkcí s jednou proměnnou není potřeba zadávat. *a,b* je interval na kterém se má integrovat. Pokud chceme počítat neurčitý integrál, tak argumenty *a,b* vynecháváme. Pokud chceme počítat nevlastní integrál od $-\infty$ do ∞ píšeme za *a* *-inf* a za *b* *inf* . Vzhledem k tomu, že Matlab často hodnotu integrálu nevypočítá přímo, ale vrátí výsledek ve složeném algebraickém tvaru, tak se nám bude ještě hodit příkaz *double(ans)*, který obecně převádí výraz *ans* na desetinné číslo. V případě, kdy integrujeme pomocí příkazu *int* funkci, kterou ale nelze řešit standartně pomocí primitivních funkcí, tak výsledek zadáme jako argument do příkazu *double* a integrál se tímto spočítá numericky. Podobný příkaz je *vpa(ans,n)*, který výraz *ans* vypočítá na *n* platných číslic. Pokud integrál nejde vůbec řešit, vypíše se nám výsledek *NaN* (not a number).

Spočítáme délku křivky danou funkcí $f(x) = e^{\frac{x}{3}} + 2\sin^2 x$ od 0 do π pomocí již dříve zmiňovaného vzorce $L = \int_a^b \sqrt{1 + [f'(x)]^2}dx$.

```
>> syms x;
>> f=exp(x/3)+2*sin(x)^2;
```

Příkaz `exp(x)` značí mocninu `x` z eulerova čísla e^x.

```
>> L=int(sqrt(1+diff(f)^2),0,pi)
L = int(((exp(x/3)/3 + 4*cos(x)*sin(x))^2 + 1)^(1/2), x, 0, pi)
>> double(L)
```

Výsledek se tímto převede na reálné číslo.

```
ans = 5.1734
>> vpa(L,9)
```

Délka křivky spočtena na 9 platných číslic.

```
ans = 5.17344049
```

[6]

3.3 Grafické zobrazování funkcí

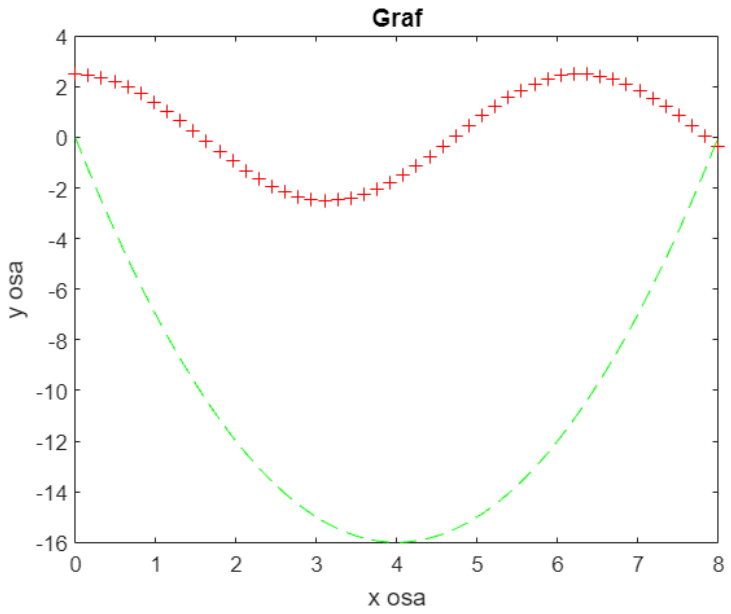
Existuje několik způsobů a forem, jak vizualizovat v Matlabu různá data nebo matematické funkce. Kromě pro nás nejdůležitějších klasických zobrazení funkcí o jedné proměnné, můžeme dále v Matlabu zobrazovat data ve formě koláčových, sloupcových, polárních grafů, histogramů, kontur nebo vektorových polí. Pro všechna tato zobrazení lze měnit jejich parametry jako barva a styl křivek nebo jiných předmětů v závislosti na formě použitého zobrazení. Můžeme taky přidávat popisky a legendy pro větší přehlednost. Tyto a několik dalších forem grafu najdeme v Matlabu ve vrchní záložce *PLOTS*, kde vybereme formu vizualizace a proměnné ze kterých se graf má sestrojit. V záložce *FIGURE* potom můžeme manuálně upravovat už vytvořené grafy bez nutnosti znalostí příkazů. Informace k této podkapitole jsou čerpány z *HelpCenter* [8].

3.3.1 Příkaz plot

Začneme příkazem `plot(x1,y1,linespec1,...,xN,yN,linespecN)`, který nám zobrazí 2D graf s proměnnou `x` a k tomu odpovídající proměnnou `y`, které jsou nejčastěji ve formě vektorů a mají stejný rozměr. Za parametr `linespec` lze potom dosadit až tři argumenty, které nám v grafu upraví styl, barvu křivky a zvýrazní body v grafu nějakým symbolem. Pro změnu stylu křivky použijeme například `"--"` pro přerušovanou křivku nebo `":"` pro tečkovanou křivku. Dále k zvýrazněním bodů v grafu třeba `"o"`, který vytvoří kruhy kolem bodů na křivce nebo `"+"` který zas znaménka plus. Nakonec ještě lze měnit barvu křivky na červenou `"red"` nebo třeba modrou `"blue"`. Pokud jeden z argumentů vynecháme, tak daný parametr zůstane původní. Kompletní seznam konfigurací najdeme na stránkách *HelpCenter* do kterých se lze dostat i příkazem `help` zapsaným do *Command Window*. Do jednoho grafu lze zavést více než jednu křivku a její parametry. Ke grafu lze potom přidávat popisky k jednotlivým osám příkazy `xlabel('xname')` pro x osu a `ylabel('ylabel')` pro y osu.

Nakonec pak ještě můžeme přidat název grafu `title('name')`. Nejjednodušší syntax tohoto příkazu je `plot(x,y)`. Na následujícím příkladu si ukážeme několik těchto konfigurací.

```
>> x=linspace(0,8,50);
>> y1=2.5*cos(x);
>> y2=x.^2-(x*8);
>> plot(x, y1, '+red', x, y2, '--green')
>> title('Graf')
>> xlabel('x osa')
>> ylabel('y osa')
```



Obrázek 6 – plot figure

Graf se nám zobrazí v okně *Figure*, kde si ho následně můžeme posouvat, přibližovat nebo uložit do pc.

3.3.2 Příkaz fplot a fimplicit

Pro explicitně zadané matematické funkce se spíše hodí používat příkaz `fplot(f,[xmin xmax],linespec)`, kde parametr `f` je ve tvaru anonymní funkce `f=@(x) expr`, kde `expr` je předpis funkce s argumentem `x`. `xinterval` je interval ve tvaru `[xmin xmax]` na horizontální ose x, který chceme v grafu zobrazit. `linespec` opět slouží pro kosmetické úpravy křivky v grafu. Rozdíl je hlavně tedy v tom, že do příkazu nevkládáme dva vektory pro osy x a y, ale jen předpis funkce. Při vynechání argumentu `xinterval` se graf funkce zobrazí automaticky v intervalu `[-5,5]`.

Pro implicitní funkce použijeme příkaz `fimplicit(f,interval,linecspec)`, kde `f` je ve tvaru anonymní funkce `f=@(x,y) expr`, kde `expr` je předpis implicitní funkce s argumenty `x` a `y`. Parametr `interval` je v jednodušším tvaru `[min max]`, tedy na grafu zobrazený interval

stejný pro osy *x* a *y*. Parametr *interval* může být zadán i ve tvaru *[xmin xmax ymin ymax]*, kde první dvojice značí zobrazený interval pro osu *x* a druhá dvojice pro osu *y*.

```
>> funkce=@(x) cos(2/x);
>> interval =[0.1 1];
>> grf=fplot(funkce,interval);
>> grid on;
```

Zobrazí se nám mřížka pro větší přehlednost.

```
>> grf
```

Zobrazí se nám vlastnosti a parametry grafu, které můžeme dále upravovat (podobně jako přes *linespec*). Stejným příkazem lze zobrazit vlastnosti dalších typů grafů.

