UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Vybrané problémy výpočtů na PC s pohyblivou řádovou čárkou

Nomindalai Naranbaatar

Bakalářská práce
2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Nomindalai NARANBAATAR**
Osobní číslo:	**I07729**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vybrané problémy výpočtů na PC s pohyblivou řádovou čárkou**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem teoretické části práce bude studium problému výpočtů s pohyblivou řadovou čárkou. Cílem implementační části bude program na zpracování základních operací s aritmetickými výrazy (sčítání, odečítání, násobení, dělení), který bude dávat co nejpřesnější výsledky. Konkrétní zadání: Množina reálných čísel je v počítači reprezentována konečnou podmnožinou racionálních čísel – soustava čísel s pohyblivou řádovou čárkou (Floating Point Number System). Každé číslo lze psát ve tvaru: číslo $= \pm M \cdot z^E$, kde
M – mantisa čísla, zobrazená v soustavě o základu z
E – exponent
z – základ pro výpočet exponentové části
Tato reprezentace reálných čísel způsobuje problémy při vyhodnocování aritmetických výrazů. Neplatí totiž asociativní a distributivní zákony pro sčítání a násobení. Například při zobrazování čísel a zaokrouhlování mezivýsledků na tři platná místa platí: $a \cdot (b \cdot c) = 0,86 \cdot (0,56 \cdot 0,08) = 0,86 \cdot 0,0448 = 0,0385$ $(a \cdot b) \cdot c = (0,86 \cdot 0,56) \cdot 0,08 = 0,482 \cdot 0,08 = 0,0365$ Přesný výsledek je: 0,038528 Je vidět, že první výsledek je přesnější. Né vždy tomu ale tak musí být. Přesnost výsledku záleží na správném pořadí prováděných operací.

Rozsah grafických prací:
Rozsah pracovní zprávy:
Forma zpracování bakalářské práce: **tištěná/elektronická**
Seznam odborné literatury:

TOPFER, P.: Algoritmy a programovací techniky, Prometheus 1995

Vedoucí bakalářské práce: **RNDr. Josef Rak**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. ledna 2010**
Termín odevzdání bakalářské práce: **14. května 2010**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.

Byla jsem seznámena 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.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 1. 8. 2010 Nomindalai Naranbaatar

Poděkování

Na tomto místě bych ráda poděkovala svému vedoucímu panu RNDr. Josefu Rakovi za pomoc, který mi poskytoval v průběhu zpracování bakalářské práce. Dále musím poděkovat své rodině za podporu během celého studia.

Anotace

Tato práce se zabývá návrhem a implementací programu umožňující různé výpočty numerických úloh s pohyblivou řádovou čárkou a zároveň bude dávat co nejpřesnější výsledky. V teoretické části práce se nejprve zaměřuje na pohyblivou čárku a dále na popis problémů, které se nastávají při výpočtu na počítači s pohyblivou čárkou. Na závěr se zabývá zhodnocením vytvořené aplikace.

Klíčová slova

pohyblivá čárka, přesnost, správnost, binární, dekadické, IEEE 754

Title

Selected problems of computing on a PC with floating point

Annotation

This bachelor thesis deals with design and implementation of the program, which allows various calculations of numerical computation with floating point and will give most accurate results. Theoretical part of the thesis is initially focused on floating-point and a description of the problems that arise in calculation with floating-point on a computer. Conclusion deals with evaluation of the created application.

Keywords

floating point, precision, accuracy, binary, decimal, IEEE 754

Obsah

Seznam zkratek

IEEE	International Institute of Electrical and Electronics Engineers
FP	Floating Point
FX	Fixed Point
BCD	Binary Coded Decimal
NaN	Not a Number
UML	Unified Modeling Language
OOP	Object Oriented Programming
GUI	Graphical User Interface
JVM	Java Virtual Machine

Seznam obrázků

Seznam tabulek

1 Úvod

Jeden charakteristický rys, který rozlišuje počítačové aritmetiky od běžné aritmetiky je, že se používají nula a jedničky pro své výpočty. Dále v matematice se implicitně předpokládají, že pracujeme nad tělesem reálných čísel. Naproti tomu v počítači nemůžou být přesně reprezentována všechna reálná čísla a je pouhou aproximací tohoto tělesa. Počet čísel v počítači je přirozený a konečný. Z tohoto důvodu se při realizaci numerických algoritmů musíme soustředit na „správnost“ výsledku. Tento náročný problém vyvolá mnoho dalších problémů. Např. některé algoritmy jsou příliš pomalé, některé jsou nestabilní, anebo špatně podmíněné.

Cílem této práce je vytvořit knihovnu pracující s reálnými čísly a následně implementovat jednoduchou aplikaci, která využívá tuto knihovnu pro výpočet řešení numerických úloh. Abychom dostali co nejpřesnější výsledky, je třeba znát několik základních faktů o tom, jak počítač uloží čísla a jak s nimi provádí aritmetika.

Nejprve si vysvětlíme princip pohyblivé čárky, jak jsou čísla uložena a reprezentována. Dále se seznámíme s IEEE standardem a základními formáty, které jsou popsané v tomto standardu.

Druhá část práce je zaměřena na popis problémů, které jsou spojené s pohyblivou řádovou čárkou a jak se k některým problémům vyhnout.

Poslední část je věnována analýze zadání a popisu použitých technologií. Na závěru bude zhodnocení vytvořené aplikace.

2 Pohyblivá čárka

Pohyblivá čárka neboli plovoucí čárka v počítači popisuje systém pro numerickou reprezentaci, ve které řetězec číslic reprezentuje racionální číslo. Termín pohyblivá čárka je odvozena ze skutečnosti, že řádová čárka může pohybovat vzhledem k mantise. Reprezentace pohyblivé čárky je podobná k pojetí vědeckého zápisu. Řádová čárka není explicitně vyjádřena, ale implicitně se předpokládá, že leží v určité pozici vzhledem k mantise, obvykle před první nenulové číslici nebo za první nenulové číslici. V této práci dále budu používat tu první možnost, kdy řádová čárka je umístěna před první nenulové číslici.

Obecná forma pro zobrazení čísla X s pohyblivou čárkou:

$$\boldsymbol{X = \pm M \times Z^{E}} \tag{2.1}$$

kde

- Z - základ
- M - mantisa zobrazená v soustavě o základu Z
- E - exponent

2.1 Základ

Základ soustavy pro zobrazení exponentu i mantisy se většinou volí shodný se základem pro výpočet exponentové části. V některých případech však z důvodu zvětšení rozsahu zobrazitelného exponentu, zobrazují mantisu i exponent v jiné soustavě např. M a E v binární Z v šestnáctkové soustavě [30].

2.2 Mantisa

Mantisa uchovává ciferné hodnoty zobrazeného čísla. Počet číslic na mantise prezentuje přesnost čísla. Čím více bitů na mantise, tím menší chyby se vyskytují při zaokrouhlování. K dosažení co největší přesnosti zobrazení daného čísla se mantisa upravuje na tzv. *normalizovaný tvar*.

