Univerzita Hradec Králové
Přírodovědecká fakulta
Katedra kybernetiky

Numerické metody ve výuce informatiky na střední škole

Bakalářská práce

Autor: Vojtěch Gärtner

Studijní program: B1801 / Informatika

Studijní obor: Bc. Učitelství – všeobecný základ
Informatika se zaměřením na vzdělávání
Historie se zaměřením na vzdělávání

Vedoucí práce: doc. RNDr. Štěpán Hubálovský, PhD.

Hradec Králové červenec 2017

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně pod vedením vedoucího práce a uvedl jsem všechnu použitou literaturu, ze které jsem vycházel.

V Hradci Králové dne Jméno a příjmení

Poděkování

Tímto bych chtěl poděkovat především svému vedoucímu práce doc. RNDr. Štěpánu Hubálovskému, PhD. za jeho cenné rady, připomínky a ochotu při spolupráci.

Anotace

GÄRTNER, Vojtěch. *Numerické metody ve výuce informatiky na SŠ*. Hradec Králové, 2017. Bakalářská práce na Přírodovědecké fakultě Univerzity Hradec Králové. Vedoucí práce Štěpán Hubálovský. 67 s.

Cílem bakalářské práce je přenést problematiku numerických metod do výuky programování na střední škole. V rámci teoretické části je formou literární rešerše zpracována teorie k numerickým metodám a dále základní principy výuky programování na střední škole. V rámci praktické části práce jsou navrženy dle didaktických zásad pracovní listy pro výuku programování z oblasti numerických metod.

Klíčová slova

Numerické metody, Gaussova eliminační metoda, Iterační metoda, Metoda půlení intervalu, Lichoběžníková metoda výpočtu integrálů, programování, Visual Basic for Application.

Annotation

GÄRTNER, Vojtěch. *Numerical methods in learning of informatics at secondary schools*. Hradec Králové, 2017. Bachelor Thesis at Faculty of Science University of Hradec Králové. Thesis Supervisor Štěpán Hubálovský. 67 p.

The aim of this bachelor's thesis is to carry over the problem of numeric methods into teaching programming on secondary schools. In research part of this work there is theoretical part to numerical methods worked out and basic principles of teaching on secondary schools. In practical part, suitable worksheets from numeric methods are suggested for teaching programming.

Keywords

Numerical methods, Gaussian eliminaton, Iteration method, Bisection method, Trapezodial method, computer programming, Visual Basic for Application.

Obsah

Úvod

Ve své bakalářské práci se budu věnovat numerickým metodám a jejich využití ve výuce programování.

Hlavním cílem této bakalářské práce je vytvořit výukový materiál pro studenty středních škol, který bude zaměřen na výuku numerické matematiky v předmětu programování. Vedlejším cílem této práce by měla být příprava žáků na studium na vysoké škole, kde se tato problematika na příslušných oborech vyučuje. Dalším cílem je převedení programování do praktických ukázek a tím motivovat žáky k dalšímu rozvoji svých programovacích dovedností. Tato práce je rozdělena na dvě hlavní části, teoretickou a praktickou.

V teoretické části této práce se pokusím co nejlépe přiblížit význam a využití numerických metod. Dále pak v této části představím základní principy algoritmizace a programování a jeho implementaci do jazyka Visual Basic for Application.

V praktické části bakalářské práce připravím pracovní listy, které by měly posloužit žákům při studiu této problematiky. Pracovní listy budou vycházet hlavně z teoretické části bakalářské práce.

Součástí praktické části je zpracovaná teorie tvorby pracovních listů.

Pracovní listy se skládají ze dvou částí, v první části jsou pracovní listy pro žáky, druhá část je pro učitele.

Výuku této problematiky bych zařadil do druhé poloviny (tj. do třetího a čtvrtého ročníku) studia na střední škole, neboť pro pochopení bakalářské práce se přepokládá základní znalost programování, ale i jistá pokročilost ve studiu matematiky.

Literaturu použitou v bakalářské práci jsem vybíral dle vlastního uvážení a její použití bylo klíčové. Snažil jsem se využít materiály, které mi byly k dispozici během studia. Výběr literatury byl ovšem

velmi obtížný, neboť jsem potřeboval získat znalosti z více oborů najednou, aby práce byla vytvořena dle představ mých i mého vedoucího. Bohužel se mi nepodařilo nalézt literaturu, která by se zabývala propojením programování s numerickými metodami.

Celkově se mi s literaturou pracovalo poměrně složitě, neboť jsem musel prostudovat publikace zabývající se vysokoškolskou matematikou.

Pro toto téma jsem se rozhodl na základě absolvování předmětu numerická matematika. Vyučujícím tohoto předmětu byl doc. RNDr. Štěpán Hubálovský, PhD., jehož přístup k numerickým metodám mě zaujal, a proto jsem se rozhodl svou bakalářskou práci psát pod jeho vedením. Na předmětu se mi líbilo praktické přenesení problematiky programování do praxe, kdy si je student schopen naprogramovat vlastní početní program, který vyřeší individuální matematické problémy.

Tuto bakalářskou práci zpracovávám dle znalostí a zkušeností získaných během svého studia na Univerzitě Hradec Králové.

1 Teoretická část

V teoretické části této bakalářské práce se pokusím představit obor numerické matematiky se zaměřením na teorii numerických metod. Tato část bude obsahovat stručnou historii oboru, dále důvody vzniku a také využití oboru v praxi. Dále se zaměřím na teorii samotných numerických metod. Součástí této části bude i propojení numerických metod a informatiky. Následující kapitoly budou obsahovat vybrané numerické metody. Jednotlivé metody budou popisovány spíše z matematického hlediska. Jejich využití v informatice bude zařazeno do části praktické.

V další kapitole bude rozebrána základní teorie programování a algoritmizace. Budou to ovšem pouze základy, neboť ke studiu programování numerických metod se již předpokládá jistá znalost těchto oborů. Následně připravím kapitolu obsahující informace o programovacím jazyku Visual Basic for Application, který jsem vybral pro použití k dosažení cílů této bakalářské práce.

1.1 Numerická matematika

Numerická matematika, nebo také výpočtová matematika, je velmi rozsáhlá oblast matematiky, která zpracovává matematické modely pomocí výpočetní techniky. Je v podstatě přemostěním z teoretické matematiky do matematiky v podobě jedniček a nul, čímž získává mnohem přesnější výsledky v kratším čase, než by bylo možné získat běžným způsobem. (Feistauer, Kučera, 2014)

Historie numerické matematiky sahá až do období přibližně 18. až 16. století před naším letopočtem. V tomto období se babylonští počtáři snažili co nejlépe vyjádřit číslo $\sqrt{2}$. Základy současné numerické matematiky byly položeny v 18. a 19. století. V těchto dobách působili

významní matematici a fyzikové jako byli sir Isaac Newton, Leonhard Paul Euler či Johan Carl Fridrich Gauss.

Nový rozmach nastal ve druhé polovině 20. století a byl spojen hlavně s rozvojem výpočetní techniky. Díky současným počítačům můžeme řešit i velmi komplikované matematické úlohy. Zároveň vznikají nové úlohy, a to právě díky používání počítačů, které nám s řešením pomáhají.

Numerická matematika se vyskytuje v mnoha oblastech běžné lidské činnosti – například v technice, přírodních vědách, ekonomice, finančnictví ale i v lékařství. Základy numerické matematiky jsou dány výsledky z matematických analýz, algebry, funkcionální analýzy, topologie a dalších.

Numerická matematika pracuje s velmi širokou oblastí matematických záležitostí. Patří sem například velké soustavy lineárních rovnic, výpočty vektorů matic, dále pak výpočty vlastních čísel, aproximace funkcí, systému nelineárních rovnic atd.

Praktické využití numerické matematiky je velmi široké. Lze ji využít pro výpočty trajektorií kosmických lodí, nosnosti stavebních konstrukcí, používá se i pro makro či mikroekonomické modely a mnoho dalších. (Feistauer, Kučera, 2014)

Pro řešení pomocí numerické matematiky je nutné nejprve problém formulovat ve tvaru numerické úlohy. Musíme tedy vytvořit úlohu tak, aby její vstupní i výstupní data byla v číselné podobě. Následuje aplikování numerické metody, což je tedy vlastně postup řešení numerické úlohy, čímž se dostáváme k algoritmu numerické metody. Algoritmus numerické metody je přesný postup, kterými realizujeme numerickou metodu. Zjednodušeně lze algoritmus popsat jako posloupnost činností, které ke vstupním datům jasně a přesně přiřadí odpovídající výstupní

hodnoty. Celý algoritmus musí být samozřejmě správně vytvořen dle pravidel, aby ho dokázal daný počítač zpracovat.

Před tím, než začneme pracovat s numerickými metodami, musíme provést tzv. *preprocessing*. Tímto procesem si připravíme soubory výsledných údajů, kterých chceme docílit – zpřístupníme množství výsledků pro naše následné využití. Metody, které toto provádějí, se nazývají *postprocessing*. Jednou formou postprocessingu je vizualizace výsledků.

Řešení reálných problémů pomocí numerických metod nám nikdy nepomůže získat přesný výsledek, musíme se spokojit s přibližným řešením, které je zatíženo chybami. Naším cílem je ovšem usměrnit výpočet, abychom minimalizovali celkovou chybu. Nejdůležitější je vyvarovat se hrubých lidských chyb, které vyplývají ze špatného pochopení problému či jiných „běžných“ chyb. (Feistauer, Kučera, 2014)

1.1.1Numerické metody

Numerické metody jsou postupy při řešení numerických úloh. Numerická úloha je popis vztahu mezi vstupními a výstupními daty. (Kulička, 2016).

Hlavním úkolem numerických metod je zjednodušení výpočtů a zkrácení času při řešení numerických úloh. Mnohé z problému nejsou ani analyticky řešitelné, či je jejich nalezení obtížné natolik, že ho nelze získat reálnými výpočty. Proto musí numerické metody pracovat i s přibližnými řešeními. Při použití přibližných výsledku ovšem vznikají chyby. Tomuto problému se ovšem věnuje jedna z následujících kapitol.

Základním principem numerických metod je tzv. diskretizace proměnných. To znamená, že spojité řešení nahradíme posloupností oddělených, izolovaných bodů. Na tomto principu pracuje většina numerických metod. (Kučera, 2006)

Numerické metody jako takové vznikaly hlavně po nástupu počítačů do tohoto oboru. Nevyužívaly se ovšem pouze běžné počítače, ale i upravená výpočetní zařízení, které vznikala pouze pro účely numerické matematiky. (Maroš, Marošová, 2006)

1.1.2 Informatika a Numerické metody

Jak bylo již nastíněno v úvodu, numerické metody jsou používány hlavně v kooperaci s výpočetní technikou. Informatika je prostředníkem výpočtů. Je tedy nutné zvolit vhodný software i hardware. Když se zaměříme na hardware, je nutné mít spolehlivou počítačovou sestavu s výkonným procesorem a mezipaměťmi, neboť jednotlivé výpočty mohou zabrat výpočetní kapacitu celého zařízení. I tak mohou jednotlivé výpočty zabrat více času.

Co se týče softwaru, existuje mnoho programů pro jednotlivé druhy výpočtů. Jednotlivé vědy mají své programy, které pracují s jasně připravenými algoritmy a mohou být specializovány pouze na jednotlivé funkce či metody numerické matematiky. Tyto programy velmi často vznikají v univerzitních prostředích. (Feistauer, Kučera 2014)

1.1.3 Chyby v numerických metodách

Základem numerických metod je práce s přibližnými výsledky, proto je nutné mít představu, jaký je rozdíl mezi řešením úlohy a řešením získaným pomocí numerické metody. Tyto chyby mohou vznikat z nejrůznějších důvodů. Ať už chybou matematického modelu, či chybou mezi vstupními daty. Problém může nastat i aplikováním numerické metody. (Kučera, 2006) Tyto chyby bychom mohli rozdělit do několika skupin:

Chyba matematického modelu

Tato chyba vzniká velmi často zjednodušováním původní úlohy, či zanedbáváním veličin, které nemají nijak velký vliv na matematický model, ale díky jejich vynechání vznikají nepřesnosti matematického modelu. Tuto chybu lze vypočítat. Získáme ji rozdílem mezi řešením reálného problému a řešením matematického modelu. Tato chyba vypovídá o přesnosti matematického modelu.