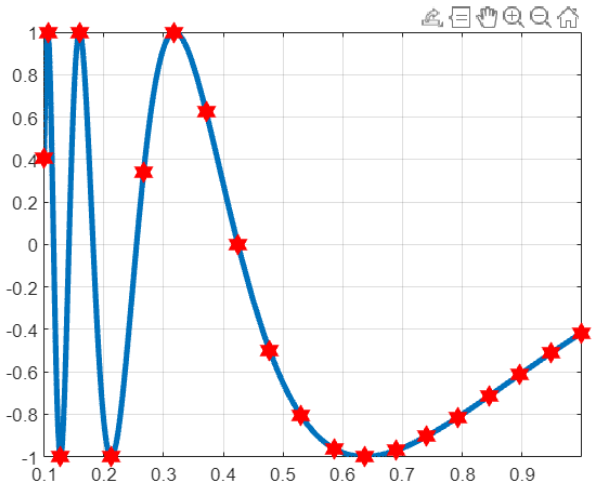
```
FunctionLine with properties:

 Function: @(x)cos(2/x)
 XRange: [0.1000 1]
 Color: [0 0.4470 0.7410]
 LineStyle: '-'
 LineWidth: 0.5000

 Show all properties
```

```
>> grf.LineWidth= 3;
>> grf.Marker="hexagram";
>> grf.MarkerEdgeColor= "r";
```

takto lze měnit jednotlivé vlastnosti grafu



Obrázek 7 – fplot figure

4. Numerické výpočty integrálů

V této hlavní kapitole si uvedeme několik hlavních numerických metod pro výpočet určitých integrálů. Každou z metod si vysvětlíme a ukážeme si její postup na jednodušším příkladu, který spočítáme „ručně“ a poté složitější příklad, který necháme spočítat Matlabem. Kromě počítání aproximacích integrálů budeme počítat i odhady absolutních chyb jednotlivých metod. Všechny matematické funkce, s kterými budeme dále pracovat jsou integrovatelné v Riemannově smyslu. K testování jednotlivých numerických metod budou vhodnější matematické funkce, které dostatečně oscilují. Pro tyto výpočty budeme v Matlabu vytvářet příslušné m-funkce/m-files. Pro některé tyto numerické metody má Matlab již implementovány své příkazy. Porovnáme přesnost jednotlivých metod a zjistíme čím se liší naše matlabovské funkce od zabudovaných příkazů. Nebudou chybět ani ilustrace pro lepší znázornění jednotlivých metod. Teoretické informace pro tuto kapitolu jsou primárně čerpány z [2].

4.1 Obdélníková metoda

Začneme asi nejintuitivnější metodou s názvem Obdélníková metoda. Jak už název napovídá, bude se jednat o rozdělení plochy pod křivkou integrované funkce na obdélníky u kterých sečteme jejich obsahy. Tato metoda vychází přímo z Rieamannovy definice určitého integrálu. Obsahy jednotlivých obdélníků získáme tak, že nahradíme obsah každého křivočarého lichoběžníku vymezeného grafem funkce $f: y = f(x)$ obdélníkem s jednou stranou na ose x s délkou $x_i - x_{i-1}$ a s druhou stranou s délkou danou hodnotou $f(\xi_i)$, kde $\xi_i \in \langle x_{i-1}, x_i \rangle$. Existuje několik možností pro volbu hodnot ξ_i.

Jedna z možností je ξ_i volit jako střed z podintervalu $\langle x_{i-1}, x_i \rangle$ jako $\xi_i = \frac{x_{i-1}+x_i}{2}$. Tato varianta obdélníkové metody je známa pod názvem Midpoint rule. Suma potom těchto všech obsahů obdélníků z funkce f je

$$\sum_{i=1}^{n} f(\xi_i)(x_i - x_{i-1}) = \sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right)(x_i - x_{i-1}).$$

Budeme vycházet z toho. že interval $\langle a, b \rangle$ je rozdělen rovnoměrně na n podintervalů s délkou $h = \frac{b-a}{n}$, pak lze výpočet aproximace integrálu zjednodušit na

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(\xi_i)(x_i - x_{i-1}) = h\sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right).$$

Pro odhad absolutní chyby aproximace integrálu $\int_{x_{i-1}}^{x_i} f(x)dx$ lze odvodit vzorec

$$E_i = \frac{h^3}{24} \max_{x \in \langle x_{i-1}, x_i \rangle} |f''(x)|.$$

Odhad absolutní chyby Obdélníkové metody je pak suma n dílčích odhadů absolutních chyb E_i na příslušných podintervalech

$E = \sum_{i=1}^{n} E_i = \frac{h^3}{24} \sum_{i=1}^{n} \max_{x \in \langle x_{i-1}, x_i \rangle} |f''(x)|.$

Hodnotu integrálu lze ještě vyjádřit jako neúplné číslo

$\int_a^b f(x)dx = h \sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right) \pm E.$

4.1.1 Příklad na obdélníkovou metodu

Zde si ukážeme na jednoduchém příkladu, jak probíhá výpočet této metody krok za krokem. Mějme funkci $f(x) = 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8$, kterou rozdělíme po ose x na 4 stejně dlouhé podintervaly na intervalu $\langle 0{,}2 \rangle$. Délku podintervalu h spočteme jako $h = \frac{2-0}{4} = 0{,}5.$

$$h\sum_{i=1}^{n} f\left(\frac{x_{i-1}+x_i}{2}\right) = 0{,}5\sum_{i=1}^{4} f\left(\frac{x_{i-1}+x_i}{2}\right) =$$

$$0{,}5 \cdot \left(f\left(\frac{0+0{,}5}{2}\right) + f\left(\frac{0{,}5+1}{2}\right) + f\left(\frac{1+1{,}5}{2}\right) + f\left(\frac{1{,}5+2}{2}\right)\right) =$$

$$0{,}5 \cdot \big(f(0{,}25) + f(0{,}75) + f(1{,}25) + f(1{,}75)\big) =$$

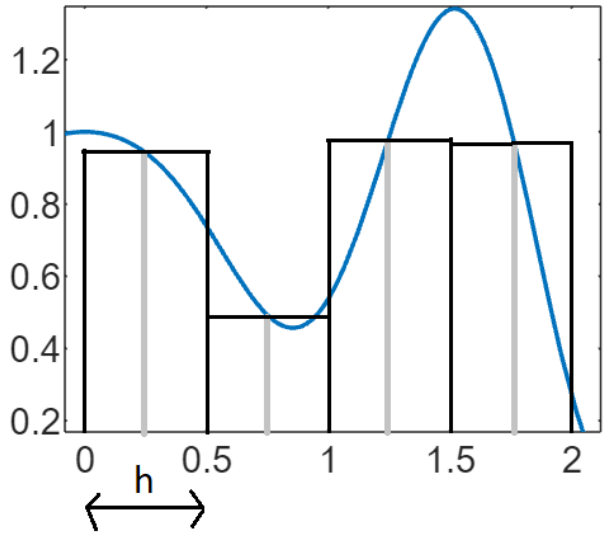
$$0{,}5 \cdot \left(0{,}2\cos(5 \cdot 0{,}25) + \frac{1}{3} \cdot 0{,}25\sin(5 \cdot 0{,}25) + 0{,}8\right) +$$

$$0{,}5 \cdot \left(0{,}2\cos(5 \cdot 0{,}75) + \frac{1}{3} \cdot 0{,}75\sin(5 \cdot 0{,}75) + 0{,}8\right) +$$

$$0{,}5 \cdot \left(0{,}2\cos(5 \cdot 1{,}25) + \frac{1}{3} \cdot 1{,}25\sin(5 \cdot 1{,}25) + 0{,}8\right) +$$

$$0{,}5 \cdot \left(0{,}2\cos(5 \cdot 1{,}75) + \frac{1}{3} \cdot 1{,}75\sin(5 \cdot 1{,}75) + 0{,}8\right) = 1{,}7147$$

Obrázek níže nám znázorňuje graf funkce $f(x) = 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8$ a na něm obdélníky, jejichž obsah jsme vypočítali na hodnotu 1,7147. Šedé čáry, které půlí obdélníky, nám znázorňují ξ_i a jejich funkční hodnoty $f(\xi_i)$. Z obrázku můžeme odhadovat, že skutečná hodnota integrálu se bude lišit od našich 1,7147.