2.2.1 Normalizovaný a nenormalizovaný tvar

Normalizovaný tvar je tvar, kdy už nelze posunout mantisu doleva a tím se zjednodušuje i provádění aritmetické operace. Mantisa je v nenormalizovaném tvaru, když první číslice mantisy je nulová. Výsledek dvou normalizovaných čísel může být nenormalizovaný, proto se po každé operaci musí provést **normalizace** výsledku, tzn. upravit výsledek na normalizovaný tvar.

V binární pohyblivé řádce první bit je vždy nastaven na jedničku, a proto nemusí být ukládán do mantisy a tím ušetříme jeden bit.

2.3 Exponent

Exponent může být kladný i záporný a vyjadřuje, kde je umístěna řádová čárka. Podle IEEE standardu existuje hodnota bias, díky které je uložený exponent vždy kladný, viz kapitolu 3. Dále počet bitů použitých pro exponent určuje rozsah čísla.

2.4 Rozsah

Rozsah čísla s pohyblivou čárkou závisí na množství bitu nebo číslice použitých na exponentu a mantise. Množství normalizovaných čísel s pohyblivou řádovou čárkou možno určit následujícím způsobem (kde B - základ P – přesnost čísla, L – nejmenší exponent U – největší exponent) [1].

Tabulka 1 - Vztahy pro výpočet rozsahu

Význam	Vzorec
Počet normalizovaných čísel	$2 * (B - 1) * B^{(P-1)} * (U - L + 1)$
Nejnižší normalizované číslo	B^{L}
Největší normalizované číslo	$(B - B^{(-P)}) * B^{U}$
Nejnižší denormalizované číslo	B^{L-P}
Počet platných čísel	$\log_{10}(B^{P})$

Příklad: Zobrazit číslo 0,34 (na 4 bytech) ve tvaru s pohyblivou čárkou:

V binární pohyblivé čárce: $\mathbf{(0,34)_{10} = (0 \mid 0111\ 1101 \mid 010\ 1110\ 1100\ 1100\ 1100\ 1100)_{IEEE}}$

0,34×2 = 0,68 **0**
0,68×2 = 1,36 **1**
0,36×2 = 0,72 **0**
0,72×2 = 1,44 **1**
0,44×2 = 0,88 **0**
0,88×2 = 1,76 **1**
0,76×2 = 1,52 **1**
0,52×2 = 1,4 **1**
0,4×2 = 0,8 **0**
0,8×2 = 1,6 **1**
0,6×2 = 1,2 **1**
0,2×2 = 0,4 **0**

$(0,125)_{10} = (0,01010111\mathbf{0110}011001100110\ldots)_2$

normalizovaný tvar: $1,010111\mathbf{0110}0110011001100 \times 2^{-2}$
exponent: $2^{8-1}-1-2 = (125)_{10} = (\mathbf{1111101})_2$
znaménko: 0,34>0 ⟹ **0**

V desítkové pohyblivé čárce: $\mathbf{0,34 = (0 \mid 0 \mid 34)_{IEEE}}$

normalizovaný tvar: = ,34
exponent: $10^0 = (\mathbf{0})_{10}$
znaménko: 0,34>0 ⟹ **0**

2.5 Další reprezentace reálných čísel[1]

I Když je skoro ve všech procesorech podporován standard IEEE, používá se někdy i další způsoby pro reprezentace reálných čísel. Níže jsou uvedené některé další možnosti:

- **BCD** neboli dvojkově reprezentované dekadické číslo, kde každá cifra desítkového zápisu je reprezentována pomocí 4 bitů.
- Někdy se rozsah základních formátů prostě nestačí pro výpočet, v tom případě pohyblivá čárka může být implementována softwarově a přesnost čísla může být proměnlivá. Tomu se říká výpočet s libovolnou přesností (**arbitrary precision**).
- **Fixed point(FX)** neboli pevná řádová čárka, kde všechna čísla mají řádovou čárku umístěnou na stejném místě. Musí být tedy zachován stejný počet cifer v každém reprezentovaném čísle.

Výhodou reprezentace pohyblivé čárky od reprezentace pevné čárky nebo celých čísel je to, že umožňuje mnohem širší rozsah hodnot. Například v FX formátu číslo se sedmi desítkovými číslicemi s desetinnou čárkou umístěnou po páté číslici, můžeme představovat číslo 12345.67, 8765.43, 123.00, atd., zatímco číslo s pohyblivou řádovou čárkou se sedmi desetinnými číslicemi může reprezentovat číslo 1.234567, 123456.7, 0.00001234567, 1234567000000000, a tak dále.

[1] Zdroj tohoto pododdílu: [1]

3 IEEE standard

IEEE standard pro pohyblivou řádovou čárku je nejrozšířenějším standardem pro výpočet s pohyblivou řádovou čárkou a je používán na mnoha hardwarových i softwarových implementacích. Dříve existovaly dva IEEE standardy pro pohyblivou čárku, a to 754 pro binární a 854 pro desítkovou. Od roku 2008 jsou tyto standardy nahrazeny jedním standardem **IEEE 754-2008**. Podle tohoto standardu jsou povoleny jen báze 2 a 10. Standard definuje formáty pro reprezentaci čísel v pohyblivé čárce včetně záporných nul, denormalizovaných čísel a zvláštních hodnot (kladné a záporné nekonečno a NaN) [1]. Hlavním rozdílem formátu zobrazení IEEE standardu od obecné reprezentace je že exponent je vždy kladný a to zajišťuje hodnota bias, který se většinou volí dle vztahu: $bias = z^{e_n - 1} - 1$

$$\mathbf{X} = (-\mathbf{1})^{s} \times \mathbf{z}^{exp-bias} \times \boldsymbol{m} \qquad \textbf{(3. 1)}$$

Standard definuje:

- Aritmetické formáty – binární nebo desetinné číslo vyjadřující konečné reálné číslo a to včetně nul, nekonečna a dalších speciálních hodnot.
- Zaokrouhlovací metody – můžou být použity při zaokrouhlení čísel nebo při konverze.
- Aritmetické operace – základní operace, jako jsou sčítání, odčítání, násobení a dělení nebo rozšířené operace – mocnina, odmocnina atd.
- Zpracování výjimek – ošetřit speciální případy (např. dělení nulou, přetečení atd.)

3.1 Základní formáty

Podle bitové nebo číselné šířky exponentu, mantisy se rozlišují základní a rozšířené formáty FP čísel. V IEEE standardu jsou definovány 5 základních formátů. Tři z nich jsou pro binární a dva pro desítkové reprezentace pohyblivé čárky.