Chyba zaokrouhlovací

Zaokrouhlovací chyba patří mezi nejčastější. Je způsobena realizací algoritmu v počítači a také nepřesným prováděním aritmetických operací.

Chyba aproximace

Chyba aproximace vzniká při aplikaci přibližné metody na matematický model, neboli nemáme naprosto ideální metodu, proto použijeme metodu zástupní, která sice funguje, ovšem vytváří nám rozdíly mezi řešením matematické úlohy a řešením numerické úlohy.

Chyba ve vstupních datech

Vstupní chyby jsou již méně časté a jsou způsobeny osobou zadavatele. Mohou ale vzniknout již při měření či vytváření úlohy. Jednou z možností je také špatná reprezentace čísel, která zadáváme do počítače.

Chyba numerické metody

V některých případech lze použít i nepřesnou metodu pro řešení specifického matematického modelu, tím se ovšem dopouštíme chyby numerické metody.

Úspěch numerického problému záleží také na volbě vhodného algoritmu. V některých případech mohou i malé nepřesnosti ve vstupních datech generovat velké chyby během výpočtů. K tomuto problému se vztahuje otázka tzv. stability algoritmu, která určuje, zda je algoritmus vhodný pro daný problém či nikoli. (Hasík, nepublikováno) Z toho vyplývá, zdali je algoritmus stabilní, nebo je použitá metoda málo citlivá na poruchy v datech a na vliv zaokrouhlení během výpočtu. Stabilitu lze studovat pomocí nejrůznějších numerických algoritmů. (Čermák, nepublikováno)

1.2 Vybrané numerické metody

V této kapitole si představím čtyři numerické metody. Nejprve je stručně charakterizuji a popíši z matematického hlediska. Informace se budou týkat především využití metod. Dále uvedu výhody a nevýhody daných metod.

1.2.1 Iterační metoda

Iterační metody se používají v případech, kdy nám postačí pouze zaokrouhlený výsledek na předem daný počet desetinných míst. Metoda je založena na konvergenci posloupnosti k hledané hodnotě. Její hlavní výhodou je, že neklade vysoké nároky na paměť počítače a její využití je poměrně univerzální. Má široké využití, ať už pro řešení lineárních rovnic, ale i pro zpřesnění výsledků, které jsme získali již dříve. (Mošová, 2003)

Nyní se pokusím vysvětlit princip iterační metody. Základní myšlenkou konstrukce této metody je vyjádření neznámé (x) z každé rovnice soustavy $A\vec{x} = \vec{b}$ v závislosti na zbývajících neznámých. Soustava $A\vec{x} = \vec{b}$ se tak transformuje ve tvar $\vec{x} = H\vec{x} + \vec{g}$.

Vlastní iterační proces probíhá tak:

- Zvolíme počáteční iteraci – $\vec{x}^{(0)}$
- Prostřednictvím iterační formule $\vec{x}^{(k+1)} = H\vec{x}^{(k)} + \overrightarrow{g}$
- Ukončíme proces – určíme počet opakování, nebo vytvoříme ukončovací podmínku
- (Ukončení pomocí podmínky získáváme novou hodnotu – hodnotu neznámé – δ)
- Na konec odhadneme chybu $(k+1)$ -ni interace
- $\left|\vec{x}^{(k+1)} - \vec{x}\right| < \varepsilon$

- tím vyjadřujeme přesné řešení x s chybou ε

Pokud se nám během výpočtů změní matice H či vektor g hovoříme o tzv. nestacionárním iteračním procesu. V opačném případě se jedná o stacionární iterační proces.

Iterační metody jsou dále děleny podle svých autorů. Mezi ty nejznámější patří Jacobiova iterační metoda, dále pak Gaussova-Seidelova metoda atd. Tyto metody jsou již konkrétní a užívají se na daném matematickém problému. (Mošová, 2003)

1.2.2 Metoda půlení intervalu

Tato metoda se využívá k hledání přibližného řešení rovnice. Je to jedna z nejjednodušších metod. Toto řešení můžeme nalézt s libovolnou (uživatelem zadanou) přesností, ovšem zaleží také na výkonu počítače. Tato metoda může být ukončena třemi způsoby:

- Některý ze středů rovnice je kořenem rovnice
- Délka intervalu klesne pod minimální zadanou hranici. (MZH)
- Překročíme zadaný počet opakování daného algoritmu

Hlavními výhodami této metody je její jednoduchost, snadno se můžeme dopočítat počtu opakování před zahájením metody. Jednou z dalších výhod je také to, že jsou-li dodrženy předpoklady metody, pak metoda vždy konverguje k některému z kořenů v daném intervalu. (Čihák, 2012)

Odhad počtu opakování

- Pro zahájení této metody je nutné nejprve nalézt interval, který by odpovídal vzorci: $f(a) \times f(b) < 0$.

- Poté můžeme tvrdit, že vzdálenost středu p_1z intervalu $\langle a, b\rangle$ od kořene p ležícího v tomto intervalu platí: $|p_1 - p| \leq \frac{b-a}{2}$
- Každá následující iterace zmenší dálku daného intervalu na polovinu a z toho vyplývá: $|p_1 - p| \leq \frac{b-a}{2^n}$
- Pokud ovšem zjistíme, že $\frac{b-a}{2^n} < MZH$
- Z této nerovnice můžeme určit počet iterací k dosažení předem dané přesnosti: $\frac{b-a}{MZH} < 2^n \Rightarrow n > \log_2\left(\frac{b-a}{MZH}\right)$.

Před zahájením této metody je nezbytné zjisti, že funkce je spojitá a má alespoň jeden kořen. Zda má rovnice kořen zjistíme díky rovnici $f(a)\times f(b) < 0$.

Nyní tento interval rozpůlíme. Kořen se nachází v té části intervalu, ve kterém mají krajní body intervalu opačná znaménka. Pro další půlení použijeme tento podinterval. Pro každý další cyklus tedy upravíme označení krajních bodů a pokračujeme dalším půlením. Tímto způsobem pokračujeme až do nalezení kořene. (Maroš, Marošová, 2003)

Hlavními výhodami této metody je její jednoduchost, snadno se můžeme dopočítat počtu opakování před zahájením metody. Jednou z dalších výhod je také to, že jsou-li dodrženy předpoklady metody, pak metoda vždy konverguje k některému z kořenů v daném intervalu. (Maroš, Marošová, 2003)

1.2.3 Gaussova eliminační metoda

Gaussova eliminační metoda patří k nejstarším a nejrozšířenějším metodám pro výpočet soustav algebraických lineárních rovnic. Je to jedna z nejdůležitějších částí numerické matematiky. Mnoho úloh v praxi využívá právě tuto metodu, například fyzika při rozkládání veličin.

Velmi zjednodušeně lze tuto metodu pojmout jako řešení problému, kdy hodnoty dané veličiny hledáme pouze v konečném počtu bodů – čím více bude těchto bodů existovat, tím lépe. Gaussovu eliminační metodu lze zařadit mezi metody přímé. (Hasík, nepublikováno)

Hlavní myšlenkou této metody je postupné snižování počtu rovnic, které násobíme vhodnými koeficienty, čímž se nám jejich počet eliminuje a zůstane nám pouze jediná rovnice, která obsahuje pouze jednu neznámou, jejíž hodnota je poté zřejmá. Pomocí této neznáme dopočítáme ostatní ještě nevyřešené proměnné.

- Vlastní princip spočívá v tom, že z původního systému $Ax = b$ vytváříme ekvivalentně přidružené systémy $A^{(k)}x = b^{(k)}$,
- Při tomto vztahu platí – $k = 0, 1, \dots, n-1, A^{(0)} = A, b^{(0)} = b$, které mají totéž řešení.
- U této metody je nutné mít splněnou podmínku $a_{kk}^{(k-1)} \neq 0$.
- Tato metoda má poměrně krátký zápis algoritmu. Lze ho vystihnout takto:
- Urči prvek $a_{rs}^{(x-1)} \neq 0,\ r = s, , s+1, \dots, n$.
- Vyměň s- tý a r-tý řádek.
- Pro $i = s + 1, s+2, \dots, n$ odečti násobek $l_{is} = \frac{a_{is}^{(s-1)}}{a_{ss}^{(s-1)}}\ a$ s-tého řádku od i-tého řádku. Výsledkem je systém $A^{(s)}x = b^{(s)}$

Gaussova eliminační metoda je velmi přesná metoda. Problém nastává při zpracování v počítači. Je vhodné pracovat v systému, který umožňuje přesnou (symbolickou) aritmetiku ve tvaru zlomků, iracionálních čísel apod. Je-li ovšem vstup zadán desetinným vyjádřením, vznikají další zbytečné nepřesnosti. Tyto nepřesnosti se poté dále hromadí vlivem zaokrouhlovacích chyb. (Čermák, nepublikováno)

Pro potřeby programování je dobré soustavy rovnic vyjádřit jako matice, neboť ty se dají mnohem jednodušším způsobem přepsat do podoby vhodné pro zpracování počítačem.

1.2.4 Lichoběžníková metoda výpočtu integrálů

Tato metoda je jednou ze základních metod pro numerický výpočet integrálu. Jak lze z názvu odvodit, interval, který se chystáme řešit, proložíme lichoběžníkem a u tohoto lichoběžníku spočítáme jeho obsah. Lichoběžníková formule je vhodná pro výpočet lineárních polynomů (např. přímek), ale pro kvadratické (např. paraboly) již nikoli. (Hasík, nepublikováno)

Základní lichoběžníková formule

Základní lichoběžníková formule spočívá ve spojení krajních bodů pomocí přímky a následného spočítání obsahu pod touto přímkou (obsah lichoběžníku)

Základní formule:

$$S = (b - a) \times f\left(\frac{a + b}{2}\right)$$

Pro přesný výpočet je nutné mít tzv. složenou formuli.

$$S = h(0{,}5 \times f(x_1) + f(x_2) + \cdots + f(x_{n-2}) + f(x_{n-1}) + f(x_n) \times 0{,}5)$$

h v tomto případě udává délku intervalu. Tuto délku spočítáme dle vzorečku:

$$h = \frac{(b-a)}{n}$$

n je počet podintervalů.

Tato metoda spočívá v rozložení hlavního intervalu na podintervaly. V těchto podintervalech se aplikuje základní lichoběžníková formule, díky které vypočítáme obsah pod přímkou – obsah lichoběžníku.

Lichoběžníková metoda je velmi přesná pro lineární funkce. Je univerzální a dá se použít pro jakoukoli křivku. Ovšem je nutné použít dostatečný počet intervalů. (Hasík, nepublikováno)

1.3 Programování

1.3.1 Základy programování

V této kapitole bych uvedl základní principy strukturovaného programování, které je nutné znát, abychom byli schopni pracovat při tvorbě jednoduchých programů pro výpočty výše uvedených numerických metod. Tato kapitola bude vycházet hlavně z učebního textu pana doc. Hubálovského (nepublikováno) a paní prof. Milkové (2011). V následujících kapitolách bude představen jazyk *VBA – Visual Basic for Application* a také příprava Microsoft Excel pro možnost programovat ve *Visual Basic for Application*.

1.3.2 Pojem Algoritmus

Základním pojmem celého programování je *algoritmus*. Algoritmus lze zjednodušeně popsat jako přesný popis postupu při vykonávání dané úlohy. (Hubálovský, nepublikováno) V informatice je to spíše teoretický princip pro řešení problému, který se poté řeší v konkrétním programovacím jazyce. Přesná definice je dle Bartoňka (2005): *„Algoritmus můžeme rozumět jako postup, podle kterého se z dat vstupních vygenerují data výstupní. Je to v podstatě předpis pro řešení nějakého problému nebo úlohy.“* Obecně se ovšem algoritmus může nacházet v běžné lidské činnosti, například v podobě kuchařského receptu.

Algoritmus má své specifické vlastnosti, které musí splňovat. (Bartoněk, 2005) Tyto vlastnosti jsou:

Konečnost

Každý algoritmus musí mít konečný počet kroků. Tento počet ovšem může být libovolně velký.

Obecnost

Algoritmus musí být univerzální a použitelný pro více možných vstupů a vyřešit je stejným způsobem. Nelze řešit pouze jeden konkrétní problém.

Jednoznačnost

Algoritmus se dělí na kroky. Tyto kroky musí být naprosto jednoznačně a přesně definovány. Dbá se na dodržování posloupnosti kroků a musí být evidentní, který krok předcházel a který následoval. Dále musí obsahovat co se v daném kroku provede.