Obrázek 8 – Graf funkce a obdélníková metoda

```
>> f=@(x) (0.2*cos(5*x)+1/3*x*sin(5*x)+0.8);
>> fplot(f,[-0.05 2.1])
```

4.1.2 Obdélníková metoda v Matlabu

Způsob, jak se obecně numerickými metodami pro výpočet integrálů více přiblížit ke skutečné hodnotě integrálu, je rozdělit interval na více podintervalů. K tomu využijeme Matlab, ve kterém si vytvoříme pro tuto metodu matlabovskou funkci. Výhoda spočívá v tom, že po jednorázovém vytvoření m-funkce budeme moci libovolně a rychle počítat integrál obdélníkovou metodou pro různé matematické funkce s různými intervaly a různým počtem podintervalů. Jednoduše tyto zmíněné proměnné vložíme jako argumenty do m-funkce stejně jako u klasických příkazů.

```
function [S] = obdelnik(funkce,a,b,n)
format ("long")
h=(b-a)/n
s=0;
x1=a;x2=a+h;
for i=1:1:n
y=feval(funkce,(x1+x2)/2);
s=s+y;
x1=x1+h;x2=x2+h;
end
S=h*s;
End
```

Do *Command Window* stačí napsat název m-funkce a do závorky argumenty. Za parametr `funkce` vložíme matematickou funkci ve tvaru anonymní funkce. Dále `a` a `b` jsou parametry značící meze intervalu rozdělené na `n` podintervalů. Díky příkazu `format ("Long")` uvidíme výstupní hodnoty v long formátu. Otestujeme naši funkci na stejné matematické funkci, pro kterou jsme počítali ručně aproximaci integrálu obdélníkovou metodou v předchozí

podkapitole. Za *n* tedy dosadíme 4, abychom ověřili funkčnost tím, že nám vyjde stejný výsledek jako v předchozím příkladě.

```
>> f=@(x) (0.2*cos(5*x)+1/3*x*sin(5*x)+0.8);
>> obdelnik(f,0,2,4)
h = 0.5000
ans = 1.714731351328701
```

Jako *ans* se nám zobrazuje výstup *S*, tedy aproximace integrálu obdélníkovou metodou. Pro zajímavost vidíme ještě hodnotu *h*, tedy délku jednotlivých podintervalů. Vidíme, že výsledek je shodný s výsledkem z předchozí podkapitoly. Nyní můžeme za *n* dosadit značně vyšší hodnotu, abychom dostali výsledek bližší skutečnému integrálu.

```
>> obdelnik(f,0,2,100)
h = 0.0200
ans = 1.682902328635482
```

Nakonec k této metodě si ještě vytvoříme m-funkci pro výpočet odhadů absolutních chyb.

```
function [] = obdchyba(funkce,a,b,n)
X=linspace(a,b,n+1);
syms x
h=(b-a)/n;
e=0;
fd=diff(funkce,2,x);
FDA=abs(subs(fd,x,X));
for i=1:n
summax=max(FDA(i),FDA(i+1));
e=e+summax;
end
E=((h^3)/24)*e;
vpa(E,16)
end
```

Vstupní parametry zůstávají stejné jako v předchozí m-funkci na samotnou obdélníkovou metodu. Příkaz *vpa* je zde použit z důvodu, že bez něj by nám Matlab vypisoval absolutní chybu *E* ve složeném tvaru. Funkci otestujeme na předchozích příkladech.

```
>> obdchyba(f,0,2,4)
ans = 0.175991575305066
```

Úplná aproximace integrálu je $\int_0^2 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\,dx = 1{,}7 \pm 0{,}2.$

```
>> obdchyba(f,0,2,100)
ans = 0.00018136342463450008
```

Úplná aproximace integrálu je $\int_0^2 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\,dx = 1{,}6829 \pm 0{,}0002.$

Získaný odhad absolutní chyby se zaokrouhlí nahoru na jednu platnou číslici. Na stejný číselný řád se zaokrouhlí i výsledek aproximace integrálu. Podle očekávání vyšla hodnota odhadu absolutní chyby vyšší u příkladu s `n=4` než u příkladu s `n=100`.

4.2 Lichoběžníková metoda

Další metodou je Lichoběžníková metoda. Pomocí ní plochu pod křivkou integrované funkce rozdělíme na lichoběžníky a jejich obsahy sečteme. Zdánlivě lze očekávat, že přesnost této metody bude vyšší než u obdélníkové metody, protože horní strany lichoběžníků budou lépe „přiléhat“ ke křivce a tím pádem obsahy lichoběžníků přesněji vyplní plochu pod křivkou v grafu funkce.

Mějme spojitou funkci f s děleným intervalem určeným dělícími body $x_0, x_1, x_2, \dots, x_n$. Funkční hodnoty v dělících bodech označíme jako $y_0 = f(x_0) = f(a)$, $y_1 = f(x_1)$, $y_2 = f(x_2)$, $y_n = f(x_n) = f(b)$. V grafu funkce lichoběžníky získáme tak, že dvě rovnoběžné strany budou určeny úsečkami danými dvojicí bodů $[x_{i-1}, f(x_{i-1})]$ a $[x_i, f(x_i)]$. Spodní strana lichoběžníku potom bude mít délku $x_i - x_{i-1}$ a horní strana bude úsečka spojující body $[f(x_{-1}), f(x_i)]$. Délku horní strany nemusíme znát, protože obsah lichoběžníku je $S = \frac{a+b}{2} \cdot h$, kde strany a a b jsou rovnoběžné a h je výška lichoběžníku tedy v našem případě $x_i - x_{i-1}$. Dílčí obsah lichoběžníku z grafu funkce je tedy $P_i = \frac{y_{i-1}+y_i}{2} \cdot (x_i - x_{i-1})$.

Integrál $\int_a^b f(x)dx$ je potom aproximován lichoběžníkovou metodou jako

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} P_i = \sum_{i=1}^{n} \frac{y_{i-1}+y_i}{2} \cdot (x_i - x_{i-1}).$$

V případě s rovnoměrným rozdělením intervalu $\langle a, b \rangle$ na n podintervalů, kde jejich délka je $h = \frac{b-a}{n}$, kterou nahradíme člen $(x_i - x_{i-1})$. Stejný výpočet lze vyjádřit jako

$$\int_a^b f(x)dx \approx \frac{h}{2}(y_0 + 2y_1 + 2y_2 + \cdots + 2y_{n-1} + y_n),$$

Nebo

$$\int_a^b f(x)dx \approx \frac{h}{2}(y_0 + 2\sum_{i=1}^{n-1} y_i + y_n).$$

Pro odhad absolutní chyby aproximace integrálu $\int_{x_{i-1}}^{x_i} f(x)dx$ lze odvodit vzorec

$$E_i = \frac{h^3}{12} \max_{x \in \langle x_{i-1}, x_i \rangle} |f''(x)|.$$

Odhad absolutní chyby lichoběžníkové metody je pak suma n dílčích odhadů absolutních chyb E_i na příslušných podintervalech

$$E = \sum_{i=1}^{n} E_i = \frac{h^3}{12} \sum_{i=1}^{n} \max_{x \in \langle x_{i-1}, x_i \rangle} |f''(x)|.$$

Hodnotu integrálu lze ještě vyjádřit jako neúplné číslo

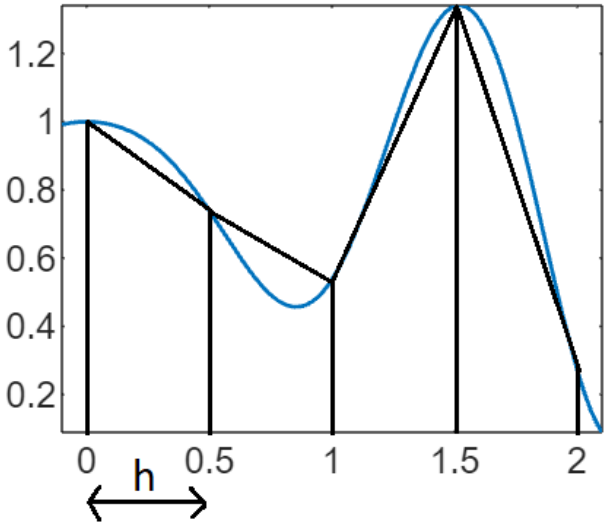
$$\int_a^b f(x)dx = \frac{h}{2}(y_0 + 2\sum_{i=1}^{n-1} y_i + y_n) \pm E.$$

4.2.1 Příklad na lichoběžníkovou metodu

Vypočítáme si obdobný příklad jako v příkladu na obdélníkovou metodu 4.1.1. Uvidíme, zda se při stejném počtu podintervalů bude lichoběžníková metoda ve svém výsledku lišit. Funkci $f(x) = 0{,}2\ \cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8$ rozdělíme po ose x na 4 stejně dlouhé podintervaly na integrovaném intervalu $\langle\, 0{,}2 \,\rangle$. Délku podintervalu h spočítáme jako $h = \frac{2-0}{4} = 0{,}5$.

$$\frac{h}{2}\left(f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right) = \frac{0{,}5}{2}\left(f(x_0) + 2\sum_{i=1}^{3} f(x_i) + f(x_4)\right) =$$
$$\frac{0{,}5}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + f(x_4)\right) =$$
$$\frac{0{,}5}{2}\left(0{,}2\ \cos(5\cdot 0) + \frac{1}{3}\cdot 0\sin(5\cdot 0) + 0{,}8\right) +$$
$$\frac{0{,}5}{2}\cdot 2\left(0{,}2\ \cos(5\cdot 0{,}5) + \frac{1}{3}\cdot 0{,}5\sin(5\cdot 0{,}5) + 0{,}8\right) +$$
$$\frac{0{,}5}{2}\cdot 2\left(0{,}2\ \cos(5\cdot 1) + \frac{1}{3}\cdot 1\sin(5\cdot 1) + 0{,}8\right) +$$
$$\frac{0{,}5}{2}\cdot 2\left(0{,}2\ \cos(5\cdot 1{,}5) + \frac{1}{3}\cdot 1{,}5\sin(5\cdot 1{,}5) + 0{,}8\right) +$$
$$\frac{0{,}5}{2}\left(0{,}2\ \cos(5\cdot 2) + \frac{1}{3}\cdot 2\sin(5\cdot 2) + 0{,}8\right) = 1{,}6248$$

Obrázek níže nám znázorňuje graf funkce $f(x) = 0{,}2\ \cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8$ a na něm vyznačené lichoběžníky, u kterých jsme vypočítali součet jejich obsahu na 1,6248.