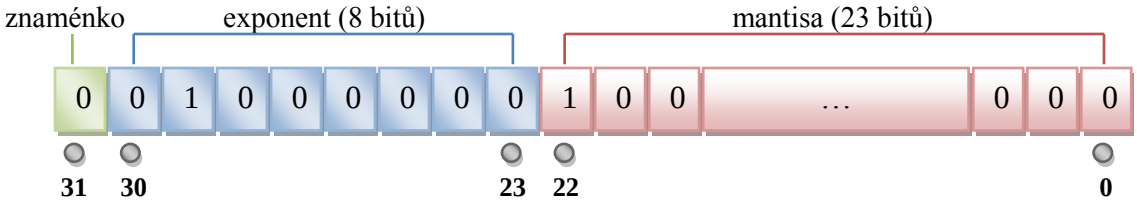
Tabulka 2 - Základní formáty IEEE 754, zdroj: [1]

Název	Obecný název	Báze	Počet číslic (bitů)	Min exponent	Max exponent
Binary32	Single	2	23+1	-126	+127
Binary64	Double	2	52+1	-1022	+1023
Binary128	Quadruple	2	112+1	-16382	+16383
Decimal64		10	16	-383	+384
Decimal128		10	34	-6143	+6144

U binárních formátů je k počtu číslici přičtena jedna, protože normalizovaný tvar mantisy obsahuje navíc skrytou jedničku. Všechny tyto základní formáty jsou implementovány jak na hardware, tak i na software. V dnešní době se v programovacích jazycích nejvíce používají první dva binární formáty, a to **binary32** a **binary64** [5]. Z tohoto důvodu, níže uvedu specifikace jen těchto dvou formátů.

1. Jednoduchá přesnost – 32 bitů

Jednoduchá přesnost (*single precision*) neboli binary32 je formát pro binární pohyblivou čárku, který zabírá 32 bitů paměť pro svou reprezentaci. Jednotlivé bity jsou rozděleny následovně:



Obrázek 1 - Jednoduchá přesnost, zdroj: vlastní

Rozsah exponentu a hodnot:

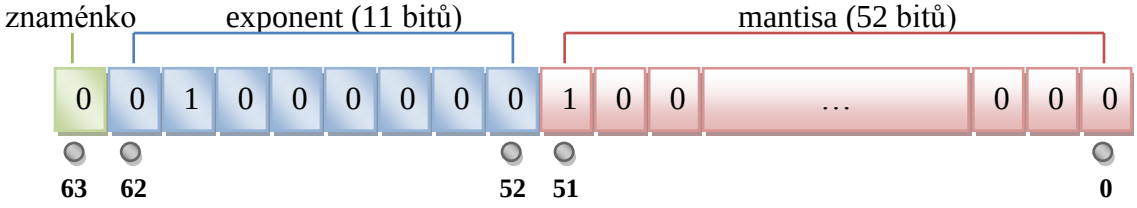
- E_{min} = -126, E_{max} = 127, bias – 127
- počet platných cifer: 7 až 8
- rozsah hodnot: $-3{,}40282\times10^{38}$ až $3{,}40282\times10^{38}$
- nejnižší kladná normalizovaná hodnota: $1{,}17549\times10^{-38}$
- nejnižší kladná nenormalizovaná hodnota: $1{,}40129\times10^{-45}$

Tabulka 3 - Datové typy s jednoduchou přesností

Jazyk	Název datového typu
Fortran	real*4 (real)
Pascal	single
Java, C, C++	float

2. Dvojitá přesnost – 64 bitů

Dvojitá přesnost (*double precision*) neboli binary64 je formát pro binární pohyblivou čárku, který zabírá 64 bitů paměť pro svou reprezentaci. Jednotlivé bity jsou pak rozděleny takhle:



Obrázek 2 - Dvojitá přesnost, zdroj: vlastní

Rozsah exponentu a hodnot:

- E_{min} = -1022, E_{max} =1023, bias = 2047
- Počet platných cifer: 15 až 16
- rozsah hodnot: $-1{,}79769\times10^{308}$ až $1{,}79769\times10^{308}$
- nejnižší normalizovaná hodnota: $2{,}2\times10^{-308}$
- nejnižší nenormalizovaná hodnota: 4.9406×10^{-324}

Tabulka 4 - Datové typy s dvojitou přesností

Jazyk	Název datového typu
Fortran[2]	real*8 (double)
Pascal	double
Java, C, C++	double

3.2 Další formáty

Kromě těchto základních formátů existují i další implementace pohyblivé čárky v programovacích jazycích.

3.2.1 Datové typy Pascalu

Firma Borland, který je známý svým jazykem Turbo Pascal vytvořila vlastní implementaci pohyblivé čárky a nazval ji „real“. Tento datový typ pro svoji reprezentaci požaduje šest bytů. Veškeré výpočty s tímto typem se provádějí na CPU, a pro přenos výsledků z funkce se používá trojice registrů. Další datové typy reálných čísel:

Tabulka 5 - Další datové typy Pascalu

Typ	Velikost	Rozsah hodnot			Přesnost
Real	6 bytů	2,9. 10–39	-	1,7. 1038	11 až 12 cifer
Extended	10 bytů	1,9. 10–4932	-	1,1. 104932	19 až 20 cifer
Comp	8 bytů	-2^{63}	-	$2^{63}-1$	19 až 20 cifer

3.2.2 Knihovna Apfloat [3]

Apfloat je knihovna pro jazyk Java a umožňuje výpočty s proměnlivou přesností až miliony. To je podobná k BigDecimal v Javě, ale umožňuje mnohem vyšší přesnost. Také obsahuje kompletní sadu matematických funkcí (všechny metody v `java.lang.Math` a více). Pro svou vnitřní strukturu používá základní formáty IEEE standardu. Existuje i verze pro C++.

[2] Fortran (FORmula TRANslator) je programovací jazyk, který v 50. letech 20. století navrhla firma IBM pro vědecké výpočty a numerické aplikace. Aktuální verze: Fortran 2008.
[3] Více informace na http://www.apfloat.org/

3.3 Speciální hodnoty

Denormalizovaná čísla – jsou taková čísla, která mají minimální hodnotu exponentu a nenulovou mantisu.

NaN – vznikne v případě, že je použita operace s nejasným výsledkem, například 0/0, 0^0 nebo při odmocňování záporných čísel

Nekonečno – vzniká typicky při dělení nulou (zde je možné zjistit znaménko), nebo při vyjádření funkcí typu $\log(0)$ atd.