Výstup

Algoritmus musí mít alespoň jeden výstup – veličinu, která vytvoří odpověď pro vyřešení konkrétního problému.

Elementárnost

Algoritmus se skládá z konečného počtu jednoduchých kroků.

Dalším důležitým pojmem této problematiky je pojem *algoritmizace*. Pod tímto pojmem si můžeme představit kompletní řešení problému pomocí algoritmu. Spadá sem analýza problému, návrh způsobu řešení a sestavení vhodného algoritmu. Celý tento proces je velmi složitý a provádí se v několika etapách, jejichž výsledkem je konečné řešení, které se pak testuje, zda celý proces pracuje správně. Testování algoritmu vzorovými příklady nazýváme validace algoritmu. (Hubálovský, nepublikováno)

Algoritmus lze tvořit dvěma způsoby:

Shora dolů – řešení rozložíme na jednodušší operace, čímž se dostaneme k základním krokům.

Zdola nahoru – opačný proces, kdy ze základních kroků vytváříme kompletní operace pro zvládnutí problému.

Celý tento proces vede k tomu, abychom byli schopni napsat kompletní program v námi vybraném programovacím jazyce. Proto je nutné celou algoritmizaci upravit požadavkům jazyka, ale i počítače, na kterém bude probíhat. Jde zde především o rychlost a efektivnost celého budoucího programu.

Další důležitou součástí algoritmizace je zápis algoritmu. Ten můžeme provést více způsoby. Výběr tohoto zápisu je naprosto dobrovolný a vybírá se dle požadavků programátora. V této bakalářské práci je předveden slovní zápis a také grafický zápis pomocí vývojových diagramů. Další možnost zápisu je například pomocí pseudokódu, konkrétním programovacím jazykem, tabulkovým zápisem nebo i matematickým zápisem. (Bartoněk, 2005)

Velmi důležitým pojmem v algoritmizaci je také pojem *proměnná*. Proměnná je datový objekt, který má své jméno a hodnotu. (Bartoněk, 2005) Do proměnných se ukládají hodnoty. Proměnná charakterizuje konkrétní paměťové místo v počítači, do kterého se uloží daná potřebná hodnota. Tato proměnná může nabývat různých hodnot. Ty jsou dány tzv. datovým typem. Datové typy se dále dělí:

Jednoduchý datový typ – jsou přímo zadány v programovacím jazyce

Dále se dělí na:

Ordinální datové typy: celé číslo, logická hodnota, znak

Neordinální datové typy: reálné číslo

Složený datový typ – datový typ, jehož proměnné obsahují více hodnot. Patří sem například jednorozměrné a dvourozměrné pole. To jsou proměnné složené ze stejného počtu hodnot i stejného datového

typu. Ty jsou poté pod společným názvem pevně seřazeny v paměti počítač.

1.3.3 Základní algoritmické konstrukce

V této kapitole popíši základní algoritmické konstrukce. Zaměřím se na charakteristiku jejich funkce a využití. Součástí každé konstrukce bude ekvivalentní zápis v jazyce *VBA*. Samotný jazyk je ovšem představen až v následující kapitole.

Jednoduché příkazy

Tento příkaz slouží k uložení hodnoty do proměnné. (Milková, 2010) Tuto hodnotu lze do proměnné uložit třemi způsoby:

1. Přímo – do proměnné se zapíše a uloží hodnota daného typu, například číslo.
2. Nepřímo – hodnota, která se zapíše do proměnné je již hodnotou jiné existující proměnné, ta tuto hodnotu předá do uživatelem vybrané proměnné.
3. Ve formě výrazu

Příklad přímé metody:

Promenna1=1

Příklad nepřímé metody:

Promenna1 = Promenna2

Formou výrazu:

A=B+C

Při používání matematických výrazů používáme tzv. operátory. Můžeme je rozdělit na 3 typy:

Příkazy vstupu

Tento příkaz uživateli umožňuje načíst do proměnné hodnotu ze vstupního zařízení počítače. (Milková, 2010) Tímto program vyzve uživatele k zadání kýžené proměnné. Výhodou VBA je také to, že k načtení tímto způsobem lze využít dva způsoby. Tyto způsoby jsou dialogové okno, nebo načtení z buňky na listu Excelu, ta je upřesněna pomocí souřadnic.

*Promenna = **InputBox**(„Zadej proměnnou“)*

Příkazy výstupu

Příkaz výstupu slouží k vypsání proměnné na výstupní zařízení. Běžně tedy na monitor či display daného zařízení. (Milková, 2010) Opět lze tento proces provádět v Excelu více způsoby. Prvním způsobem je dialogové okno. Druhým způsobem je vypsání proměnné do námi určené buňky na listu Excelu.

***MsgBox**(„Promenna“)*

Strukturované příkazy

Posloupnost příkazů

Tato jednoduchá algoritmická konstrukce spočívá v tom, že příkazy, ať už jednoduché či strukturované, musí být vykonány v jednoznačně určeném pořadí. Důležitost této posloupnosti si nyní ukážeme:

```
Sub Start()
    Prikaz 1
    Prikaz 2
    ...
End Sub
```

Příkaz větvení - podmínka

Algoritmická konstrukce tohoto typu se v algoritmizaci nachází velmi často. Větvení je založeno na vložení podmínky (logického výrazu), kterámá dva výstupy (ano či ne), čímž nám vzniknou dvě větve. Tyto větve můžeme následně dále větvit. (Milková, 2010)

Existují dva typy větvení, a to větvení úplné a neúplné.

Úplné větvení

Úplné větvení znamená, že algoritmus otestuje podmínku a dále pokračuje na příkaz, který následuje. Ať algoritmus podmínku splňuje nebo nesplňuje, čeká na něho v dalším kroku nějaký příkaz. Tudíž se hodnota proměnné, která je testována, změní dle příslušného příkazu za podmínkou.

If *podmínka* ***Then***

PrikazA1

...

PrikazAN

Else

PrikazB1

...

PrikazBN

End If

Neúplné větvení

Neúplné větvení, na rozdíl od úplného, narazí na podmínku, tu splní nebo nesplní, ovšem změna proměnné, která je testována se nachází pouze v jedné z možností. (Milková, 2010)

If *podmínka* ***Then***

Prikaz1

...

PrikazN

End If

Logické výrazy

Logické výrazy jsou využívány pro tvorbu podmínky, která zajišťuje větvení daného algoritmu. Výsledky logických výrazů mohou být *ano* či *ne* (*pravda* či *nepravda*).

Logické výrazy se tvoří z aritmetických výrazů, které jsou v podobě relačních operátorů, nebo z logických operátorů. (Bartoněk, 2005) Základní logické výrazy jsou uvedeny v následujícím výčtu:

Relační operátory		**Logické operátory**	
=	rovná se	AND	konjunkce
< >	nerovná se	OR	disjunkce
>=	větší nebo rovno	NOT	negace
<=	menší nebo rovno		
>	větší		
<	menší		

Příkaz cyklu

Příkaz cyklu nám umožňuje opakované vykonávání zadaného příkazu. Mezi základní typy cyklů patří cyklus s konečným počtem opakování, cyklus s podmínkou na začátku, cyklus s podmínkou na konci. (Milková, 2010)

Cyklus s konečným počtem opakování

Tento cyklus se vykoná pouze tolikrát, kolik si určí uživatel. Je třeba zavedení řídící proměnné cyklu, která po každém jednotlivém průchodu konkrétním cyklem zvýší svou hodnotu přesně o hodnotu provedeného kroku. (Milková, 2010)

Do While *I <= N*
Prikaz
I=I+1
Loop

Cyklus s podmínkou na začátku

Tento cyklus lze považovat za základní typ cyklu. Princip spočívá v tom, že na začátku zvolíme podmínku, která určí, zda daný cyklus provede, či nikoli. V případě, že podmínka není splněna, pokračuje program k dalším příkazům programu. Pokud je podmínka splněna, proběhne daný cyklus (splní se příkazy v cyklu) a program opět pokračuje dále k dalším příkazům. (Milková, 2010)

Do While *podmínka*
    Prikaz
Loop

Cyklus s podmínkou na konci

Jak již z názvu vyplývá, na rozdíl od cyklu s podmínkou na začátku je zde podmínka umístěna na konci cyklu, což způsobí to, že daný cyklus proběhne alespoň jednou, poté narazí na podmínku, která pokud je splněna, umožnuje programu postoupit do další části algoritmu, ale pokud nikoli, je algoritmus vrácen na začátek tohoto cyklu. To znamená, že pokud není podmínka splněna, probíhá cyklus neustále dokola. (Milková, 2010)

Do
    Prikaz
Loop Until *podminka*

V části *Programování* této bakalářské práce jsou pouze základy programování a algoritmizace. Tyto základy jsou ale dle mého názoru dostačující pro práci s výše uvedenými numerickými metodami a slouží spíše pro připomenutí již ovládaných znalostí. Co se ovšem týče programování, je nutné zvolit vhodný jazyk. Výběr jazyka závisí na zkušenostech a možnostech programátora. Proto jsem zvolil jazyk *Visual Basic for Application*, který je dle mého názoru velmi snadno dostupný a také není nijak náročný. Tento jazyk jsem vybral i proto, že jsem jej používal během svého bakalářského studia.

Dle mých dosavadních zkušenostní si myslím, že je nutné mít základní znalosti o jazyku, který bude programátor používat.

1.4 Visual Basic for Aplication

Tato kapitola obsahuje informace o programovacím jazyku, který je použit v praktické části bakalářské práce. Základní algoritmické konstrukce tohoto jazyka jsou již součástí kapitoly předchozí. Představení jazyka ovšem pokládám za nutnost, neboť VBA má jako každý programovací jazyk svá jistá specifika. Jedním z těchto specifik je příprava programovacího prostředí, které je popsáno v této kapitole. Dále je zde uvedeno, jak nakládat s již vytvořeným projektem, aby byl správně uložen apod. Nyní představím základní informace o tomto jazyku.

1.4.1 Základní informace

Programovací jazyk *Visual Basic for Application* (dále jen VBA), je objektově orientovaný programovací jazyk. Vychází z jazyka Visual Basic, který byl vytvořen ve společnosti Microsoft a poprvé použit v roce 1993. Programování v jazyce VBA je velmi snadno dostupné, neboť se dá provádět v programu Microsoft Excel, který je součástí sady Microsoft Office. Microsoft Office najdeme zcela běžně na většině základních i středních škol. Programovací jazyk vyniká především svou jednoduchostí, i přes to se dokáže alespoň částečně vyrovnat programovacím jazykům jako jsou například Pascal, C a C++. (Hanák, 2011)

Ve VBA lze konstruovat i jednoduché programy. Nevýhodou je ovšem to, že tyto programy poté nelze vydat jako samostatné aplikace s příponou *.exe*. Toto je způsobeno tím, aby jazyk Visual Basic neztratil svůj význam. (Hanák, 2011)

VBA je implementovány do všech programů MS Office – Excel, Access, Word, PowerPoint, Outlook. Současná verze je tvořena verzi Visual Basic 6 a jeho přidruženým vývojovým prostředím. VBA ovšem

není tak obsáhlý jako samotný Visual Basic, což je samozřejmě nevýhodou. Ovšem umí manipulovat se všemi objekty programu Excel, jako jsou například buňky, listy a další. Dokáže také propojit jednotlivé programy této sady.

Součástí VBA jsou také tzv. dynamicky linkované knihovny, které nám umožňují využití již automatizovaných procesů a funkcí. Dále je VBA schopen vytvářet uživatelsky definované funkce, automatické procesy (Makra). Může také přistupovat k Windows API.

Vytvořený kód ve VBA je přeložen do tzv. Microsoft P-code, což je proprietární prováděcí kód, který se ukládá jako samostatný dokument. V Excelu je to soubor „.xlsx“ (s podporou maker). Tento kód se poté spouští ve virtuálním stroji hostující aplikace. (Hubálovský, nepublikováno)

Dle mého názoru je tento jazyk velmi dobrým startem do dalšího programování. Lze si v tomto jazyku vyzkoušet základní algoritmické konstrukce, od kterých se pak lze dále dostat ke složitějším programům. Výhodou je finanční nenáročnost programu a také to, že nezabírá další kapacitu disku, neboť je součásti již MS Office.