Obrázek 9 – Graf funkce a lichoběžníková metoda

```
>> f=@(x) (0.2*cos(5*x)+1/3*x*sin(5*x)+0.8);
>> fplot(f,[-0.05 2.1])
```

4.2.2 Lichoběžníková metoda v Matlabu

Pro lichoběžníkovou metodu vytvoříme další matlabovskou funkci. To i přesto, že v Matlabu už existuje přímo příkaz na počítání integrálů lichoběžníkovou metodou (kterým se budeme zabývat v další podkapitole). Jak uvidíme má totiž lehce jinou aplikaci.

```
function [S] = lichobeznik(funkce,a,b,n)
format ("long")
h=(b-a)/n
s=0;
x1=a;x2=a+h;
for i=1:1:n
y=(feval(funkce,x1)+feval(funkce,x2))/2;
s=s+y;
x1=x1+h;x2=x2+h;
end
S=h*s;
end
```

Opět otestujeme naši funkci tak, že ji necháme spočítat stejný příklad jako v předchozí podkapitole. Za *funkce* vložíme matematickou funkci ve tvaru anonymní funkce. Za *a,b* vložíme meze intervalu s *n* počtem podintervalů.

```
>> f=@(x) (0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8);
>> lichobeznik(f,0,2,4)
h = 0.5000
ans = 1.624843587598239
```

Dostali jsme stejný výsledek. Nyní počet podintervalů *n* zvýšíme na 100.

```
>> lichobeznik(f,0,2,100)
h = 0.0200
ans = 1.682780588275007
```

Pro lichoběžníkovou metodu vytvoříme m-funkci, která počítá odhad absolutní chyby.

```
function [] = lichchyba(funkce,a,b,n)
X=linspace(a,b,n+1);
syms x
h=(b-a)/n;
e=0;
fd=diff(funkce,2,x);
FDA=abs(subs(fd,x,X));
for i=1:n
summax=max(FDA(i),FDA(i+1));
e=e+summax;
end
E=((h^3)/12)*e;
vpa(E,16)
end
```

Funkce je téměř identická s funkcí pro výpočet odhadů absolutních chyb pro obdélníkovou metodu. Liší se pouze číselnou hodnotou v jednom děliteli. Necháme si Matlabem spočítat odhady absolutních chyb u dvou předešlých příkladů.

```
>> lichchyba(f,0,2,4)
ans = 0.351983150610132
```

Úplná aproximace integrálu je $\int_0^2 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\,dx = 1{,}6 \pm 0{,}4$.

```
>> lichchyba(f,0,2,100)
ans = 0.0003627268492690016
```

Úplná aproximace integrálu je $\int_0^2 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\,dx = 1{,}6828 \pm 0{,}0004$.

Odhad absolutní chyby `E` vyšel nižší podle očekávání u příkladu s `n=100`.

4.2.3 Příkaz trapz

Pro lichoběžníkovou metodu má Matlab přímo implementován příkaz `trapz(x,y)` – z anglického slova trapezoid neboli lichoběžník. Za `x` dosadíme vektor, jehož hodnoty jsou dělící body $a = x_0, x_1, x_2, \dots, x_n = b$ v intervalu $\langle a, b \rangle$. Za `y` potom stačí dosadit vektor jehož hodnoty jsou funkční hodnoty funkce f v dělících bodech $y_0 = f(x_0)$, $y_1 = f(x_1)$, $y_2 = f(x_2), \dots, y_n = f(x_n)$. Oba vektory musí mít stejný počet hodnot $n + 1$, ale nemusí být rozloženy rovnoměrně. Na rozdíl od naší m-funkce v předchozí podkapitole, kde interval matematické funkce dělíme rovnoměrně na n podintervalů s délkou $h = \frac{b-a}{2}$. Další výhoda tohoto příkazu je, že nemusíme znát předpis funkce, ale jen data. Pokud vektor `x` do příkazu nedosadíme, Matlab spočítá příkaz implicitně s vektorem `x=1:1:n`, kde `n` je délka vektoru `y`.

Z důvodu jednotnosti si vytvoříme m-funkci i pro příkaz *trapz*.

```
function [S] = trapzf(funkce,a,b,n)
format("long")
x=linspace(a,b,n+1);
y=feval(funkce,x);
S=trapz(x,y);
end
```

Stejný příklad jako v předchozí podkapitole vypočítáme příkazem *trapz*.

```
>> f=@(x) (0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8);
>> trapzf(f,0,2,100)
ans = 1.682780588275008
```

Výsledky se liší jen zanedbatelně v poslední číslici o 1. Pravděpodobně jde jen o rozdíl v zaokrouhlování.

4.3 Simpsonova metoda

Jako další metodu pro numerické počítání určitých integrálů si uvedeme Simpsonovu metodu – konkrétně Simpsonovo 1/3 pravidlo. Co se týče grafické interpretace je tato metoda z předchozích dvou nejméně intuitivní. Vychází se zde z Lagrangeova interpolačního polynomu druhého stupně. Uvažujme spojitou a nezápornou funkci f definovanou na intervalu $\langle a, b \rangle$ s dělením tohoto intervalu na sudý počet n stejně dlouhých podintervalů s délkou $h = \frac{b-a}{n}$. Dělící body označíme $x_0, x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n$ a funkční hodnoty v těchto bodech označíme jako $y_0 = f(x_0) = f(a), y_1 = f(x_1), y_2 = f(x_2), \dots, y_{i-1} = f(x_{i-1}), y_i = f(x_i), y_{i+1} = f(x_{i+1}), \dots, y_{n-1} = f(x_{n-1}), y_n = f(x_n) = f(b)$. Pro definici Simpsonovy metody uvažujme i jako liché číslo. Aproximace určitého integrálu touto metodou spočívá v nahrazení oblouku grafu funkce f nad dvojici podintervalů $\langle x_{i-1}, x_i \rangle$ a $\langle x_i, x_{i+1} \rangle$ obloukem paraboly, který je určen body grafu Lagrangeova interpolačního polynomu druhého stupně daným body $[x_{i-1}, f(x_{i-1})]$,$[x_i, f(x_i)]$ a $[x_{i+1}, f(x_{i+1})]$. Obsah obrazce P_i je tedy vymezen zdola osou x, shora obloukem paraboly Lagrangeova interpolačního polynomu druhého stupně pro $x \in \langle x_{i-1}, x_{i+1} \rangle$, zleva přímkou $x = x_{i-1}$ a zprava přimkou $x = x_{i+1}$. Obsah P_i tohoto obrazce se spočítá podle vzorce

$$P_i = \frac{h}{3}(y_{i-1} + 4y_i + y_{i+1}).$$

Simpsonova aproximace integrálu $\int_a^b f(x)dx$ je pak dána součtem všech $\frac{n}{2}$ obsahů P_i jako

$$\int_a^b f(x)dx \approx \sum_{i=1}^{\frac{n}{2}} P_i = \frac{h}{3}\sum_{i=1}^{\frac{n}{2}}(y_{2i-2} + 4y_{2i-1} + y_{2i}).$$

Další způsob vyjádření je

$$\int_a^b f(x)dx \approx \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n),$$

nebo

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(y_0 + 4\sum_{i=1}^{\frac{n}{2}} y_{2i-1} + 2\sum_{i=1}^{\frac{n}{2}-1} y_{2i} + y_n\right).$$

Pro odhad absolutní chyby aproximace integrálu $\int_{x_{i-1}}^{x_{i+1}} f(x)dx$ lze odvodit vzorec

$$E_i = \frac{h^5}{90} \max_{x \in \langle x_{i-1}, x_{i+1} \rangle} \left|f^{(4)}(x)\right|.$$