Tabulka 6 - Speciální hodnoty, zdroj: [14]

Znaménko	Exponent	Mantisa	Význam
0	E_{min} <e<E_{max}	>0	normalizované kladné číslo
1	E_{min} <e<E_{max}	>0	normalizované záporné číslo
0	E_{min}	>0	denormalizované kladné číslo
1	E_{min}	>0	denormalizované záporné číslo
0	E_{min}	0	kladná nula
1	E_{min}	0	záporná nula
0	E_{max}	0	kladné nekonečno
1	E_{max}	0	záporné nekonečno
0	E_{max}	>0	NaN – not a number
1	E_{max}	>0	NaN – not a number

3.4 Zaokrouhlování

Každé číslo reprezentované pomocí pohyblivé čárky je racionální číslo s konečným počtem číslic. Iracionální číslo jako jsou π, $\sqrt{2}$ nebo racionální číslo s nekonečným počtem číslic musí být zaokrouhlované na konečný počet číslic. Počet platných číslic omezuje také správnost daného čísla, např. číslo 12345 nemůže být přesně reprezentované s 4 platnými číslicemi. Zda číslo je konečné nebo nekonečné závisí na základě. Například, v reprezentaci se základem 10 číslo ½ (0.5) je konečné zatímco číslo 1/3 (0.33333333…) konečné není [1].

Typy zaokrouhlování[4]

V normě je specifikovaných několik způsobů zaokrouhlení (tzv. **rounding modes**). Od zavedení standardu IEEE 754 se dnes nejvíce používá metoda zaokrouhlení k nejbližšímu. Typy zaokrouhlování:

- zaokrouhlení k nejbližšímu (*round*) – výsledkem je celé číslo, které je na číselné ose nejblíže zaokrouhlovanému číslu (např. 2.4 na 2, 2.7 na 3),

[4] Zdroj: [13]

- zaokrouhlení nahoru (*ceil*) – výsledkem je nejbližší celé číslo, které je větší nebo rovno zaokrouhlovanému číslu (např. -2.4 na -2, 2.4 na 3),
- zaokrouhlení dolů (*floor*) – výsledkem je nejbližší celé číslo, které je menší nebo rovno zaokrouhlovanému číslu (např. -2.4 je zaokrouhleno na -3, 2.4 na 2),
- zaokrouhlení s preferencí sudé číslice.

Při jakémkoli zaokrouhlování dochází k nutné a žádoucí chybě nepřesnosti. Kromě toho ovšem dochází k nežádoucím chybám, jako jsou zaokrouhlení pětky a postupné zaokrouhlování.

Při zaokrouhlení pětky se volí mezi 2 možnosti, a to nahoru nebo s preferencí sudé. IEEE 754 standard používá zaokrouhlování s preferencí sudé pro zobrazení čísla s pohyblivou čárkou.

Příklad zaokrouhlení pětky: Zaokrouhlit číslo 7,685 na 2 desetinná místa

$$7{,}685 \doteq \begin{cases} 7{,}69 \text{ (zaokrouhlení nahoru)} \\ 7{,}68 \text{ (zaokrouhlení s preferencí sudé)} \end{cases}$$

Při postupném zaokrouhlování (zaokrouhlení na nižší a pak až na vyšší řád) vzniká další typ chyby. Výsledek se může lišit od výsledku při přímém zaokrouhlení na vyšší řád.

Příklad postupného zaokrouhlování: Zaokrouhlit číslo 2,4445 na 2 desetinná místa

$$2{,}4445 \doteq 2{,}445 \doteq 2{,}45 \doteq 2{,}5 \doteq 3 \text{ (postupné zaokrouhlování)}$$
$$2{,}4445 \doteq 2 \text{ (přímé zaokrouhlování)}$$

4 Aritmetika s pohyblivou řádovou čárkou

Pro všechny aritmetické operace se znaménko, mantisa a exponent musejí zpracovat zvlášť. Algoritmy pro aritmetické operace jsou závislé na systému reprezentace čísel. Při každé operace je výhodné pracovat s normalizovanými čísly, protože výsledek aritmetické operace je opět normalizované číslo. Po každé operace se provede normalizace a testuje, jestli došlo k přetečení nebo podtečení.

4.1 Speciální případy

Při provádění aritmetické operace může dojít k několika speciálním případům.

- Přetečení *(overflow*) – je jev, který nastane, pokud exponent výsledku dosáhne maximální hodnotu. Výsledek se nastaví na kladné/záporné nekonečno podle znaménka původního čísla.
- Podtečení (*underflow*) – je jev, který nastane, pokud exponent výsledku dosáhne minimální hodnotu. Výsledek se nastaví na kladnou/zápornou nulu podle znaménka původního čísla.
- Dělení nulou (*divide by zero*) – se nastane při operaci dělení, při němž je dělitel rovný nule. Výsledek tohoto případu je uvedený v tabulce 8.

Tabulka 7 - Složitost algoritmů aritmetických operací, zdroj: [29]

Operace	Vstup	Výstup	Algoritmus	Složitost
Sčítání	2 *n*-ciferná čísla	1 *n*+1ciferné číslo	Školní sčítání s přenosem	O (*n*)
Odečítání	2 *n*-ciferná čísla	1 *n*+1ciferné číslo	Školní odečítání	O (*n*)
Násobení	2 *n*-ciferná čísla	2 *n*-ciferná čísla	Školní násobení	O (n^2)
Dělení	2 *n*-ciferná čísla	1 *n*+1ciferné číslo	Školní dlouhé dělení	O (n^2)

4.2 Sčítání a odčítání

Sčítání je ta nejzákladnější operace, kterou používá ostatní operace pro svůj výpočet. Nejprve se musejí převést obě čísla na stejný exponent. Odečítání můžeme dostat přičtením číslem s opačným znaménkem.

$$\text{MATEMATICKY: } \boldsymbol{x_1 \times x_2 = (-1)^{\max(x_1,x_2)} \times z^{\max(e_1,e_2)} \times (m_1 + m_2)} \quad \textbf{(4.1)}$$

Algoritmus sčítání a odečítání[5]

- Zjistit, zda se sčítání či odečítání neprovádí **se speciálními hodnotami**. Výsledek provedeného součtu se speciálními hodnotami je uvedený v tabulce 4.

[5] Zdroj: [14]

- Dále se porovnávají exponenty. Výsledný **exponent** bude větší z obou operandů. Vypočítá se číselný rozdíl exponentů, tj. $e_{dif}=e_1-e_2$. Pokud je $e_{dif} \neq 0$, potom mantisu menšího čísla nutné posunout doprava o tolik bitů, kolik činí rozdíl exponentů.
- Vyjádřit výsledné **znaménko** podle velkého čísla.
- **Přesnost** výsledků je větší z nich.
- Podle znaménka obou operandů je vyčíslen buď součet, nebo rozdíl jejich mantis.
- Pokud hodnota výsledku přeteče, posunou se bity v mantise doprava o jeden bit a exponent se zvýší o jeden.
- Provádí se **normalizace**, podle které se mantisa posune doleva o tolik, kolik je nula na její začátku a exponent se přitom snižuje.
- Po normalizace se provede **zaokrouhlení** hodnoty na požadované přesnosti.