1.4.2 Příprava programu Microsoft Excel pro VBA

Aby bylo možné v programu Microsoft Excel programovat, je nutné povolit tzv. makra, toho lze docílit tím, že se aktivuje karta Vývojář, ta ovšem není aktivní v běžném nastavení. Pro její zobrazení je nutné otevřít řádek *Soubor*, dále postupovat na složku *Možnosti*, zde v řádku *Přizpůsobit pás karet* a v pravém sloupci zaškrtnout políčko *Vývojář*.

Dále je nezbytné nastavit zabezpečení maker na nejnižší úroveň. Bez tohoto kroku není možné prostředí využívat. Tento krok se provádí na kartě *Vývojář*. Zde klikneme na *Zabezpečení maker*, kde v nastavení zaškrtneme políčko *Povolit všechna makra* a *Důvěřovat přístupu*

k objektovému modelu. Tento krok je nezbytný, ale je rizikový, neboť pomocí maker se dají přenášet i viry a další nebezpečné kódy, proto je doporučeno po ukončení práce s makry uvést nastavení do původního stavu.

Makro se používá k automatizaci úloh. Je to sled příkazů, jehož cílem je automatizovat a urychlit pracovní postupy, nahradit ručně prováděné operace, při kterých mohou vznikat chyby. Makra se tvoří většinou pomocí záznamníku maker.

Důležitý je samotný Editor jazyka VBA. Ten se nám nachází na kartě *Vývojář*, kde je možné otevřít programovací prostředí pomocí ikony *Visual Basic* vlevo na *Pásu karet rychlého přístupu*, či stisknutím kombinace kláves *Alt* a *F11*.

Toto vývojové prostředí nelze otevřít samostatně, vždy je nutné mít otevřený také program MS Excel. Pokud zavřeme Excel, zavře se i vývojové prostředí. Vývojové prostředí se otevírá samostatně v novém okně.

V tomto prostředí jsou otevřena dvě podokna, která slouží ke zjednodušení obsluhy. Jsou to podokna *Project* a *Properties*. *Project* zobrazuje seznam aktuálně otevřených projektů, listů, modulů a formulářů. *Properties* zobrazují vlastnosti jednotlivých listů, sešitů a dalších.

Pro programátora jsou zpočátku nejdůležitější tzv. moduly. Do těchto modulů se píše VBA kód a je v nich také uložen. Při otevření vývojového prostředí není vytvořen žádný modul. Ten musí být nejprve vytvořen a to pomocí kliknutí na *Insert*, které se nachází v horní nabídce. Po tomto úkonu se rozbalí nabídka a z ní je nutné vybrat *Module*. Tímto krokem se v programovacím prostředí objeví bíla plocha, na kterou lze psát kód. Také se tento *Module* objeví v podokně Project.

Nyní pomocí tlačítka *Insert* je třeba otevřít *Procedure*, čímž se zobrazí dialogové okno. Zde je nutné uvést v kolonce *Name* název

procedury, kterou se uživatel snaží vytvořit. Vytvoření názvu je na uživateli. Je ovšem vhodné zvolit názvy dle základních zásad, které se dodržují při programování. Při tvorbě je nutné zaškrtnout políčka *Sub* a *Public* a potvrdit tlačítkem *OK*. Po tomto úkonu se na bílé ploše vytvoří zápis *Public Sub* a následuje Název procedury, který byl zadán ( ). Tento zápis tvoří začátek každého programu. Na *Dalším řádku* se vytvoří uzavírací řádek End Sub, který tvoří konec programu. Mezi tyto dva řádky se postupně zapisuje kód jazyka VBA. Po celém tomto procesu, je-li vykonán správně, vznikne nový program. Ten poté lze spustit pomocí ikony ve tvaru šipky zelené barvy na horní lište okna. Lze použít i klávesovou zkratku *F5*.

Specifické je i ukládání vytvořených programů. Program se ukládá společně se sešitem Excelu, ve kterém je vytvořen. Při ukládání je nutné dbát na výběr správného formátu, ve kterém bude sešit uložit. Požadovaný formát je *.xlsm*, neboli sešit Excel s podporou maker.

Celý tento postup by měl být stejný pro všechny verze programu Microsoft Excel od roku 2007 do verze aktuální, tj. verze Microsoft Excel 2016.

2 Praktická část

V této části bakalářské práce jsem vytvořil pracovní listy, které budou sloužit studentům při výuce programování na střední škole. V pracovních listech jsou shrnuty základní poznatky z teoretické části této práce, které by měli studenti pochopit a také si osvojit.

Součástí praktické části je i stručná teorie tvorby pracovních listů, ze které jsem vycházel. Dále je zde uvedeno i mé rozhodování a popis postupu při tvorbě listů.

Pracovní listy jsou rozděleny na čtyři hlavní části. První část se zabývá teorií numerických metod. Druhá část obsahuje problematiku algoritmizace a programování, aby si studenti připomněli již získané dovednosti. V další části jsou vybrané numerické metody, jejich význam a využití. Následuje jejich naprogramování. Poslední část pracovních listů je určena pro učitele, který tyto pracovní listy použije při výuce.

Využití pracovních listů v praxi vyžaduje jistou znalost celé problematiky numerických metod a také programování. Tyto znalosti žáci získají hlavně od svého učitele. Předpokládám, že jejich užití bude součástí výkladu celé této látky doprovázeno například multimediální prezentací a podobně. Rozhodně nepředpokládám, že by ve výuce postačily pouze tyto pracovní listy. Listy mají sloužit žákům k prohloubení jejich znalostí a dovedností. K tomuto účelu jsem se rozhodl navrhnout i obal těchto listů v podobě desek, které by žáci obdrželi s prvním listem. Tyto desky by poté sloužily k uchování pracovních listů a obsahovaly by informace k práci a orientaci v nich. Například legendu s vysvětlivkami a také stručný obsah těchto listů, aby byli žáci schopni tyto desky efektivně využít. Viz. příloha číslo XXIV.

2. 1 Pracovní listy

Dle Čapka (2015) je pracovní list: *„Souborem cvičení, úkolů didaktického obrazového materiálu apod.“* Slouží především k samostatnému procvičování žáka, častěji mu ovšem poskytuje podporu, při jeho práci ve škole a následně k opakování. Pracovní listy se mohou využívat i ve dvojicích nebo ve skupinách, přičemž podporují kooperaci mezi žáky. Práci s pracovním listem řadíme mezi výukové metody práce s učebním textem. V tomto procesu tedy žák samostatně pracuje a učitel koordinuje tuto činnost a následně ji kontroluje. Pracovní listy připravuje učitel na základě schopností a dovedností žáků konkrétní třídy. (Čapek, 2015)

Velmi důležitou vlastností pracovních listu je skutečnost, že neslouží k testování znalostí, ale k získávání a opakování znalostí. Neměly by tedy být hodnoceny. Jejich hodnocení může být provedeno pouze jako odměněna bonusovou známkou za výborný výkon. Práce s pracovními listy by měla probíhat ve škole. (Čapek, 2015)

Před tvorbou pracovních listů si jejich tvůrce musí stanovit cíl pracovních listů, jejich určení, dále jejich časové využití. Nedílnou součástí je také motivace studentů, volba vhodné obtížnosti závislé například na prostředí apod. Pracovní listy by se neměly odklánět od původních cílů konkrétního předmětu. Jejich obtížnost musí korespondovat s dovednostmi studenta konkrétního věku. Obtížnost úloh by měla být ovšem stupňována. (Mrázová, 2013)

V pracovních listech by se měly nacházet pouze stručné definice a teoretické znalosti, které by měl být žák posléze schopen použít v praxi, a dále úkoly s jasným a stručným zadním. Listy musí obsahovat obrázky, grafy apod. ve vhodném grafickém zobrazení. Nutná je vhodná kombinace fondů a velikostí písma, barevnost atd. Důležitou součástí jsou také

otevřené otázky k tématu. Tyto otázky nutí žáky projevit vlastní tvořivost, či formulaci vlastních myšlenek apod. (Mrázová, 2013)

Pracovní listy lze rozdělit dle jejich cílů. Pracovní listy mohou sloužit k opakování učiva, vyhledávání informací v literatuře a jiných zdrojích, procvičování, shrnutí a poukázání na souvislosti. V ideálním případě pracovní listy obsahují všechny tyto prvky.

Aby listy plnily svůj účel, je nezbytná kontrola jejich vyplnění. Tato kontrola probíhá u každého žáka zvlášť z důvodu jeho individuálního přístupu k výuce a konkrétní problematice. (Frýzová 2014)

Pracovní listy se nepoužívají pouze pro účely vzdělávání na školách, velmi často jsou k vidění například v muzeích, či galeriích apod.

2.2 Pracovní listy numerických metod

Pro tvorbu pracovních listů, jsem se rozhodl použít jako hlavní program MS Word. Považuji ho za dostupný a jednoduchý (případné úpravy by zvládl každý učitel bez větších grafických znalostí). Jeho další výhodou je možnost vkládání rovnic a speciálních matematických znaků. Tyto listy jsem vypracovával dle svých znalostí a zkušeností získaných během studia na UHK a literatury, ze které jsem tvořil i předchozí kapitolu. Dále jsem použil program „Adobe Photoshop CS6“ a „Adobe Illustrator CS6". Tyto programy mi sloužily k vytvoření grafických prvků – barevných přechodů v záhlaví a drobných vektorových ikon sloužících k označení typu úlohy. Grafy, které jsou použity v pracovních listech, jsem vytvořil pomocí programu MS Excel a diagramy jsou vytvořeny v programu Word po nainstalování doplňku „Lucidchart for Microsoft Word“. Tento doplněk je určen pro tvorbu diagramů a je pro to speciálně určen, pracuje na principu cloudového programu a je v základní verzi zdarma.

Pracovní listy jsem rozdělil obdobně jako bakalářskou práci. Kompletní pracovní listy mají za cíl podporovat žáky při studiu programování numerických metod. Dalším cílem je opakování základů programování a také propojení mezi vědními obory, čímž si studenti mohou povšimnout souvislostí mezi jednotlivými školními předměty. Mohou také získat vhled do problematiky studia na vysoké škole.

Numerické metody nejsou běžnou součástí při výuce na středních školách, objevují se až při studiu na vysoké škole. Proto jsem se snažil připravit takové pracovní listy, které dopomohou studentům k získání základních znalostí o numerických metodách a budou je moci využít pro účely předmětu programování a případně i jako zopakování a procvičení při studiu na vysoké škole.

Každý pracovní list v úvodu obsahuje základní teorii, následují obrázky, otázky a jednoduché úkoly a příklady. Obtížnost, kterou jsem volil je založena pouze na mém osobním uvážení a myslím si, že je vhodná pro žáky střední školy. Vzhledem k tomu, že tyto pracovní listy budou využívány v předmětu informatika, zařadil jsem i úkoly, k jejichž vypracování je potřeba internet. Žáci si díky tomu procvičí své schopnosti vyhledávání a třídění informací z internetu.

Grafické zpracování je uzpůsobeno věku studentů a také ekonomičnosti celé záležitosti. Předpokládám, že tyto pracovní listy by si museli studenti nejspíše tisknout sami, proto jsem se snažil vytvořit co nejméně pracovních listů a maximálně využít plochu jednotlivých listů. Listy lze vytisknou i černobíle.

Kompletní pracovní listy jsou přiloženy k této bakalářské práci. Přílohy I až XXIII.

Závěr

Hlavním cílem této bakalářské práce je tvorba pracovních listů a jejich pozdější využití v praxi, tudíž ve výuce. Mým přáním je, aby se tato skutečnost alespoň částečně proběhla. Pokud se mi podaří dokončit svá studia, určitě použiji poznatky nabité právě při tvorbě této bakalářské práce.

Jako vedlejší cíl bakalářské práce bych uvedl také nácvik tvorby didaktických pomůcek. Tato schopnost je nezbytnou dovedností každého učitele.

Tvorba pracovních listů byla pro mne velmi zajímavá, neboť jsem se snažil propojit získané znalosti během svého studia. Snažil jsem se také vcítit do potřeb studentů a rozpomenout si na svá studia na střední škole, kde jsem se již s podobnými materiály setkal.

Pracovní listy jsem prozatím nepoužil v žádné vyučovací hodině informatiky na střední škole.