Odhad absolutní chyby Simpsonovy metody je pak suma n dílčích odhadů absolutních chyb E_i na příslušných podintervalech

$$E = \sum_{i=1}^{\frac{n}{2}} E_i = \frac{h^5}{90} \sum_{i=1}^{\frac{n}{2}} \max_{x \in \langle x_{2i-2}, x_{2i} \rangle} \left|f^{(4)}(x)\right|.$$

Hodnotu integrálu lze ještě vyjádřit jako neúplné číslo

$$\int_a^b f(x)dx = \frac{h}{3}\left(y_0 + 4\sum_{i=1}^{\frac{n}{2}} y_{2i-1} + 2\sum_{i=1}^{\frac{n}{2}-1} y_{2i} + y_n\right) \pm E.$$

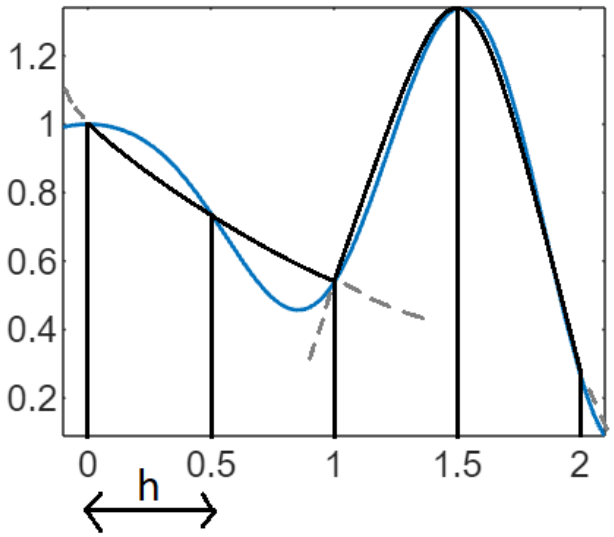
4.3.1 Příklad na Simpsonovu metodu

Pro funkci $f(x) = 0{,}2\ \cos(5x) + \frac{1}{3}x \sin(5x) + 0{,}8$ vypočítáme ručně aproximaci integrálu Simpsonovou metodou na intervalu $\langle\ 0{,}2\ \rangle$, který rozdělíme rovnoměrně na 4 podintervaly se stejnou délkou $h = \frac{2-0}{4} = 0{,}5$.

$$\frac{h}{3}\sum_{i=1}^{\frac{n}{2}} \left(f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\right) = \frac{0{,}5}{3}\sum_{i=1}^{2} \left(f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\right) =$$
$$\frac{0{,}5}{3} \cdot \left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4)\right) =$$
$$\frac{0{,}5}{3}\left(0{,}2\ \cos(5 \cdot 0) + \frac{1}{3} \cdot 0 \sin(5 \cdot 0) + 0{,}8\right) +$$
$$\frac{2}{3}\left(0{,}2\ \cos(5 \cdot 0{,}5) + \frac{1}{3} \cdot 0{,}5 \sin(5 \cdot 0{,}5) + 0{,}8\right) +$$
$$\frac{1}{3}\left(0{,}2\ \cos(5 \cdot 1) + \frac{1}{3} \cdot 1 \sin(5 \cdot 1) + 0{,}8\right) +$$
$$\frac{2}{3}\left(0{,}2\ \cos(5 \cdot 1{,}5) + \frac{1}{3} \cdot 1{,}5 \sin(5 \cdot 1{,}5) + 0{,}8\right) +$$
$$\frac{0{,}5}{3}\left(0{,}2\ \cos(5 \cdot 2) + \frac{1}{3} \cdot 2 \sin(5 \cdot 2) + 0{,}8\right) = 1{,}7758$$

Obrázek níže nám znázorňuje graf funkce $f(x) = 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8$ a na něm přibližně vyznačené dva obrazce. Jeden z nich je ohraničen na ose x na intervalu

$\langle 0{,}1 \rangle$ a druhý na $\langle 1{,}2 \rangle$. Shora je každý z obrazců ohraničený jednou parabolou. Jejich celkový obsah je 1,7758. Šedou přerušovanou čárou jsou znázorněny pokračování parabol.



Obrázek 10 – Graf funkce a Simpsonovo 1/3 pravidlo

```
>> f=@(x) (0.2*cos(5*x)+1/3*x*sin(5*x)+0.8);
>> fplot(f,[-0.05 2.1])
```

4.3.2 Simpsonova metoda v Matlabu

Vytvoříme si matlabovskou funkci pro Simpsonovo 1/3 pravidlo. V Matlabu existuje příkaz na Simpsonovu metodu, ale jeho algoritmus a použití je jiné (budeme řešit v následující podkapitole).

```
function [S] = simpson(funkce,a,b,n)
if mod(n,2)==1
disp('Pocet podintervalu n musi byt sudy')
else
h=(b-a)/n, h2=2*h;
s=0;
x1=a;x2=a+h;x3=a+2*h;
for i=1:1:n/2
y=(feval(funkce,x1)+4*feval(funkce,x2)+feval(funkce,x3));
x1=x1+h2;x2=x2+h2;x3=x3+h2;
s=s+y;
end
S=h/3*s;
end
end
```

Pro správnou funkčnost Simpsonového 1/3 pravidla je nutné, aby počet podintervalů *n* byl sudý. Proto testujeme hodnotu *n* na dělitelnost dvěma beze zbytku. K tomu nám poslouží příkaz *mod(a,m)*, který vrací zbytek po dělení hodnoty *a* hodnotou *m*. Funkci otestujeme na příkladě z předchozí podkapitoly.

```
>> f=@(x) (0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8);
>> simpson(f,0,2,4)
h = 0.5000
ans = 1.775843620464042

>> simpson(f,0,2,5)
Pocet podintervalu n musi byt sudy
```

Výsledek prvního výpočtu se shoduje s příkladem z kapitoly 4.3.1. Pro *n=5* nám Matlab správně místo výpočtu vypsal chybovou hlášku. Dále si vypočítáme příklad s *n=100* pro výsledek bližší skutečnému integrálu a pro porovnání výsledku s dalšími metodami o stejném *n*.

```
>> simpson(f,0,2,100)
h = 0.0200
ans = 1.682861806866558
```

Vytvoříme si m-funkci pro počítání odhadů absolutních chyb pro Simpsonovo 1/3 pravidlo.

```
function [] = simpchyba(funkce,a,b,n)
if mod(n,2)==1
disp('Pocet podintervalu n musi byt sudy')
else
X=linspace(a,b,n+1);
syms x
h=(b-a)/n;
e=0;
fd=diff(funkce,4,x);
FDA=abs(subs(fd,x,X));
for i=1:n/2
summax=max([FDA(2*i-1),FDA(2*i),FDA(2*i+1)]);
e=e+summax;
end
E=((h^5)/90)*e;
vpa(E,16)
end
end
```

Necháme funkci spočítat odhady absolutních chyb z předchozích příkladů.

```
>> simpchyba(f,0,2,4)
ans = 0.1702350704102204
```

Úplná aproximace integrálu je $\int_0^2 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\,dx = 1{,}8 \pm 0{,}2.$

```
>> simpchyba(f,0,2,100)
ans = 0.0000002553844496475596
```

Úplná aproximace integrálu je $\int_0^2 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\,dx = 1{,}6828618 \pm 0{,}0000003.$

```
>> simpchyba(f,0,2,5)
Pocet podintervalu n musi byt sudy
```

Matlab nám vypočítal vyšší hodnotu absolutní chyby u příkladu s *n=4* a správně vrátil chybovou hlášku při *n=5*.

4.3.3 Příkaz quad

Pro Simpsonovu metodu má Matlab implementován příkaz *quad(fun,a,b,tol)*. Nejedná se přesně o Simpsonovo 1/3 pravidlo, které jsme řešili doposud, ale o Adaptivní rekurzivní Simpsonovo pravidlo. Parametr *fun* je ve tvaru anonymní funkce *fun=@(x) expr*, kde *expr* je integrovaná funkce. Parametry *a,b* jsou meze integrálu a *tol* je tolerance absolutní chyby z jednotlivých podintervalů. Čím menší toleranci absolutní chyby zvolíme, tím bude výpočet déle trvat, protože se tím zvyšuje počet operací. Bez zadání argumentu *tol* se použije implicitně hodnota 0,000001. [8]

Vytvářet m-funkci pro příkaz *quad* (podobně jako v 4.2.3 u příkazu *trapz*) je možná až zbytečné, ale pro jednotnost ji přesto vytvoříme.

```
function [S] = quadf(funkce,a,b,tol)
format("long")
S=quad(funkce,a,b,tol);
end
```

```
>> f=@(x) (0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8);
>> quadf(f,0,2,0.001)
ans = 1.682808243996129

>> quadf(f,0,2,0.00000000001)
ans = 1.682861744627483
```

Druhý výpočet je blíž ke skutečné hodnotě integrálu funkce *f* kvůli nižší zvolené hodnotě tolerance absolutní chyby *tol*.