Tabulka 8 - Speciální případy při operaci sčítání, zdroj: [26]

X	Y	X+Y
+nekonečno	+nekonečno	+nekonečno
-nekonečno	-nekonečno	-nekonečno
+nekonečno	-nekonečno	NaN
NaN	libovolné	NaN

Příklad: Vypočítat součet a rozdíl čísel: 123,45 a 0,12234 s přesností na 5 platných cifer

Nejprve převedeme čísla na FP formátu:

$e_1 = 3$; $m = 0{,}12345$; $p = 5$
$e_2 = 0$; $m_2 = 0{,}12234$; $p_2 = 5$

Pak převedeme čísla na stejný exponent:

$e = 3$; $m = 0{,}12345$; $p = 5$
$e_2 = 3$; $m_2 = 0{,}00012234$; $p_2 = 5$

Sčítáme dvě čísla:

$e_1 = 3$; $m_1 = 0{,}12345$; $p = 5$
\+ $e_2 = 3$; $m_2 = 0{,}00012234$; $p_2 = 5$

$e = 3$; $m = 0{,}12357234$; $p = 8$

Odečítáme dvě čísla:

$e = 3$; $m_1 = 0{,}12345$; $p_1 = 5$
− $e_2 = 3$; $m_2 = 0{,}00012234$; $p_2 = 5$

$e = 3$; $m = 0{,}12332765$; $p = 8$

Normalizace a zaokrouhlení výsledku:

$e = 3$; $m = 0{,}12357$**(234)**; $p = 5$ — $e = 3$; $m = 0{,}12332$**(765)**; $p = 5$

Výsledek: **12357** — **12332**

4.3 Násobení

Operace násobení se provádí různými způsoby, jednou z možností je pomocí několikanásobného sčítání, pro dvě větší čísla tenhle způsob je moc pomalý. Ještě lepším algoritmem je „školní“ algoritmus. Pomocí školního algoritmu součin dostaneme jako součet částečných součinů, pro každou číslici násobitele při sčítání je každý částečný součet posunut o jeden bit doleva vzhledem k předchozímu součinu.

$$\text{MATEMATICKY: } \boldsymbol{x_1 \times x_2 = (-1)^{s1 \oplus s2} \times z^{e_1+e_2} \times (m_1 \times m_2)} \quad \textbf{(4.2)}$$

Algoritmus násobení

- Zjistit, zda se součin neprovádí se speciálními hodnotami. Výsledek provedeného součinu se speciálními hodnotami je uvedený v tabulce 5.
- **Exponent** výsledku je součet exponentů operandů.
- **Přesnost** výsledků je větší z nich.
- Výsledné **znaménko** je XOR znaménka obou operandů.
- Zjistí se větší **mantisa**, aby minimalizoval počet částečných součtů. Poté je vypočítána výsledná mantisa podle školního algoritmu.
- Provede se **normalizace** výsledku.
- Po normalizaci **se zaokrouhlí** výsledek na požadované přesnosti.

Tabulka 9 - Speciální případy při operaci násobení, zdroj: [26]

X	Y	X×Y
nenulové kladné	+nekonečno	+nekonečno
nenulové záporné	+nekonečno	-nekonečno
nenulové kladné	-nekonečno	-nekonečno
nenulové záporné	-nekonečno	-nekonečno
+/- nekonečno	+/- 0	NaN
+nekonečno	+nekonečno	+nekonečno
-nekonečno	-nekonečno	+nekonečno
+nekonečno	-nekonečno	-nekonečno
libovolné	NaN	NaN

Příklad: Vypočítat součin dvou čísel: 123,45 a 0,12234 s přesností na 5 platných cifer

Nejprve převedeme číslo na FP formátu
$e_1 = 3;\ m_1 = 0{,}12345;\ p_1 = 5$ $e_2 = 0;\ m_2 = 0{,}12234;\ p_2 = 5$

Násobení:
$(0{,}12224 \times 0{,}12345) \times 10^{3+0} = 0{,}015102873 \times 10^{3}$

Normalizace:
$0{,}015102873 \times 10^{3} = 0{,}15102873 \times 10^{2}$

Zaokrouhlení:
$0{,}15102873 \times 10^{2} \doteq 0{,}15102 \times 10^{2}$

Výsledek: <u>15,102</u>

4.4 Dělení

Je mnohem komplikovanější operace než předešlé tři operace, protože předem neznáme přesnost výsledného čísla. Opět tu existují několik možností realizace, např. postupné odečítání dělitele od dělence, výpočet převrácené hodnoty dělitele a následně vynásobit dělence převrácenou hodnotou dělitele.

$$\text{MATEMATICKY: } \boldsymbol{x_1 \times x_2 = (-1)^{s1 \oplus s2} \times z^{e_1 - e_2} \times \left(\frac{m_1}{m_2}\right)} \qquad \textbf{(4.3)}$$

Algoritmus školního dělení[6]

- Zjistit, zda se dělení neprovádí se speciálními hodnotami nebo dělitel se nerovná nule. Výsledek provedeného dělení se speciálními hodnotami nebo dělitelem nula je uvedený v tabulce 6.
- V ostatním případě se výsledek počítá podle školního algoritmu.
  1. Vytvoříme novou proměnnou **adjust** a inicializujeme na nulu.
  2. Mantisy operandů jsou upravené tak, aby mantisa dělence byla větší nebo se rovná děliteli a mantisa dělitele by byla maximálně deset krát méně než dělitele.
     - Dokud mantisa dělitele je větší než dělence, mantisa dělence je násobena deseti a proměnná adjust se inkrementuje.
     - Dokud desetinásobek mantisy dělitele je menší než dělence, mantisa dělitele je násobena deseti a proměnná adjust se dekrementuje.
  3. Výsledek je inicializován na nulu.
  4. Následující kroky jsou opakované, dokud nezískáme výsledek.
     - Dokud mantisa dělence je větší než dělitele, dělitel je odečítán od dělence a výsledek se inkrementuje.
     - Je-li nyní mantisa dělence je nula, nebo počet číslic ve výsledné mantise je větší než požadované přesnosti, následuje krokem 5.
     - Jinak je dělenec násoben deseti a adjust se inkrementuje
  5. Adjust se rovná rozdílu počtu číslic ve výsledné mantise a původního adjust
  6. **Exponent** = (exponent dělence – počet číslic v mantise dělence) – (exponent dělitele – počet číslic v mantise dělitele) + adjust.

[6] Zdroj: [7]

7. Výsledné **znaménko** je XOR znaménka obou operandů.
8. Provede se **normalizace** výsledku.
9. Po normalizaci **se zaokrouhlí** výsledek na požadované přesnosti.