Během mého studia problematiky numerických metod jsem narazil na skutečnost, že jejich propojení s programováním není tak často rozpracováno, jak bych očekával. Metody jsou povětšinou zpracovávány pouze v oboru, pro který byly vytvořeny. Nejčastěji se nachází samozřejmě v matematice, kde jsou velmi často zpracovávány pouze z matematického hlediska.

Současná verze této bakalářské práce nemusí být tou poslední, existuje nepřeberné množství numerických metod, které by se daly zpracovat pro účely programování. Variantou je její rozšíření o jiný programovací jazyk, čímž bych mohl rozpracovat i komparaci mezi jednotlivými jazyky, jejich výhodami při programování numerických metod apod.

Seznam použité literatury

BARTONĚK, Dalibor. *Teoretické základy informatiky*. Kunovice: Evropský polytechnický institut, 2005. ISBN 80-7314-059-4.

ČAPEK, Robert. *Moderní didaktika: lexikon výukových a hodnoticích metod*. Praha: Grada, 2015. Pedagogika (Grada). ISBN 978-80-247-3450-7.

ČERMÁK, Libor a Rudolf HLAVIČKA. *Numerické metody*. Vyd. 2. Brno: Akademické nakladatelství CERM, 2008. ISBN 9788021437524.

FEISTAUER, Miloslav a Václav KUČERA. *Základy numerické matematiky*. Praha: Matfyzpress, 2014. ISBN 978-80-7378-264-1.

KUČERA, Radek. *Numerické metody*. Ostrava: VŠB – Technická univerzita, 2006. ISBN 80-248-1198-7.

KULIČKA, Jiří. *Elementární algoritmy aplikované matematiky: studijní opora*. Vydání druhé. Pardubice: Univerzita Pardubice, 2016. ISBN 978-80-7395-972-2.

MAROŠ, Bohumil a Marie MAROŠOVÁ. *Numerické metody I*. Brno: Akademické nakladatelství CERM, 2003. ISBN 8021423889.

MILKOVÁ, Eva. *Algoritmy: základní konstrukce v příkladech a jejich vizualizace*. Hradec Králové: Gaudeamus, 2010. ISBN 978-80-7435-064-1.

MOŠOVÁ, Vratislava. *Numerické metody*. Olomouc: Univerzita Palackého, 2003. ISBN 8024406209.

MRÁZOVÁ, Lenka. *Tvorba pracovních listů: metodický materiál*. Brno: Moravské zemské muzeum, 2013. ISBN 978-80-7028-403-2.

VANÍČEK, Jiří. *Teoretické základy informatiky*. Praha: Kernberg, 2007. Informatika studium (Kernberg). ISBN 978-80-903962-4-1.

Další zdroje

Algoritmus pro numerické integrování – Lichoběžníková metoda. Itnetwork.cz [online]. ©2017 [cit. 2017-04-25]. Dostupné z: https://www.itnetwork.cz/algoritmy/matematicke/algoritmus-numericke-integrovani-lichobeznikova-metoda.

ČIHÁK, Michal. *Metoda půlení intervalu*. [internet]. [Hradec Králové]; [2012]. [cit. 2017-04-25]. Dostupné z: http://lide.uhk.cz/prf/ucitel/cihakmi1/.

FRÝZOVÁ, Iva. *Pracovní list nejen v přírodovědném vzdělávání. Komenský*, Brno: Masarykova univerzita, 2014, roč. 139, 01/2014, s. 48–54. ISSN 0323-0449.

HANÁK, Jan, *Programování v jazyce Visual Basic 2010*, Kralice na Hané: Computer Media, 2011.

HASÍK, Karel. *Numerické metody*. Opava: Matematický ústav Slezské univerzity v Opavě. Nepublikováno.

HUBÁLOVSKÝ, Štěpán. *Programování 1*. Hradec Králové: Katedra kybernetiky PřF UHK. Nepublikováno.

RŮŽICKOVÁ, Irena; HLAVIČKA, Rudolf. *Numerické metody*. Brno: Ústav matematiky FSI VÚT. Nepublikováno

Přílohy

Pracovní list č. 1

1 NUMERICKÁ MATEMATIKA

Numerická matematika = matematický obor zabývající se řešením matematických problémů. Je přemostěním mezi praktickou a teoretickou matematikou.

Numerické metody = základní pojem numerické matematiky

Zakroužkuj vhodný logický operátor:

	>	
Numerická matematika	=	Numerické metody
	<	

Historie oboru numerická matematika

Počátky tohoto oboru se nacházejí mezi lety 1800 – 1600 př. n. l. v Mezopotámii

- Snahy vyčíslit číslo $\sqrt{2}$.

Počátky oboru v moderní době – 18. století.

- Doba velkých matematiků – Issac Newton, Gauss, ...
- Největší rozmach numerická matematika zaznamenala během třetí průmyslové revoluce (2. pol. 20. století.) s nástupem výpočetní techniky.

Proč vznikla numerická matematika?

_____

_____

Jaké je využití Numerické matematiky?

_____

_____

Abychom mohli využít numerickou matematiku, musíme konkrétní problém formulovat ve tvaru numerické úlohy – vstup a výstup musí být v číselné podobě. Poté aplikujeme konkrétní Numerickou metodu.

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č.2

NUMERICKÁ MATEMATIKA

Při používání numerických metod mohou velmi často vznikat chyby. Můžeme je rozdělit na:

- Chyba matematického modelu

  Vzniká zjednodušováním původní úlohy či zanedbáváním veličin, které nemají nijak velký vliv na matematický model.

- Chyba zaokrouhlovací

  Je nejčastější, způsobena realizací algoritmu v počítači a také nepřesným prováděním aritmetických operací.

- Chyba aproximace

  Vzniká aplikací přibližné metody na matematický model → nemáme naprosto ideální metodu, proto použijeme metodu zástupní, která sice funguje, ale vytváří nám rozdíly ve výpočtech.

- Chyba ve vstupních datech

  Špatné měření či chybné vytváření úlohy; špatná reprezentace čísel, které zadáváme do počítače.

- Chyba numerické metody

  Při použití nepřesné metody pro řešení specifického matematického modelu.

Kdo nebo co je způsobuje nejvíce chyb?

_____

_____

_____

K tématu chyb se váže pojem stabilita algoritmu. Za pomoci internetu se pokuste zjistit, co tento pojem znamená.

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 3

NUMERICKÁ MATEMATIKA

Numerická metoda je cesta, která vede k řešení matematické úlohy = algoritmus numerické metody.
Numerická metoda má obdobné vlastnosti jako běžný algoritmus. O vlastnostech algoritmů se více dozvíš v pracovních listech v druhé části.

Kdy a kde se můžeme setkat s numerickými metodami v běžné lidské činnosti?

_____

_____

_____

Vyhledej na internetu konkrétní numerické metody a napiš jejich funkci:

_____

_____

_____

_____

_____

Seřaď následující pojmy do řady, podle toho, jak jdou za sebou:

- Matematický model
- Matematický problém
- Numerické řešení
- Numerická metoda
- Výpočty

_____ → _____ → _____ →

→ _____ → _____

3

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 4

2 PROGRAMOVÁNÍ NUMERICKÝCH METOD

Programování = proces tvorby programů, který má za úkol řešení problému. Má několik fází:

- Analýza problému a pochopení problému
- Nalezení řešení – v podobě algoritmu
- Tvorba programu v konkrétním programovacím jazyce
- Překlad z programovacího jazyka do strojového kódu, se kterým pracuje procesor (probíhá pomocí překladače)

Programovací jazyk = jazyk, kterým zapíšeme algoritmus a tím vznikne program.

Vyjmenuj programovací jazyky, které znáš:

_____

_____

_____

Programovací jazyky dělíme:

Dle způsobu překladu		Dle míry abstrakce	
Interpretované	Kompilované	Nižší	Vyšší
Jazyky překládané do strojového kódu za běhu programu.	Jazyky, které jsou nejdříve přeloženy a až pak spuštěny.	Jazyky pracující s instrukcemi pro daný procesor. (strojový kód)	Jazyky bližší uvažování člověka – jsou logické. (Basic)

Vyhledej na internetu ukázky strojového kódu a běžného programovacího jazyka a přepiš je do následujících kolonek:

Strojový kód:

_____

Běžný programovací jazyk:

_____

1

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 5

2 PROGRAMOVÁNÍ NUMERICKÝCH METOD

Algoritmus = přesný návod, či návrh postupu, kterým můžeme řešit jakoukoli úlohu.
Pokud je algoritmus znázorněn graficky, nazýváme ho vývojový diagram.

Spoj vlastnosti algoritmů s jejich vysvětlením:

Konečnost	Algoritmus lze použít na problémy stejného typu (např. sčítání), ne pouze na jeden konkrétní problém.
Determinovanost	Algoritmus má vždy alespoň jeden výstup (výsledek).
Elementárnost	Algoritmus se skládá ze základních instrukcí.
Obecnost	Algoritmus nám jasně říká, co máme provést v každém jednotlivém kroku.
Resultativnost	Algoritmus musí vždy končit (nelze vytvořit nekonečný algoritmus).

Navrhni jednoduchý algoritmus, který bude sloužit jako návod k otevření konzervy s jídlem. Dbej na dodržení výše popsaných zásad.

2

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 6

2 PROGRAMOVÁNÍ NUMERICKÝCH METOD

Základní algoritmické konstrukce se dělí na jednoduché příkazy a na strukturované příkazy. Příkazy jednoduché nemají žádnou vnitřní strukturu, naproti tomu příkazy strukturované se skládají z více dílčích příkazů a mají tedy vnitřní strukturu.

Z následujícího výčtu se pokus tyto příkazy rozdělit do správné kategorie a poté k nim doplnit jejich charakteristiku:

Příkaz Vstupu, Příkaz větvení, Příkaz cyklu, Příkaz výstupu, Cyklus s podmínkou na konci, Neúplné větvení, Cyklus s konečným počtem opakování, Přiřazovací příkaz

Jednoduché	Strukturované

Příkaz vstupu: _____

Příkaz větvení: _____

Příkaz výstupu: _____

Cyklus s podmínkou na konci: _____

_____

Neúplné větvení: _____

Cyklus s konečným počtem opakování: _____

_____

Přiřazovací příkaz: _____

3

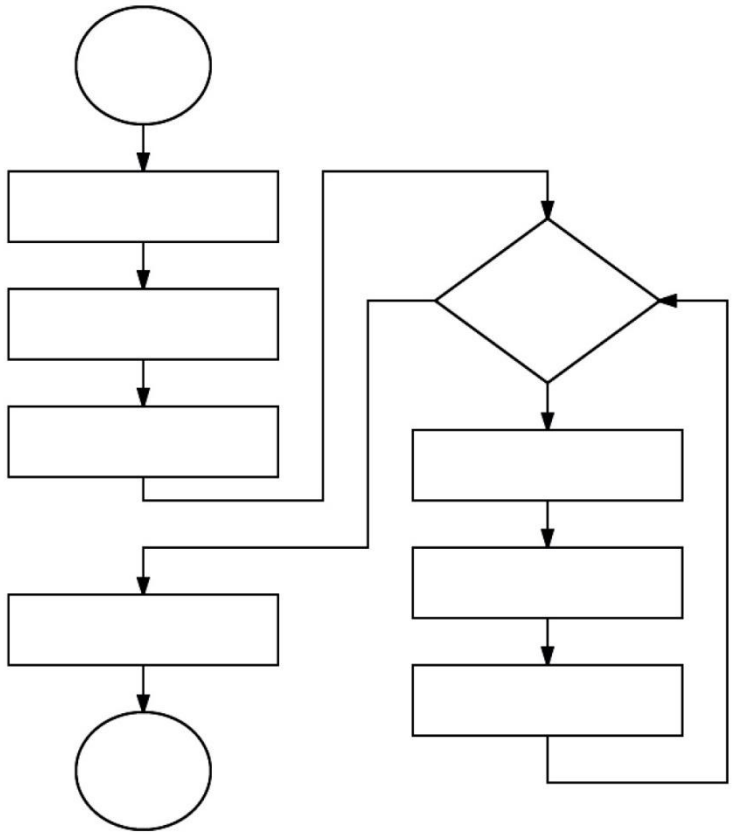
Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 7

2 PROGRAMOVÁNÍ NUMERICKÝCH METOD

Vývojový diagram = grafické znázornění jednotlivých kroků algoritmu. Obsahuje různé tvary, které mají odlišný význam a jsou navzájem propojené pomocí šipek.