4.4 Metoda Monte Carlo

Poslední metodu, kterou si uvedeme je jedna ze základních verzí metody Monte Carlo, která k aproximaci integrálu přistupuje značně odlišně od předchozích metod. Používá totiž generaci náhodných čísel. Navíc je snadno použitelná i pro funkce o více proměnných. Uvažujme spojitou a nezápornou funkci f definovanou na intervalu $\langle a, b \rangle$ s N dělícími body $X_1, X_2, \dots, X_n$, jejichž hodnoty jsou náhodně zvolená reálná čísla z intervalu $\langle a, b \rangle$. Aproximace integrálu touto metodou spočívá v zisku aritmetického průměru N funkčních hodnot $f(X_i)$ a jeho vynásobením délkou intervalu $\langle b - a \rangle$ jako

$$\int_a^b f(x)dx \approx (b-a)\frac{1}{N}\sum_{i=1}^{N} f(X_i).$$

Odhad absolutní chyby spočítáme vzorcem

$$E = \sqrt{\frac{\frac{b-a}{N-1}\sum_{i=1}^{N} f^2(X_i) - (b-a)\mu^2}{N}} = \frac{\sigma}{\sqrt{N}}.$$

Kde μ je střední hodnota z $f(X_1), f(X_2), \dots, f(X_n)$ a σ je směrodatná odchylka.

Hodnotu integrálu lze ještě vyjádřit jako neúplné číslo

$$\int_a^b f(x)dx = (b-a)\frac{1}{N}\sum_{i=1}^{N} f(X_i) \pm E.$$

[4]

4.4.1 Metoda Monte Carlo v Matlabu

Vzhledem k tomu, že metoda Monte Carlo vyžaduje velký počet operací a generaci náhodných čísel, tak přeskočíme část s „ručním“ počítáním a přejdeme rovnou k tvorbě m-funkce, která bude počítat aproximaci integrálu a zároveň odhad absolutní chyby. Počítat jej samostatně by totiž při této metodě nebylo dobře realizovatelné. Využijeme nově další dva příkazy `mean` a `rand`. Příkaz `mean(A)` slouží k výpočtu střední hodnoty z pole `A`. Pokud je `A` vektor, tak příkaz vrátí jednu střední hodnotu. Pokud `A` je matice, tak příkaz vrátí střední hodnoty pro každý jednotlivý sloupec z `A`. Příkaz `rand(d1,…,dn)` náhodně generuje reálná čísla od 0 do 1. Parametry `d1,…,dn` značí velikost `n`-té dimenze výstupního pole. Například `rand(2,3)` nám vrátí matici 2×3.

```
function [S] = montecarlo(funkce,a,b,n)
format("long")
x=a+(b-a)*rand(n,1);
y=feval(funkce,x);
prumer=mean(y)
S=(b-a)*prumer;
yy=sum(y.*y);
prum2=prumer*prumer;
E=sqrt(((b-a)/n*yy-(b-a)*prum2)/n)
end
```

Za parametr *funkce* vkládáme anonymní funkci, pro kterou chceme počítat aproximaci integrálu v mezích od *a* do *b*. Parametr *n* značí počet náhodných, respektive pseudonáhodných bodů v rozmezí od *a* do *b*. M-funkce nám vrací kromě odhadu integrálu *ans*, taky *prumer* který značí střední hodnotu z vybraných *n* funkčních hodnot a *E* odhad absolutní chyby.

Naši m-funkci vyzkoušíme třikrát s argumentem *n=1000* a poté třikrát s *n=10^6*.

```
>> ff=(0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8)-y;
>> montecarlo(f,0,2,1000)

prumer = 0.838004520245318, 0.832832183659233, 0.856552345154138
E = 0.012446679682183, 0.012806755820392, 0.012681446205458
ans = 1.676009040490635, 1.665664367318467, 1.713104690308277
```

Úplná aproximace integrálu je $\int_0^2 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\,dx = 1{,}68 \pm 0{,}02.$

```
>> montecarlo(f,0,2,10^6)

prumer = 0.841394738106136, 0.841854335322751, 0.841420137304293
E = 4.007590044532652e-04, 4.008787293465111e-04, 4.007997072913699e-04
ans = 1.682789476212271, 1.683708670645503, 1.682840274608587
```

Úplná aproximace integrálu je $\int_0^2 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\,dx = 1{,}6828 \pm 0{,}0004.$

Vidíme, že při *n=1000* máme mezi výstupními hodnoty poměrně větší rozptyl jak s *n=10^6*.

4.5 Řešení pro implicitní funkce

I pro implicitní funkce lze aproximovat určité integrály dříve zmíněnými metodami. Bude ale zapotřebí modifikovat naše kódy matlabovských funkcí, které jsou stavěné pro explicitní funkce. Implicitní funkce je funkce zadaná ve formě anulované rovnice s alespoň dvěma neznámými, kde jednu proměnnou považujeme za argument funkce a druhou proměnnou za funkční hodnotu. Příklad jednoduché implicitní funkce je třeba $2x - y = 0$, kterou lze převést na explicitní funkci jako $f(x) = 2x$.

V našich příkladech jsme doposud pracovali s explicitní funkcí $f(x) = 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8$, kterou si pro následující příklady nejjednodušším způsobem upravíme na implicitní funkci jako $0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8 - y = 0$. Jako argument budeme uvažovat proměnnou x a jako funkční hodnotu proměnnou y. Doba vyhodnocení následujících m-funkcí je výrazně delší (z důvodu použití příkazu *solve*) než doba pro vyhodnocení m-funkcí z předchozích kapitol 4.1-4.

4.5.1 Obdélníková metoda pro implicitní funkce

Budeme uvažovat stejný typ obdélníkové metody jako v kapitole 4.1. Funkční hodnoty pro výpočet budeme tedy opět volit jako prostřední hodnoty v podintervalech (Midpoint rule). Vytvoříme m-funkci pro výpočet integrálu obdélníkovou metodou pro implicitní funkce.

```
function [] = impobdelnik(funkce,a,b,n)
syms x y
h=(b-a)/n
s=0;
x1=a;x2=a+h;
for i=1:1:n
f1=subs(funkce,x,(x1+x2)/2);
y=solve(f1);
s=s+y;
x1=x1+h;x2=x2+h;
end
S=h*s;
vpa(S,16)
end
```

Za parametr *funkce* vkládáme funkci ve tvaru anulované rovnice se symbolickými proměnnými *x* a *y*. Kde *x* uvažujeme jako argument funkce a *y* jako funkční hodnotu. Dále parametry *a* a *b* jsou meze integrálu a *n* je počet podintervalů.

```
>> syms x y
>> ff=(0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8)-y;
>> impobdelnik(ff,0,2,100)
h = 0.0200
ans = 1.682902328635482
```

Vidíme, že pro stejnou funkci v implicitním tvaru vyšel výsledek stejně jako v příkladu s *n=100* v kapitole 4.1.2. Tato funkce bude fungovat i na jiné implicitní funkce, ale je nutné definovat symbolickou proměnnou *x* jako argument funkce a *y* jako funkční hodnotu.

4.5.2 Lichoběžníková metoda pro implicitní funkce

Další m-funkci vytvoříme pro lichoběžníkovou metodu. Lehce upravíme funkci z kapitoly 4.2.

```
function [] = implichobeznik(funkce,a,b,n)
syms x y;
h=(b-a)/n
s=0;
x1=a;x2=a+h;
for i=1:1:n
f1=subs(funkce,x,x1);
f2=subs(funkce,x,x2);
y=(solve(f1)+solve(f2))/2;
s=s+y;
x1=x1+h;x2=x2+h;
end
S=h*s;
vpa(S,16)
end
```

Za parametr *funkce* vkládáme funkci ve tvaru anulované rovnice se symbolickými proměnnými *x* a *y*. Kde *x* uvažujeme jako argument funkce a *y* jako funkční hodnotu. Dále parametry *a* a *b* jsou meze integrálu a *n* je počet podintervalů. Otestujeme m-funkci na stejné funkci jako v příkladu 4.2.2, ale tentokrát v implicitním tvaru, abychom zjistili, zda nám vyjde stejný výsledek.

```
>> syms x y
>> ff=(0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8)-y;
>> implichobeznik(ff,0,2,100)
h = 0.0200
ans = 1.682780588275007
```

Výsledky jsou srovnatelné.