Tabulka 10 - Speciální případy při operaci dělení, zdroj: [26]

X	Y	X/Y
nenulové číslo	+0	+nekonečno
nenulové číslo	-0	-nekonečno
nenulové číslo	+/-nekonečno	0
+/-nekonečno	+-nekonečno	NaN
0	0	NaN

Příklad: Vypočítat podíl dvou čísel: 13 a 5 s přesností na 5 platných cifer

Dělení:

1. adjust = 0; result = 0; $m_1 = 13$; $m_2 = 5$

2. $m_1 = 8$ result = 1 // Dokud ($m_1 > m_2$), $m_1 = m_1 - m_2$ a result++
$m_1 = 3$ result = 2

3. $m_1 \neq 0 \cup presnost_{result} < 5$ // Výsledek nedosáhl požadovanou přesnost ⇒ pokračujeme

4. $m_1 = 30$; adjust = 1; result = 2**0**

5. $m_1 = 25$ result = 21 // Dokud ($m_1 > m_2$), $m_1 = m_1 - m_2$ a result++
$m_1 = 20$ result = 22
$m_1 = 15$ result = 23
$m_1 = 10$ result = 24
$m_1 = 5$ result = 25
$m_1 = 0$ result = 26

3. $m_1 == 0$ // Výpočet končí

Exponent:

$exp_{result} = (2 - 2) - (1 - 1) + 1 = 1$

Normalizace a zaokrouhlení:

výsledek se nezmění

Výsledek: 2,6

5 Problémy výpočtů s pohyblivou řádovou čárkou

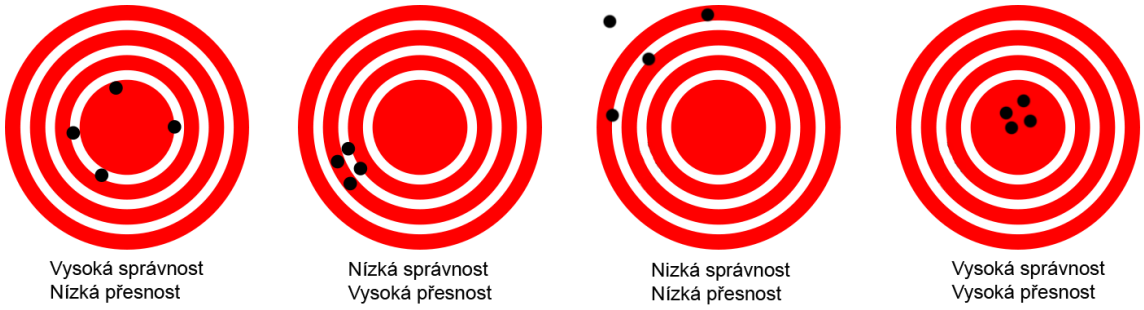
Skutečnost, že číslo s pohyblivou řádovou čárkou nemůže přesně nahradit reálné číslo a výpočty s těmito čísly nemohou dávat výsledky jako aritmetické operace, vede k mnoha překvapujícím situacím. Toto je spojen s konečným počtem číslic. Například, číslo $(0.1)_{10}$ nemůže být přesně reprezentované v binární soustavě. V počítačové aritmetice neplatí ze základních aritmetických zákonů asociativní a distributivní zákony. Implementace výpočtu, která nebere tyto skutečnosti na ohled, může dávat nepřesné nebo i zcela chybné výsledky (přestože počítačová aritmetika funguje tak, jak má). Při numerických úkolech např. vypočítání inverzní matice, vlastní číslo nebo při řešení diferenciálních rovnic, kde je třeba mnohokrát provádět aritmetické operace, malé chyby můžou akumulovat.

5.1 Správnost a přesnost[7]

Zatímco se v aritmetice přesnost vyjadřuje jako rozptyl kolem střední hodnoty, v počítači může mít několik významů.

- Přesnost určuje počet číslic v mantise (kromě vedoucích nul). Číslo nula je speciálním případem, jeho přesnost závisí na konkrétní implementaci, ale většinou má přesnost 1. Např. následujících pět čísel má přesnost 3.
  `123 1.23 1.50 0.00123 1.23E+22`.
- V některých programovacích jazycích nebo v pevné řádové čárce udává počet číslic za desetinnou čárkou.

Správnost udává průměrnou vzdálenost výsledků měření od skutečné hodnoty. I když čísla v počítači může mít vysokou přesnost, správnost těchto čísel nemusí být vysoká.



Obrázek 3 - Správnost proti přesnosti, zdroj: vlastní

Například součet dvou čísel **109.856, 65.89** v dvojité přesnosti je **175.74**599999999998. I když výsledek má přesnost 17, správnost výsledku je jen na 5 číslicích. Přesný výsledek je **175,746**. Rozdílu mezi skutečnou hodnotou a výsledkem získaný v počítači se říká zaokrouhlovací chyba (**round-off**) a je omezena přesnosti stroje, viz 5. 1. 1.

[7] Zdroje tohoto oddílu: [3]

Kromě výše uvedených problémů mohou nastat jiné problémy při výpočtu s pohyblivou řádovou čárkou.

- Krácení platných cifer (*cancellation*) se nastává v případě, kdy odečítáme dvě téměř shodná čísla.
- Omezený rozsah exponentu může způsobit přetečení nebo podtečení a dávat zcela jiné výsledky. V tom případě přesnost čísla bude ztracena.
- Testování bezpečného dělení: kontrolovat jen dělitele, zda není nula, negarantuje, že výsledek nebude přetékat a dávat nekonečno.
- Testování rovnosti: dva výrazy, které se matematicky rovnají, nemusejí se rovnat v aritmetice s pohyblivou řádovou čárkou.

Použití testu rovnosti jako `if(x==y)` je obvykle nedoporučený, když očekávání je založené na výsledku od čisté matematiky. Takový test by měl být nahrazen `if(abs(x-y) < epsilon)`, kde epsilon je dostatečně malé číslo přizpůsobené k aplikaci a říká se mu přesnost stroje, viz níže.

5.2 Přesnost stroje

Přesnost stroje je největší relativní chyba vyskytující při zaokrouhlování a může ji označit symbolem ε nebo **u**. Častým zdrojem zaokrouhlovacích chyb je přesnost. Někdy přesnost stroje znamená nejmenší číslo v daném systému, které splňuje podmínku `((1.0 + u) ≠ 1.0)`. Výpočet přesnosti stroje závisí na systému reprezentace a režimu zaokrouhlování. Například při zaokrouhlování směrem k nule je vypočítán $\mathbf{u = báze^{(1 - přesnost)}}$, při zaokrouhlování k nejbližšímu: $\mathbf{u = ½(báze^{(1 - přesnost)})}$.

5.3 Absolutní a relativní chyba

Nechť hodnota $\bar{x}$ je aproximace přesné hodnoty x, pak je absolutní chyba definována jako:

$$\boldsymbol{\varepsilon = x - \bar{x}} \tag{5.1}$$

a relativní chyba:

$$\boldsymbol{\varepsilon_r = \frac{x - \bar{x}}{\bar{x}}} \tag{5.2}$$

6 Podmíněnost matic

V praxi je však chybami zatížen nejen samotný výpočet, ale i hodnoty do něj vstupující. Pomocí čísla podmíněnosti, můžeme odhadnout, jak moc se nám projeví chyby zadání ve výpočtu konečného řešení. Vzhledem k chybám vstupních údajů a k chybám zaokrouhlovacím, řešíme na místo $A \times x = b$, úlohu $(A + \Delta A)(x + \Delta x) = b + \Delta b$.