Prohlédni si následující prázdný vývojový diagram a doplň ho tak, aby řešil následující úlohu: Žáci na základní škole přispívají na společný dárek pro svou třídní učitelku. Žáků bylo celkem 26 a každý přispěl dle svého uvážení. Spočítej, jaká částka se na této akci vybrala.



Vyzkoušej si navrhování vývojových diagramů v počítači. Na webových stránkách www.draw.io vytvoř návrh algoritmu, který načte dvě čísla, poté ověří, zda jsou obě kladná – a pokud ano, sečte je a vypíše výsledek. Pokud ne, informuje uživatele o tom, že číslo není kladné.

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

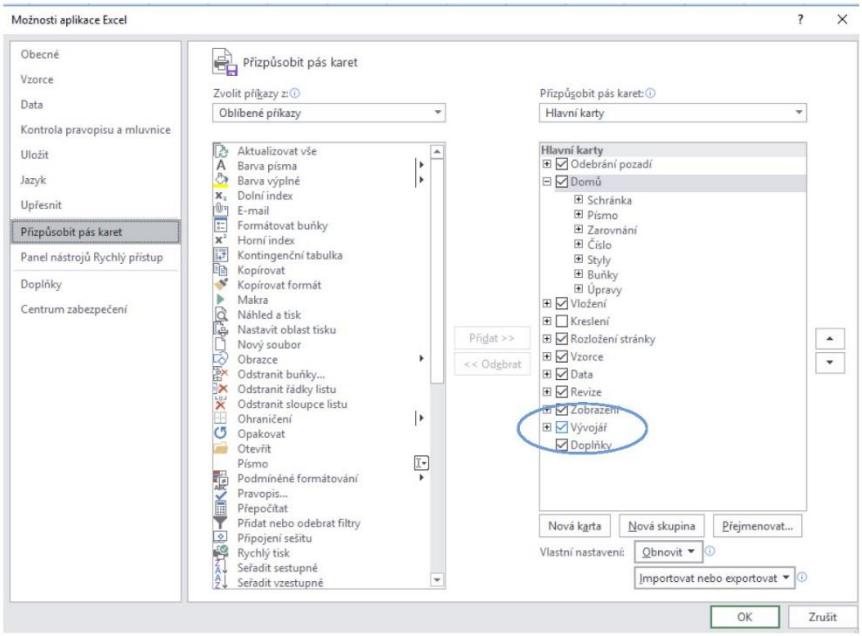
Pracovní list č. 8

2 PROGRAMOVÁNÍ NUMERICKÝCH METOD

Visual Basic for Application je programovací jazyk, který je implementovaný do všech programů Microsoft Office. Tento jazyk vychází z jazyka Visual Basic, jehož autorem je také společnost Microsoft.

Abys mohl v programu Microsoft Excel programovat, je nutné provést v něm několik nastavení. Postupuj podle následujícího návodu a poté si vytvoř svůj první program.

1. Otevři program MS Excel. Zde rozklikni nabídku pod tlačítkem „Soubor“ a otevři „Možnosti aplikace Excel“. V levé části možností otevři „Přizpůsobit pás karet“ a zde zaškrtni okénko u možnosti „Vývojář“ tak jak je uvedeno na obrázku:

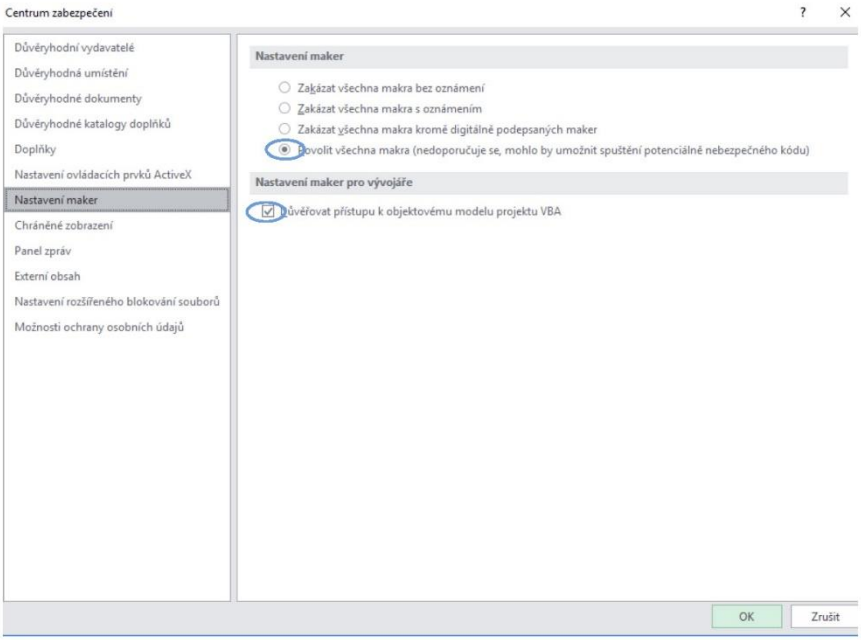


2. V dalším kroku otevři „Centrum zabezpečení“ (poslední položka v levém menu)

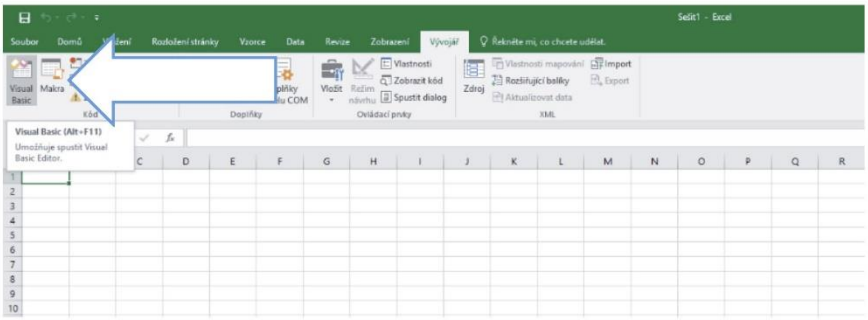
Pracovní list č. 9

2 PROGRAMOVÁNÍ NUMERICKÝCH METOD

3. Zobrazí se ti následující okno a zde označ možnosti, které jsou uvedeny na následujícím obrázku.



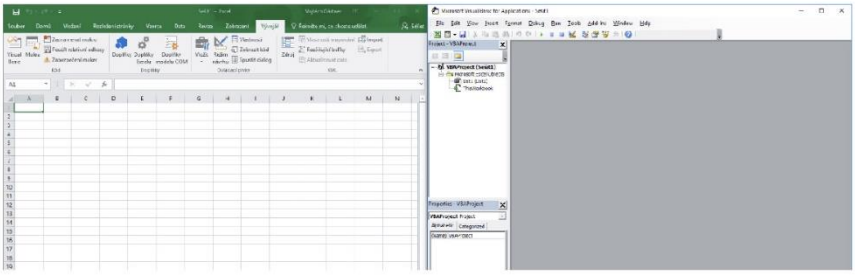
4. Nyní již můžeš rozkliknout na horním panelu kartu „Vývojář“ a zde otevřít „Visual Basic“ – viz. Obrázek.



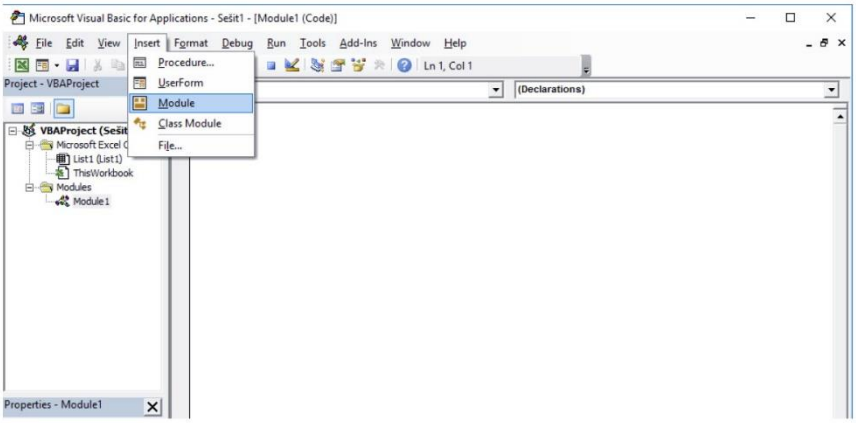
Pracovní list č. 10

2 PROGRAMOVÁNÍ NUMERICKÝCH METOD

5. Po splnění předchozího kroku se ti otevře programovací prostředí.



6. Nyní v horním panelu klikneme na tlačítko „Insert“ a v nabídce vybereme možnost „Module“. Tím nám šedá plocha změní barvu a zde již můžeme začít psát kód našeho programu.



Důležité je při ukládání souboru Excel zvolit v nabídce „Uložit jako typ:“ – „Sešit excel s podporou maker“ .

Vytvořte následující program, který sečte dvě čísla a vypíše výsledek:

```
Sub Scitani()
   A = 1 * InputBox("Zadej A")
   B = 1 * InputBox("Zadej B")
   C = A + B
   MsgBox (C)
End Sub
```

Pracovní list č. 11

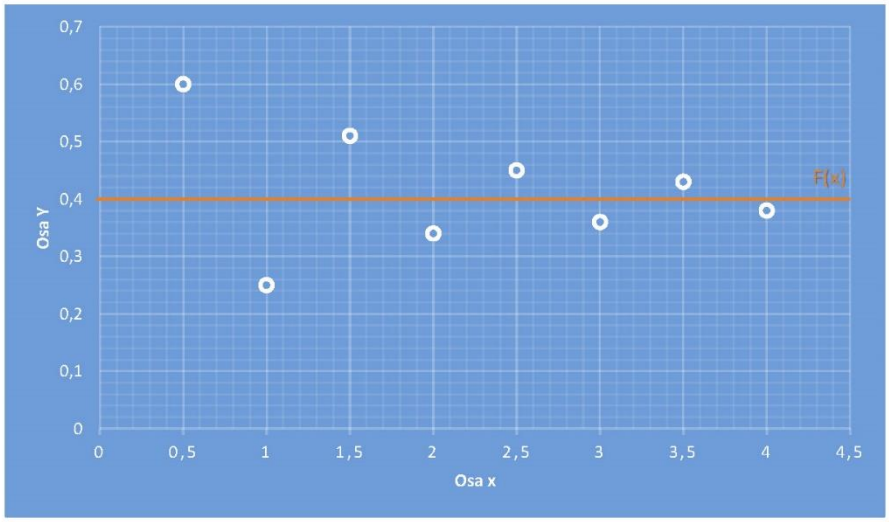
3 ITERAČNÍ METODA

Iterační metoda je jednou ze základních numerických metod. Používá se pro řešení lineárních rovnic či zpřesňování již získaných výsledků. Má ale i mnoho dalších využití. Tato metoda je založena na konvergenci posloupnosti k hledané hodnotě, což znamená, že každým jednotlivým výpočtem se přibližujeme k výsledku.

V této metodě lze použít dvou podmínek:

- konečný počet opakování
- přesnost, se kterou máme hledaný výsledek najít.

Graf metody:



Iterační proces spočívá v neustálém opakování použití funkce, čímž se tento proces snaží přiblížit výsledku. Na obrázku lze vidět, jak se po každém cyklu algoritmu této metody vždy výsledek přibližuje dané hledané funkci „F(x)“. Výsledku ovšem nemusí být dosaženo s naprostou přesností, proto v algoritmu musíme použít podmínky, který celý tento proces ukončí.