4.5.3 Příkaz trapz pro implicitní funkce

Příkaz *trapz* sám o sobě není stavěný na implicitní funkce, ale lze vytvořit m-funkci, která z implicitní funkce vytvoří dva vektory *x* a *y*, které už půjdou vložit jako argumenty do příkazu *trapz*.

```
function [] = imptrapz(funkce,a,b,n)
syms x y
format ("long")
X=linspace(a,b,n+1);
f=subs(funkce,x,X);
for i=1:1:n+1
Y(i)=solve(f(i))
end
S=trapz(X,Y);
vpa(S,16)
end
```

Parametry jsou stejné jako v předchozích m-funkcích.

```
>> syms x y
>> ff=(0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8)-y;
>> imptrapz(ff,0,2,100)
ans = 1.682780588275007
```

Výsledek je srovnatelný s příkladem z kapitoly 4.2.3.

4.5.4 Simpsonova metoda pro implicitní funkce

Pro Simpsonovu metodu (přesněji Simpsonovo 1/3 pravidlo) si vytvoříme m-funkci pro práci s implicitními funkcemi. Příkaz *quad*, který ke svému řešení používá upravenou Simpsonovu metodu, nejde použít pro řešení integrace implicitních funkcích.

```
function [] = impsimpson(funkce,a,b,n)
syms x y
if mod(n,2)==1
disp('Pocet podintervalu n musi byt sudy')
else
h=(b-a)/n, h2=2*h;
s=0;
x1=a;x2=a+h;x3=a+2*h;
for i=1:1:n/2
f1=subs(funkce,x,x1); f2=subs(funkce,x,x2); f3=subs(funkce,x,x3);
y=(solve(f1)+4*solve(f2)+solve(f3));
x1=x1+h2;x2=x2+h2;x3=x3+h2;
s=s+y;
end
S=h/3*s;
vpa(S,16)
end
end
```

Za parametr *funkce* vložíme funkci ve tvaru anulované rovnice se symbolickými proměnnými *x* a *y*. Parametry *a* a *b* jsou meze integrálu a *n* je počet podintervalů. Funkce navíc hlídá, aby *n* bylo sudé.

```
>> syms x y
>> ff=(0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8)-y;
>> impsimpson(ff,0,2,100)
h = 0.0200
ans = 1.682861806866559
```

Výsledek je shodný s příkladem z kapitoly 4.3.3.

4.5.5 Metoda Monte Carlo pro implicitní funkce

Nakonec vytvoříme m-funkci pro aproximaci integrálu implicitní funkce metodou Monte Carlo.

```
function [S] = impmontecarlo(funkce,a,b,n)
syms x y
X=a+(b-a)*rand(n,1);
f=subs(funkce,x,X);
for i=1:1:n
y(i)=solve(f(i));
end
prumer=vpa(mean(y),16)
S=vpa((b-a)*prumer,16);
yy=sum(y.*y);
prum2=prumer*prumer;
E=vpa(sqrt(((b-a)/n*yy-(b-a)*prum2)/n),16)
end
```

Za parametr *funkce* vložíme funkci ve tvaru anulované rovnice. *a* a *b* jsou meze integrálu a *n* je počet náhodně vygenerovaných hodnot v rozmezí od *a* do *b*. Tato m-funkce kvůli spoustě volání příkazu *solve* se vyhodnocuje značně nejdéle ze všech předchozích m-funkcí. Pro měření času můžeme použít příkazy *tic* a *toc*. První z příkazů spouští měření a druhý příkaz měření zastavuje a vypíše uběhnutý čas.

```
>> syms x y
>> ff=(0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8)-y;
>> tic
impmontecarlo(ff,0,2,1000)
toc

prumer = 0.8401179751172475
E = 0.0125479978791936
ans = 1.680235950234495
Elapsed time is 42.202852 seconds.
```

Metoda Monte Carlo pro implicitní funkce se zdá kvůli délce výpočtu jako nejméně vhodná.

4.6 Porovnání metod

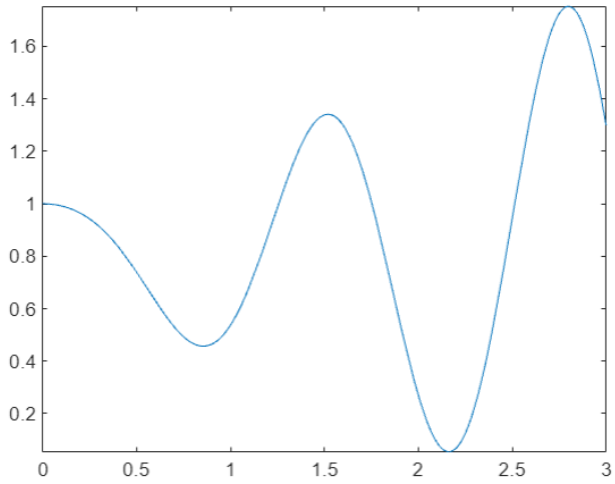
V poslední kapitole si porovnáme přesnost a případně čas potřebný pro výpočet jednotlivých numerických metod/m-funkcí v závislosti na počtu podintervalů.

4.6.1 Oscilační funkce

Jako funkci s oscilačním charakterem můžeme použít opět funkci
$f(x) = 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8$, kterou budeme integrovat na intervalu $\langle\, 0{,}3 \,\rangle$.
Funkci si na daném intervalu zobrazíme a zároveň spočítáme určitý integrál jako rozdíl dvou funkčních hodnot primitivní funkce.

```
>> f=@(x) (0.2*cos(5.*x)+1/3.*x.*sin(5.*x)+0.8);
>> fplot(f,[0,3])
>> syms x
>> vpa(int(f,x,0,3),16)
ans = 2.586619600713477
```

Tedy $\int_0^3 0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8\ dx \doteq 2{,}5866196.$



Obrázek 11 – Oscilační funkce

Následující tabulky ukazují výsledky aproximace integrálů jednotlivých numerických metod aplikované na příklad $\int_0^3 (0{,}2\cos(5x) + \frac{1}{3}x\sin(5x) + 0{,}8)\ dx$ v závislosti na počtu podintervalů n. V případě metody Monte Carlo se jako n myslí počet pseudonáhodně vygenerovaných hodnot. Pro příkaz *quad* jsou výsledky závislé na hodnotě *tol* – tolerance absolutní chyby integrace na jednotlivých podintervalech.

Tabulka 1 – Výsledky pro oscilační funkci

	n=10	n=500	n=2000	odchylka (n=10)	odchylka (n=500)	odchylka (n=2000)
obdelník.	2,603476	2,586626	2,58662	0,0169	6,40E-06	4,0E-07
lichoběž.	2,553758	2,586607	2,5866188	-0,0329	-1,26E-05	-8,0E-07
simpson.	2,592166	2,586619	2,5866196	0,0055	-6,00E-07	0,0E+00
monte carlo	3,442048	2,550638	2,5655549	0,8554	-3,60E-02	-2,1E-02
trapz	2,553758	2,586607	2,5866188	-0,0329	-1,26E-05	-8,0E-07

Tabulka 2 – Výsledky quad pro oscilační funkci

	tol=0,01	tol=10^-6	tol=10^-9	odchylka (tol=0,01)	odchylka (tol=10^-6)	odchylka (tol=10^-9)
quad	2,586024	2,586619	2,586619	-0,000595	4,18E-09	3,03E-11

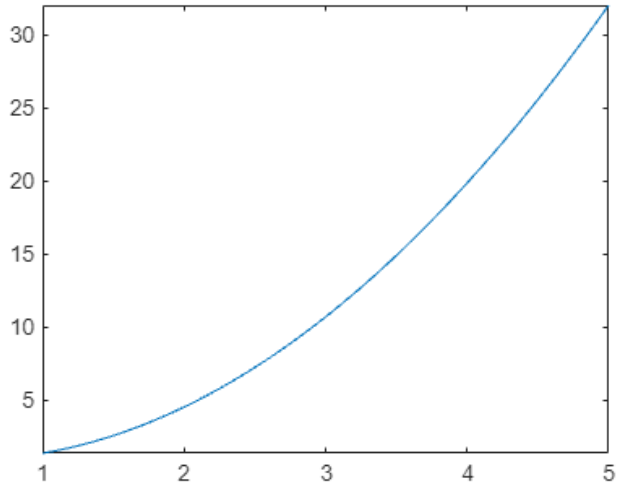
Odchylka je rozdíl mezi aproximací integrálu a skutečnou hodnotou integrálu.