Pro relativní chybu řešení platí:

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A\| \times \left\|A^{-1}\right\| \times \frac{\|\Delta b\|}{\|b\|} \tag{6.1}$$

Nechť A je čtvercová regulární matice. Číslem podmíněnosti matice A rozumíme číslo

$$\kappa(A) = \|A\| \times \left\|A^{-1}\right\| \tag{6.2}$$

Soustava se nazývá dobře podmíněná, jestliže κ(A) ≈ 1, v opačném případě se nazývá špatně podmíněná a malé chyby vstupních dat i malé zaokrouhlovací chyby ve výpočtu se projeví velkou chybou řešení.

Příklad: *Zjistit, jestli daná lineární rovnice je dobře podmíněná.*

$$x + y = 2$$

$$x + 1.0001y = 2.0001$$

Řešení: *Tato soustava má řešení x=1 a y=1. Uděláme malou změnu v pravé straně rovnice*

$$x + y = 2$$

$$x + 1.0001y = 2.0002$$

Tato soustava má však již řešení $x = 0$, $y = 2$.

Výpočet čísla podmíněnosti: Nejprve soustavu lineární rovnice převedeme na maticový tvar.

$$\begin{pmatrix} 1 & 1 \\ 1 & 1.0001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2.0001 \end{pmatrix}$$

Pro výpočet čísla podmíněnosti musíme počítat inverzní matici matice A.

$$A^{-1} = \begin{pmatrix} 10001 & -10000 \\ -10000 & -10000 \end{pmatrix}$$

$$\|A\| = 2.0001, \|A^{-1}\| = 20001 \implies \kappa(A) = 2.0001 \times 20001 = 40004.0001$$

Vidíme tedy, že při velmi malé změně pravé strany došlo k velmi podstatné změně výsledku.

7 Soustava rovnic

V matematice a lineární algebře se jako soustava lineárních rovnic označuje množina lineárních rovnic. Obecně může být soustava $\boldsymbol{m}$ lineárních rovnic s $\boldsymbol{n}$ proměnnými zapsána takhle:

$$\begin{pmatrix} \boldsymbol{a_{11}x_1} & + & \boldsymbol{a_{12}x_2} & + & \cdots & \boldsymbol{a_{1n}x_n} & = & \boldsymbol{b_1} \\ \boldsymbol{a_{21}} & + & \boldsymbol{a_{22}x_2} & + & \cdots & \boldsymbol{a_{2n}x_n} & = & \boldsymbol{b_2} \\ \cdots & & \cdots & & \cdots & & & \\ \boldsymbol{a_{m1}} & + & \boldsymbol{a_{m2}x_m} & + & \cdots & \boldsymbol{a_{mn}} & = & \boldsymbol{b_m} \end{pmatrix} \tag{7.1}$$

kde proměnné $x_1, \cdots, x_n$ jsou neznámé a_{ij} jsou *koeficienty* soustavy rovnic. Čísla b_i, kde $i = 1,2, \cdots, m$ jsou *absolutní členy soustavy* (nebo také tzv. pravá strana soustavy). Koeficienty lze zapsat ve tvaru matice:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a_{11}} & \boldsymbol{a_{12}} & \cdots & \boldsymbol{a_{1n}} \\ \boldsymbol{a_{21}} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \boldsymbol{a_{(m-1)n}} \\ \boldsymbol{a_{m1}} & \cdots & \boldsymbol{a_{m(n-1)}} & \boldsymbol{a_{mn}} \end{pmatrix} \tag{7.2}$$

Tuto matici označujeme jako ***matice soustavy***. Neznámé a pravou stranu soustavy je možné vyjádřit jako vektory:

$$\vec{\boldsymbol{x}} = \begin{pmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \vdots \\ \boldsymbol{x_n} \end{pmatrix}, \quad \vec{\boldsymbol{b}} = \begin{pmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \vdots \\ \boldsymbol{b_m} \end{pmatrix} \tag{7.3}$$

Celou soustavu rovnic v maticovém zápisu je tedy možné vyjádřit takto:

$$\begin{pmatrix} \boldsymbol{a_{11}} & \boldsymbol{a_{12}} & \cdots & \boldsymbol{a_{1n}} \\ \boldsymbol{a_{21}} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \boldsymbol{a_{(m-1)n}} \\ \boldsymbol{a_{m1}} & \cdots & \boldsymbol{a_{m(n-1)}} & \boldsymbol{a_{mn}} \end{pmatrix} \begin{pmatrix} \boldsymbol{x_1} \\ \boldsymbol{x_2} \\ \vdots \\ \boldsymbol{x_n} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \vdots \\ \boldsymbol{b_m} \end{pmatrix} \tag{7.4}$$

nebo zkráceně:

$$\boldsymbol{A} \times \vec{\boldsymbol{x}} = \vec{\boldsymbol{b}} \tag{7.5}$$

7.1 Matice

Matice je v matematice obdélníková tabulka čísel nebo nějakých matematických objektů. Obsahuje obecně $\boldsymbol{m}$ řádků a $\boldsymbol{n}$ sloupců. Hovoříme pak o matici typu $\boldsymbol{m} \times \boldsymbol{n}$ a může být vyjádřena takhle:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a_{11}} & \boldsymbol{a_{12}} & \cdots & \boldsymbol{a_{1n}} \\ \boldsymbol{a_{21}} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \boldsymbol{a_{(m-1)n}} \\ \boldsymbol{a_{m1}} & \cdots & \boldsymbol{a_{m(n-1)}} & \boldsymbol{a_{mn}} \end{pmatrix} \tag{7.6}$$

Matice se často využívají pro vyjádření obecné rotace vektorů, transformace vektorů od jedné báze k bázi jiné, k výpočtu soustav lineárních rovnic.

7.2 Metody pro výpočet řešení

Úkolem při řešení soustavy rovnice je najít takové hodnoty $x_1, \cdots, x_n$ pro které platí všechny rovnice zároveň. Pro řešení soustavy rovnice existují různé metody např. metoda nejmenších čtverců, iterační metody jako jsou Gauss-Seidel nebo Jacobiho metoda atd. Níže věnuji popisu některých metod, které jsem použila ve své práci.

- Gaussova[8] eliminační metoda (bez pivotace) představuje řešení převedením rozšířené matice na horní trojúhelníkový tvar a pak pomocí substitucí vypočítat všechny řešení soustavy.
- Gaussova eliminace s pivotací: rozdíl od předešlé metody je v tom, že při úpravě na trojúhelníkový tvar se jako pivot bere prvek s maximální absolutní hodnotou v daném sloupci.
- Gauss-Jordanova metoda je rozšířením Gaussovy eliminační metody, princip spočívá ve převodu rozšířené matice na diagonální tvar. Po této úpravě nemusíme nic dosazovat, všechny výsledky se pak vyskytnou na místě pravých stran.
- Řešení pomocí inverzní matice: nechť zadáním je rovnice (9.5), pak výsledek soustavy podle této metody můžeme počítat: $\vec{\boldsymbol{x}} = \boldsymbol{A^{-1}} \times \vec{\boldsymbol{b}}$ (7.7)
- Cramerovo pravidlo: výsledek dostaneme postupním nahrazováním sloupců matice se sloupci pravých stran a podílem determinantů těchto matic.