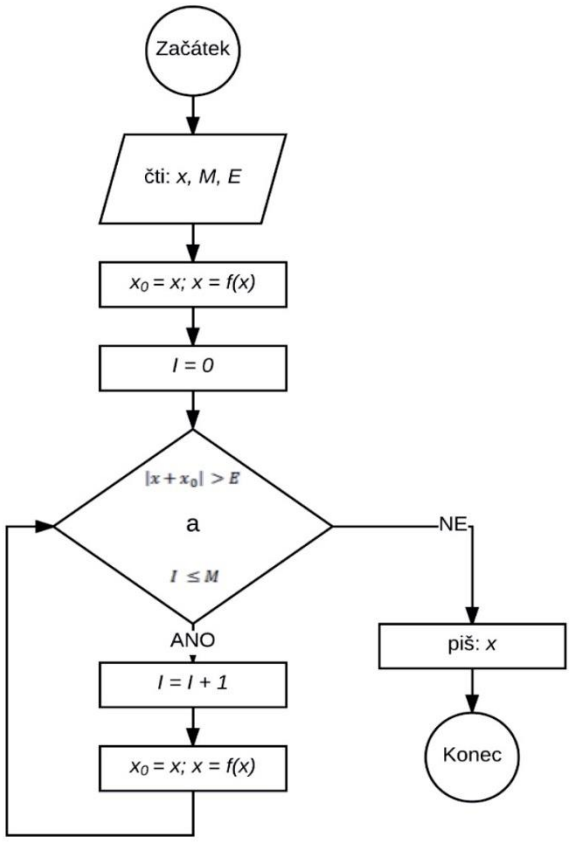
Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 12

3 ITERAČNÍ METODA

Prohlédni si algoritmus a vývojový diagram iterační metody:

1. Načteme si hodnotu bodu, který hledáme v dané funkci, tu uložíme do proměnné x, další proměnná M nám zastupuje maximální počet iterací a proměnná E vyjadřuje přesnost výsledku.
2. Do pomocné proměnné x_0 uložíme hodnotu x a do x nahrajeme hodnotu funkce, kterou počítáme.
3. Vytvoříme si proměnou I, která nám bude sledovat počet iterací.
4. omocí složené podmínky provedeme iterační proces, který je ukončen dosažením maximálního počtu iterací, nebo dosažením kýžené přesnosti.
5. Vypíšeme hodnotu x, tedy výsledek



V jazyku Visual Basic sestav program, který pomocí iterační metody spočítá funkci $f(x) = \cos x$. Inspirovat se můžeš v předchozím algoritmu.

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 13

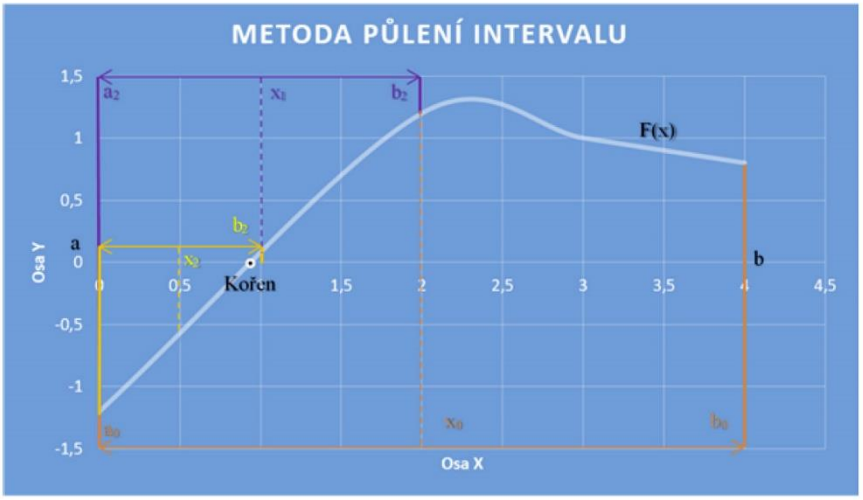
4 METODA PŮLENÍ INTERVALU

Metoda půlení intervalu je jednou z jednodušších numerických metod a využívá se hlavně pro hledání kořenů rovnice. Tato metoda pracuje s intervaly, které dělí na půl, a následně zkoumá, ve kterém ze dvou částí intervalu se nachází daný kořen. V následném kroku už pracuje pouze s jednou části původního intervalu, kterou dále půlí. Tento proces probíhá, dokud není nalezen kořen.

V praxi se ovšem velmi často pracuje pouze s přibližným výsledkem. V programech se tedy velmi často používají podmínky. Lze použít podmínku s konečným počtem opakování, či podmínku která nám sleduje přesnost, se kterou chceme daný kořen najít. Z toho vyplývá že tato metoda může být ukončena jednou ze tří možností:

- Některý ze středů rovnice je kořenem rovnice
- Délka intervalu klesne pod minimální zadanou hranici (MZH)
- Překročíme zadaný počet opakování daného algoritmu

Graf metody:



Metoda nám rozděluje danou funkci na dvě stejné poloviny. V každé polovině následně zjišťuje existenci kořene. Metoda zjistí, ve které polovině se kořen nachází a daný interval opět rozpůlí a celý proces se opakuje. Tento proces je ukončen nalezením kořene či podmínkou. Na obrázku jsou naznačeny pouze první tři iterace. Opakování procesu ovšem řídí programátor.

1

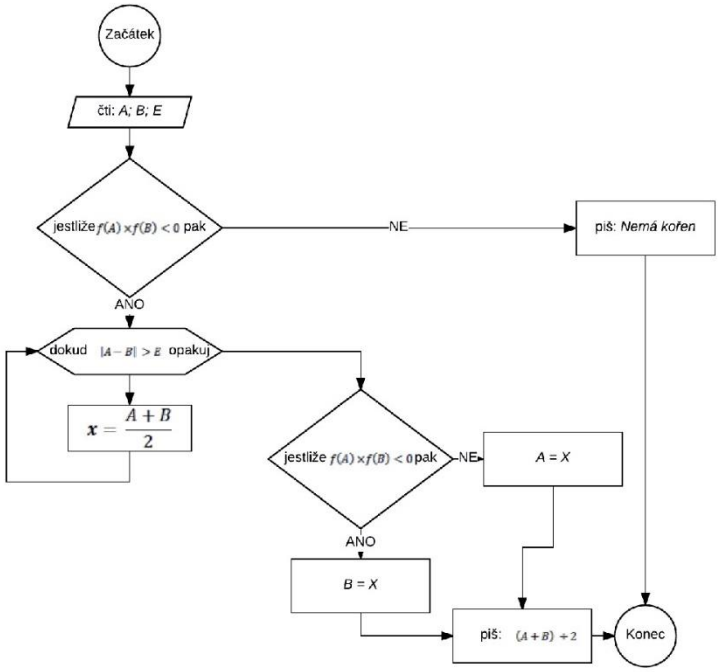
Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 14

4 METODA PŮLENÍ INTERVALU

Prohlédni si algoritmus a vývojový diagram metody půlení intervalu:

1. Po zahájení tohoto algoritmu načteme funkční hodnoty krajních bodů intervalu – $f(A)$ – A, $f(B)$ – B; dále zvolíme parametr E – tj. přesnost se kterou chceme najít daný kořen.
2. Pomocí podmínky ověříme, zda v daném intervalu existuje alespoň jeden kořen. Není-li tato podmínka splněna, program na to upozorní a algoritmus ukončí.
3. Následuje cyklus, který půlí daný interval, čímž hledá kořen funkce. Tento cyklus je ukončen při dosažení potřebné přesnosti.
4. V posledním kroku se vybere vhodný kořen rovnice, který se vypíše. Tento kořen je naším výsledkem.



V jazyku Visual Basic vytvoř program, který metodou půlení intervalu vypočte kořen rovnice $f(x) = \cos x - x$.

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 15

5 GAUSSOVA ELIMINAČNÍ METODA

Tato metoda slouží pro výpočet soustavy lineárních rovnic. Je to metoda velmi přesná. Je jednou z nejstarších a nejpoužívanějších numerických metod.

Při práci s Gaussovou eliminační metodou je nutné si soustavu rovnic upravit do maticového tvaru. Gaussova eliminační metoda spočívá v úpravě matice na tzv. schodovitý tvar. Schodovitý tvar znamená, že danou matici řešíme tak abychom jednotlivými výpočty eliminovali jednotlivé prvky matice a zůstali nám pouze výsledky na diagonále dané matice.

Ukázka:
Naším cílem je pomocí vhodných úprav, jako je násobení řádků nenulovým číslem a přičtení řádku nebo jeho násobku k jinému řádku, získat matici ve tvaru:

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & A \\ 0 & 1 & 0 & B \\ 0 & 0 & 1 & C \end{array}\right) \rightarrow \text{Schodovitý tvar}$$

$$\left(\begin{array}{ccc|c} 1 & 2 & 1 & 4 \\ 3 & 2 & -2 & 3 \\ 2 & 5 & -5 & 2 \end{array}\right)$$

- (-3) násobek prvního řádku přičteme k řádku druhému a ten nám vynuluje první člen v druhém řádku.
- Po této úpravě vynásobíme druhý řádek číslem (-2) a sečteme s třetím řádkem.
- $\left(\begin{array}{ccc|c} 1 & 2 & 1 & 4 \\ 3 & 2 & -2 & 3 \\ 2 & 5 & -5 & 2 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 2 & 1 & 4 \\ 0 & -4 & -5 & -9 \\ 2 & 5 & -5 & 2 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 2 & 1 & 4 \\ 0 & -4 & -5 & -9 \\ 0 & 1 & -7 & -6 \end{array}\right) \sim$
- Nyní jsme vynulovali první sloupec a obdobným způsobem pokračujeme dál.

$$\sim \left(\begin{array}{ccc|c} 1 & 2 & 1 & 4 \\ 0 & 1 & -7 & -6 \\ 0 & -4 & -5 & 9 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 2 & 1 & 4 \\ 0 & 4 & -7 & -6 \\ 0 & 0 & -33 & -33 \end{array}\right) \sim \left(\begin{array}{ccc|c} 1 & 2 & 1 & 4 \\ 0 & 1 & -7 & -6 \\ 0 & 0 & 1 & 1 \end{array}\right) \sim$$

- Po úpravách získáme řešení.

$$\sim \left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array}\right)$$

1

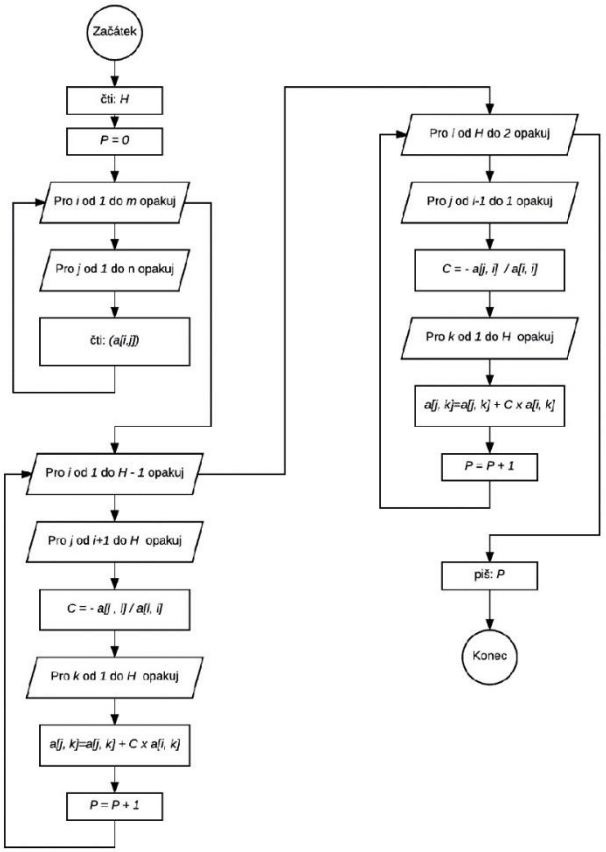
Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 16

5 GAUSSOVA ELIMINAČNÍ METODA

Prohlédni si algoritmus a vývojový diagram GEM:

1. Na počátku této metody vytvoříme proměnou H. H nám určuje počet prvků v diagonále této matice.
2. Vytvoříme proměnnou P, která nám bude počítat kolikrát proběhnou jednotlivé eliminace.
3. Následně načteme matici.
4. V dalším kroku vytvoříme dva cykly, které postupně vypočítají jednotlivé řádky matice a výsledky jednotlivých rovnic, které zastupují tyto matice.
5. Program nám na konci vypíše počet eliminací. Matice bude mít na hlavní diagonále výsledky, ostatní prvky budou rovny nule.



Sestav program, který vygeneruje náhodnou čtvercovou matici o velikosti H a tu poté vyřešte pomocí GEM.