4.6.2 Kvadratické funkce

Otestujeme naše m-funkce na funkci s odlišným průběhem. Funkci $f(x) = \frac{3}{2}x^2 - \frac{4}{3}\,x + \frac{5}{4}$ budeme integrovat na intervalu ⟨ 1,5 ⟩. Funkci si na tomto intervalu zobrazíme a vypočítáme určitý integrál jako rozdíl funkčních hodnot primitivních funkcí.

```
>> f2= @(x) (3/2)*x.^2 - (4/3)*x + (5/4);
>> fplot(f2,[1,5])
>> vpa(int(f2,x,1,5))
ans = 51.0
```

Tedy $\int_1^5 (\frac{3}{2}x^2 - \frac{4}{3}\,x + \frac{5}{4})dx = 51.$



Obrázek 12 – Kvadratická funkce

Následující tabulky ukazují výsledky aproximace integrálů jednotlivých numerických metod aplikované na příklad $\int_1^5 (\frac{3}{2}x^2 - \frac{4}{3}\,x + \frac{5}{4})dx$ v závislosti na počtu podintervalů n. V případě metody Monte Carlo se jako n myslí počet pseudonáhodně vygenerovaných hodnot. Pro příkaz *quad* jsou výsledky závislé na hodnotě `tol` – tolerance absolutní chyby integrace na jednotlivých podintervalech.

Tabulka 3 – Výsledky pro kvadratickou funkci

	n=10	n=500	n=2000	odchylka (n=10)	odchylka (n=500)	odchylka (n=2000)
obdelník.	50,92	50,99997	50,999998	-0,08	-3,00E-05	-2,00E-06
lichoběž.	51,16	51,00006	51,000004	0,16	6,00E-05	4,00E-06
simpson.	51	51	51	0,00	0,00E+00	-7,03E-13
monte carlo	70	49,58834	51,25484	19,00	-1,41E+00	2,55E-01
trapz	51,16	51,00006	51,000004	0,16	6,00E-05	4,00E-06

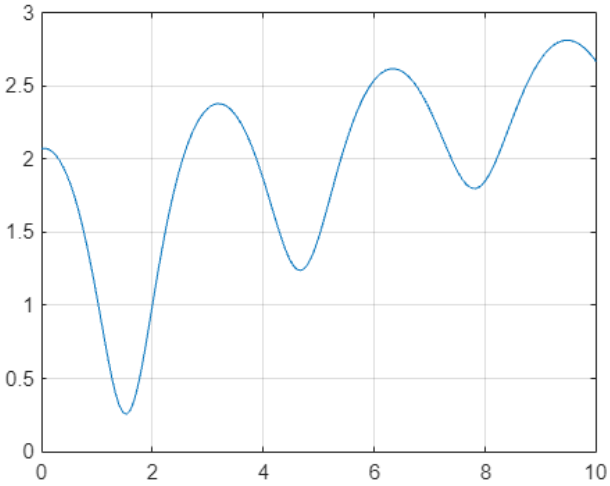
Tabulka 4 – Výsledky quad pro kvadratickou funkci

	tol=0,01	tol=10^-6	tol=10^-9	odchylka (tol=0,01)	odchylka (tol=10^-6)	odchylka (tol=10^-9)
quad	51	51	51	0	0	0

4.6.3 Implicitní funkce

Nakonec porovnáme mezi sebou m-funkce určené k řešení určitého integrálu pro implicitní funkce. Integrovat budeme implicitní funkci $x + 10\cos^2 x - y - e^y = 0$ na intervalu $\langle 0{,}10 \rangle$.

```
>> f3=@(x,y)x+10*cos(x).^2-y-exp(y)
>> fimplicit(f3,[0 10 0 3])
>> grid on
```



Obrázek 13 – Implicitní funkce

U této funkce nemůžeme integrál spočítat příkazem *int*, protože ji nelze převést do explicitního tvaru. Můžeme ale měřit uběhnutý čas pro výpočet dané metody. Doba pro numerickou integraci implicitních funkcí je obecně značně delší než u explicitních funkcí a liší se značně i podle zvolené metody.

Tabulka 5 – Výsledky pro implicitní funkci

	n=10	n=500	n=2000	čas (n=10)	čas (n=500)	čas (n=2000)
obdelník.	19,38082	19,457401	19,457392	0,22	10,5	44
lichoběž.	19,50665	19,457369	19,457389	0,41	20	82
simpson.	19,59529	19,457391	19,457391	0,34	15,3	61
monte carlo	20,88619	19,013493	19,781609	0,28	22	582
trapz	19,50665	19,457369	19,457389	0,24	26,5	630

Čas je vyjádřen v sekundách.

Závěr

Hlavním cílem této bakalářské práce bylo vytvořit m-funkce pro několik numerických metod výpočtů určitých integrálů. Tomu předcházelo jejich nastudování, sepsání definic a vytvoření ilustrativních příkladů. K numerickým metodám byly dále přidány výpočty odhadů absolutních chyb. Kromě integrace explicitních funkcí byla řešena integrace zvolené implicitní funkce.

V závěru práce byly mezi sebou jednotlivé numerické metody porovnány na třech matematických funkcích. V případě prvních dvou šlo o explicitně zadané funkce, pro které bylo možno spočítat určitý integrál jako rozdíl dvou funkčních hodnot primitivní funkce. Výsledky aproximace určitého integrálu jednotlivých metod byly závislé na hodnotě n, která značí podle zvolené metody počet podintervalů, počet náhodně zvolených funkčních hodnot nebo toleranci absolutních chyb na podintervalech. Z výsledků se jako nejpřesnější ukázalo Simpsonovo 1/3 pravidlo, a to pro obě funkce (kvadratickou a s oscilačním průběhem). Nejhůř pro stejné n dopadla metoda Monte Carlo, která pro dostatečně přesnou aproximaci integrálu potřebuje hodnotu n aspoň v řádech milionů. Lépe pak dopadla lichoběžníková metoda se stejnými výsledky jako matlabovský příkaz `trapz`. Jejich výhodou je ale aplikace na funkce, u kterých známe omezený počet funkčních hodnot. Druhá nejpřesnější pak vyšla obdélníková metoda ve formě midpoint rule. Čas potřebný pro výpočet se u všech metod pohyboval kolem 0,008-0,01 sekund, a to nezávisle na hodnotě n. Pro třetí funkci, která byla zadaná implicitně se hodnota integrálu nedala zjistit pomocí rozdílu funkčních hodnot primitivní funkce. Není ale důvod předpokládat, že by přesnost metod měla být jiná než při explicitních funkcích. Výrazně se ale lišil čas potřebný pro výpočet v závislosti na zvolené metodě a hodnotě n. Je to z důvodu použití příkazu `solve`, který počítá hodnotu neznámé z rovnice. Jako nejrychlejší se ukázala obdélníková metoda. Při vysoké hodnotě n výpočty metodou Monte Carlo a příkazem `trapz` trvaly příliš dlouho. Při menších hodnotách n se jako nejpomalejší ukázala metoda `trapz` a Monte Carlo. Metoda s příkazem `quad` nebyla při této funkci zahrnuta, protože se nedá použít pro implicitní funkce.

Literatura

[1] SCHWABIK, Štefan a Petra ŠARMANOVÁ, 1996. *Malý průvodce historií integrálu: diskretizační metody numerické matematiky*. Vyd. 2. Praha: Prometheus. Dějiny matematiky. ISBN 80-719-6038-1.

[2] ZAHRÁDKA, Jaromír, 2014. *Diskrétní matematika pro SII: diskretizační metody numerické matematiky*. Pardubice: Univerzita Pardubice. ISBN 978-80-7395-841-1.

[3] BUDINSKÝ, Bruno a Jura CHARVÁT, 2000. *Matematika I*. Vyd. 2. Praha: České vysoké učení technické. ISBN 80-010-2174-2.

[4] The basics of Monte Carlo integration, 2020. *Towards Data Science* [online]. [cit. 2024-04-03]. Dostupné z: https://towardsdatascience.com/the-basics-of-monte-carlo-integration-5fe16b40482d

[5] MATLAB, last edited on 25 November 2023. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2023-12-06]. Dostupné z: https://en.wikipedia.org/wiki/MATLAB

[6] ZAHRÁDKA, Jaromír, 2013. *Matematický seminář - MATLAB*. Pardubice: Univerzita Pardubice. ISBN 978-80-7395-691-2.

[7] MAREK, Jaroslav, Karel PASTOR a Alena POZDÍLKOVÁ, 2021. *Vysokoškolská matematika- výklad, řešené příklady a cvičení*. Univerzita Pardubice. ISBN 978-80-7560-373-9.

[8] THE MATHWORKS, INC., c1994-2024. *MathWorks Help Center* [online]. [cit. 2024-04-16]. Dostupné z: https://www.mathworks.com/help/matlab/index.html?s_tid=hc_panel