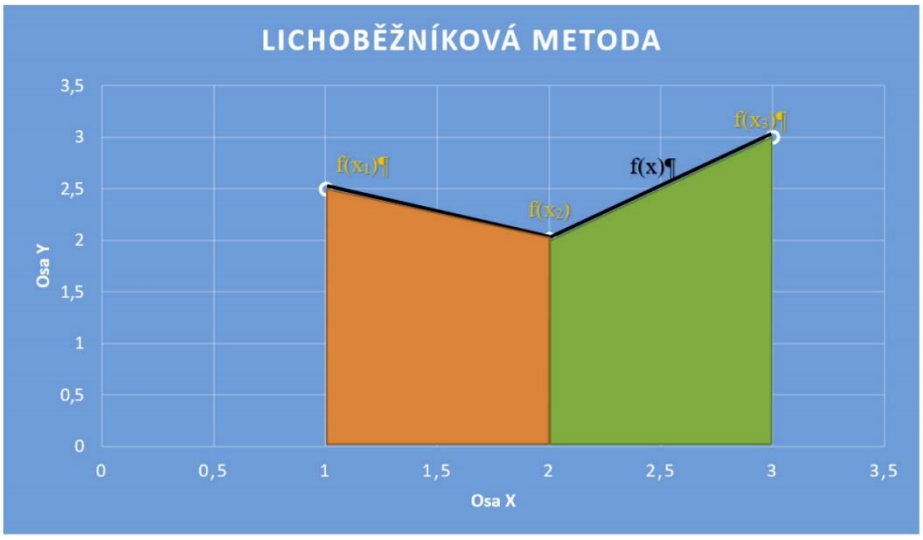
Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 17

6 LICHOBĚŽNÍKOVÁ METODA

Lichoběžníková metoda je jednou z metod pro výpočet integrálů. Tato metoda je poměrně univerzální, má ovšem jednu nevýhodu: lze ji použít pouze pro výpočet lineárních polynomů, například přímek.
Celá metoda spočívá v tom, že interval proložíme lichoběžníkem, u kterého následně vypočítáme jeho obsah. Velmi často se v praxi setkáváme s tím, že celý interval musíme rozdělit na více podintervalů. To zajišťuje větší přesnost výpočtu.

Graf metody:



Při lichoběžníkové metodě rozdělíme interval na více podintervalů a spočítáme obsah lichoběžníků, které nám vzniknou pod křivkou. Tyto lichoběžníky jsou zde barevně vyznačeny.

Čím více podintervalů vytvoříme, tím bude výsledek přesnější.

Výsledek získáme pomocí vzorce:

$$S = h(0{,}5 \times f(x_1) + f(x_2) + \cdots + f(x_{n-2}) + f(x_{n-1}) + f(x_n) \times 0{,}5)$$

1

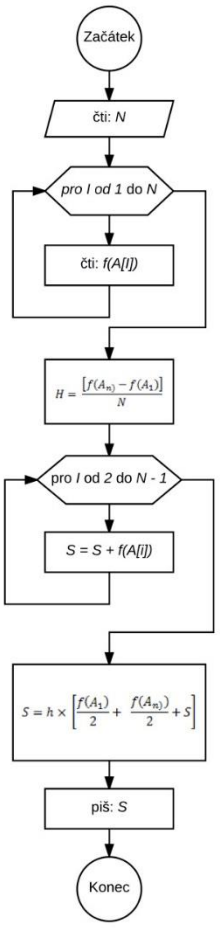
Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 18

6 LICHOBĚŽNÍKOVÁ METODA

Prohlédni si algoritmus a vývojový diagram lichoběžníkové metody:

1. Načteme počet bodů, které nám dělí interval na podintervaly. Čím více bodů zvolíme, tím bude výpočet přesnější. Tato proměnná je zde pojmenována *N*.
2. Načteme funkční hodnoty jednotlivých bodů.
3. Vypočítáme délku intervalu neboli vzdálenost od prvního bodu k bodu poslednímu. Tímto krokem vytvoříme proměnnou *H*.
4. Vytvoříme cyklus, který nám sečte celkovou funkční hodnotu všech bodů, vyjma těch krajních. Tuto proměnou pojmenujeme *F*.
5. Vypočítáme celkový obsah lichoběžníku, tento krok je založen na tzv. složené formuli.
6. Algoritmus vypíše obsah *S* a ukončí se.



Vypočítejte obsah pod křivkou pomocí Lichoběžníkové metody výpočtu integrálů. Křivka je zadána krajními body $f(A) = 2, f(B) = 4$.

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 19 – řešení pro učitele

ŘEŠENÍ

PRO UČITELE

NUMERICKÁ MATEMATIKA

- Numerická matematika > Numerické metody
- Numerická matematika vznikla kvůli vyčíslení $\sqrt{2}$
- Numerická matematika se využívá pro řešení matematických problémů
- Nejvíce chyb způsobuje člověk
- Stabilita algoritmu = určuje, zda je algoritmus vhodný pro daný problém či nikoli
- S numerickými metodami v běžné lidské činnosti se setkáváme např. v technice, přírodních vědách, ekonomice, financích ale i například v lékařských vědách..
- Konkrétní numerické metody: iterační, půlení intervalu, gaussova eliminační, lichoběžníková
- Pojmy v řadě: matematický problém → matematický model → numerická metoda → numerické řešení → výpočty

PROGRAMOVÁNÍ

- Programovací jazyky: c#, c++, delphi, pascal, java, visual basic atd.
- Ukázka strojového kódu: 00 90 BE 07 36 40, běžný programovací jazyk: int x;
- Vlastnosti algoritmů s vysvětlením:
    - Konečnost = Algoritmus musí vždy končit
    - Determinovanost = Algoritmus nám jasně říká, co máme provést v každém jednotlivém kroku.
    - Elementárnost = Algoritmus se skládá ze základních instrukcí
    - Obecnost = Algoritmus lze použít na problémy stejného typu (např. sčítání), ne pouze na jeden konkrétní problém.
    - Resultativnost = Algoritmus má vždy alespoň jeden výstup (výsledek).

Vytvořeno v rámci bakalářské práce, Vojtěch Gärtner, 2017.

Pracovní list č. 20 – řešení pro učitele

ŘEŠENÍ

PRO UČITELE

- Jednoduchý algoritmus pro otevření konzervy s jídlem:

  Začátek
  - Uchop konzervu do levé ruky
  - Uchop otvírák do pravé ruky
  - Upevni otvírák na horní okraj konzervy
  - Dokud horní část konzervy drží na místě, opakuj
    - Otoč kolečkem na otvíráku
  - Odříznutou část opatrně odlož
  - Vedle konzervy polož talíř
  - Obsah konzervy přemísti na talíř

  Konec

- Rozdělení příkazů do kategorií
  - Jednoduché: příkaz vstupu, příkaz výstupu, přiřazovací příkaz
  - Strukturované: příkaz větvení, příkaz cyklu, cyklus s podmínkou na konci, neúplné větvení, cyklus s konečným počtem opakování

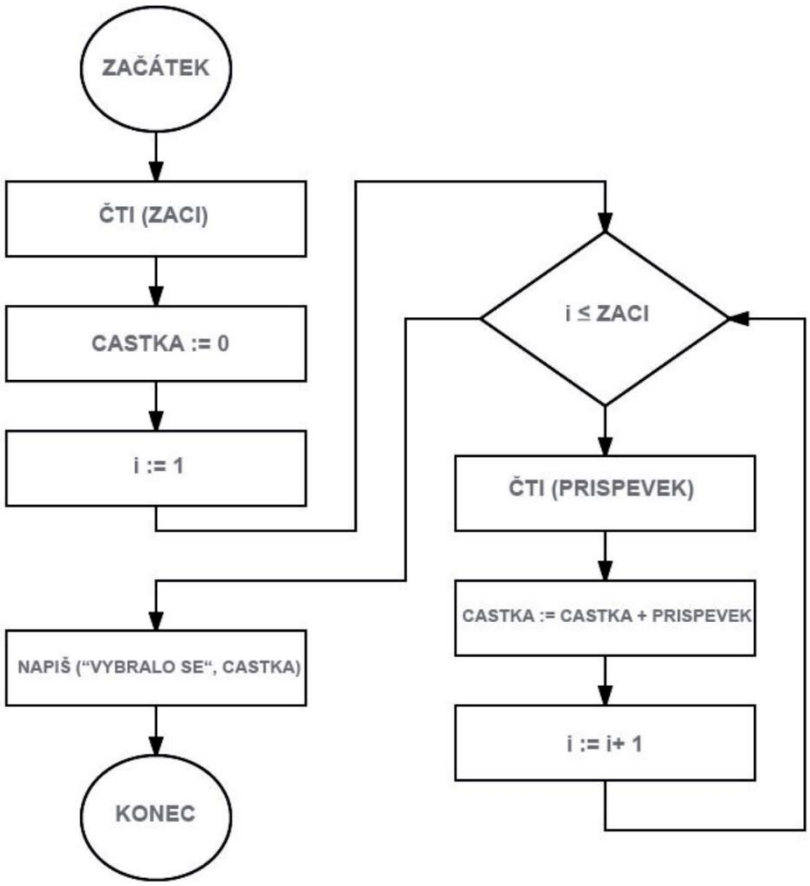
- Charakteristika
  - Příkaz vstupu: k zadání vstupních dat do programu z klávesnice
  - Příkaz větvení: skupina příkazů, které rozhodují a určují, jak se bude skript dál vyvíjet
  - Příkaz cyklu: když potřebujeme, aby se určitý příkaz nebo skupina příkazů vykonal vícekrát
  - Příkaz výstupu: k výstupu na monitor
  - Cyklus s podmínkou na konci: cyklus musí být proveden alespoň jednou
  - Neúplné větvení: jedna z větví je prázdná, neobsahuje žádné příkazy
  - Cyklus s konečným počtem opakování: cyklus, u kterého dopředu víme, kolikrát se zopakuje
  - Přiřazovací příkaz: proměnné v těle programu přiřadí určitou hodnotu

Pracovní list č. 21 – řešení pro učitele

ŘEŠENÍ

PRO UČITELE

- Doplnění vývojového diagramu:



Pracovní list č. 22 – řešení pro učitele

ŘEŠENÍ

PRO UČITELE

ITERAČNÍ METODA

```
Sub Iterace()
X = 1 * InputBox("X")
M = 1 * InputBox("Max")
E = 1 * InputBox("E")
I = 0

Xo = X
X = Cos(X)
Do While (Abs(Xo - X) > E) And (I <
M)
    I = I + 1
    Xo = X
    X = Cos(X)
Loop
MsgBox (X)
End Sub
```

METODA PŮLENÍ INTERVALU

```
Public Function F(X)
F = Cos(X) - X
End Function

Sub PuleniIntervalu()
A = 1 * InputBox("A")
B = 1 * InputBox("B")
E = 1 * InputBox("E")
I = 0
If F(A) * F(B) < 0 Then
    Do While (Abs(A - B) > E)
        X = (A + B) / 2
        If F(A) * F(X) < 0 Then
            B = X
        Else
            A = X
        End If
        I = I + 1
    Loop
    Else
    MsgBox ("Error")
End If
MsgBox ((A + B) / 2)
MsgBox (I)
End Sub
```

PRO UČITELE

GAUSSOVA ELIMINAČNÍ METODA

```
Sub Generuj()
Cells.ClearContents
H = InputBox("H")
For I = 1 To H
    For J = 1 To H
        Cells(I, J) = Int(100 * Rnd)
    Next J
    Next I
End Sub

Sub GEM()
H = InputBox("H")
P = 0
For I = 1 To (H - 1)
    For J = (I + 1) To H
        C = -Cells(J, I) / Cells(I, I)
        For K = 1 To H
            Cells(J, K) = Cells(J,
K) + C * Cells(I, K)
            P = P + 1
        Next K
    Next J
Next I

For I = H To 2 Step -1
    For J = (I - 1) To 1 Step -1
        C = -Cells(J, I) / Cells(I, I)
        For K = 1 To H
```

LICHOBĚŽNÍKOVÁ METODA

```
Sub LMVI()
A = 1 * InputBox("F(A)")
B = 1 * InputBox("F(B)")
N = 2
H = (B - A) / N
S0 = A / 2 + B / 2
For I = 1 To N
    S0 = S0 + (A + I * H)
Next I

S0 = H + S0
Do
    S1 = S0
    N = 2 * N
    H = (B - A) / N
    S0 = (A / 2 + B / 2)
    For I = 1 To N
        S0 = S0 + (A + I * H)
    Next I
    S0 = H * S0
Loop While (Abs(S0 - S1) < E)
MsgBox (S1)
End Sub
```

Desky k pracovním listům – vizualizace