Příklad: Řešit následující soustavu rovnic.

$$\begin{array}{rcrcl} x_1 & + & 2x_2 & = & 4 \\ 2x_1 & + & 3x_2 & = & 7 \end{array}$$

1. **Gaussova eliminační metoda (bez pivotace)**

$$\begin{pmatrix} 1 & 2 & | & 4 \\ 2 & 3 & | & 7 \end{pmatrix} \approx \begin{pmatrix} 1 & 2 & | & 4 \\ 0 & -1 & | & -1 \end{pmatrix} \Rightarrow \boldsymbol{x_2 = 1};\ x_1 + 2 \times 1 = 4 \Rightarrow \boldsymbol{x_1 = 2}$$

2. **Gaussova eliminační metoda (s pivotaci)**

$$\begin{pmatrix} 1 & 2 & | & 4 \\ 2 & 3 & | & 7 \end{pmatrix} \approx \begin{pmatrix} 2 & 3 & | & 7 \\ 0 & 1 & | & 1 \end{pmatrix} \Rightarrow \boldsymbol{x_2 = 1};\ x_1 + 2 \times 1 = 4 \Rightarrow \boldsymbol{x_1 = 2}$$

3. **Gauss-Jordanova metoda**

$$\begin{pmatrix} 1 & 2 & | & 4 \\ 2 & 3 & | & 7 \end{pmatrix} \approx \begin{pmatrix} 1 & 2 & | & 4 \\ 0 & -1 & | & -1 \end{pmatrix} \approx \begin{pmatrix} 1 & 0 & | & 2 \\ 0 & 1 & | & 1 \end{pmatrix} \Rightarrow \boldsymbol{x_2 = 1};\ \boldsymbol{x_1 = 2}$$

4. **Pomocí inverzní matice**

$$A^{-1} = \begin{pmatrix} -3 & 2 \\ 2 & -1 \end{pmatrix}, \quad A^{-1} \times b = \begin{pmatrix} -3 & 2 \\ 2 & -1 \end{pmatrix} \times \begin{pmatrix} 4 \\ 7 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \Rightarrow \boldsymbol{x_2 = 1};\ \boldsymbol{x_1 = 2}$$

5. **Cramerovo pravidlo**

$\mathrm{detA} = -1$ poněvadž je $\mathrm{detA} \neq 0$, můžeme pokračovat ve výpočtu

$$\mathrm{detA}_1 = \begin{vmatrix} 4 & 2 \\ 7 & 3 \end{vmatrix} = -2, \quad x_1 = \frac{\mathrm{detA}_1}{\mathrm{detA}} = \frac{-2}{-1} = 2$$

$$\mathrm{detA}_2 = \begin{vmatrix} 1 & 4 \\ 2 & 7 \end{vmatrix} = -1, \quad x_2 = \frac{\mathrm{detA}_2}{\mathrm{detA}} = \frac{-1}{-1} = 1$$

[8] Pro více informace o této metodě viz http://www.karlin.mff.cuni.cz/~tuma/2002/NLinalg2.pdf

8 Polynomy

Polynom neboli mnohočlen je výraz sestávající jen ze součtů (rozdílů), násobků a celočíselných mocnin proměnných. Dále se ale budeme zabývat jen polynomy jedné proměnné x, které můžeme zapsat v obecném tvaru:

$$\boldsymbol{p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0} \quad \textbf{(8.1)}$$

nebo zkráceně:

$$\boldsymbol{p(x) = \sum_{i=0}^{n} a_i x^i} \quad \textbf{(8.2)}$$

Konstanty a_0 až $a_n$jsou koeficienty a může se jednat o libovolná reálná nebo komplexní čísla.

8.1 Hornerovo schéma

Hornerovo schéma neboli hornerova metoda je v aritmetice algoritmus pro efektivní vyhodnocování polynomů. Pomocí něho můžeme zjišťovat hodnotu polynomu v daném bodě, otestovat jestli daný bod je řešením polynomu nebo můžeme polynom dělit lineárním polynomem. Dále nám umožňuje vypočítat derivace v konkrétním bodě.

Tabulka 11 - Znázornění hornerova schématu

x	a_n	a_{n-1}	…	a_1	a_0
x_0	1	$x_0 \times a_n + a_{n-1}$	…	$x_0 \times a_2 + a_1$	$x_0 \times a_1 + a_0$

Pro vyčíslení polynomu a otestování kořenu, daný bod dosadíme za $\boldsymbol{x_0}$ v tabulce 10 a polynom vyhodnotíme. Hodnota ve sloupci a_0 vyjadřuje funkční hodnotu v daném bodě. Jestli tato hodnota se rovná nule, daný bod je zároveň kořenem polynomu.

Nechť úkolem je dělit polynom $\boldsymbol{p(x) = \sum_{i=0}^{n} a_i x^i, \quad a_n \neq 0}$ polynomem $\boldsymbol{q(x) = b_1 x + b_0, b_1 \neq 0}$. Abychom mohli použít hornerovo schéma pro řešení tohoto úkolu, polynom $\boldsymbol{q(x)}$ musí být prvního stupně. Jinak tento algoritmus neplatí. Pokud je $q(x)$ polynomem prvního stupně, opět dosadíme hodnotu $\boldsymbol{-b_0/b_1}$ za $\boldsymbol{x_0}$ v tabulce 10 a výsledek tohoto úkolu bude $\boldsymbol{a_n x^{n-1} + \cdots + a_1 + a_0(b_1 x + b_0)}$.

Příklad: Pro polynom $p(x) = x^3 - 3x + 2$

a) zjistit funkční hodnotu v bodě 1
b) otestovat jestli 1 je kořenem tohoto polynomu
c) dělit polynomem $q(x) = 2x - 2$

Řešení: Přepíšeme všechny koeficienty polynomu do tabulky. Za chybějící koeficienty píšeme nulu. Pro řešení všech třech úkolů, stačí vyhodnotit polynom v bodě 1.

Tabulka 12 - Výsledek polynomu

x	1	0	-3	**2**
1	**1**	1 × 1 + 0 = **1**	1× 1 + (-3) = **-2**	1× (-2) + 2 = **0**

a) funkční hodnota v bodě 1 je nula
b) bod 1 je kořenem polynomu
c) $q(x) = 2x - 2 \implies x_0 = \frac{2}{2} = 1$

$x^3 - 3x + 2 = (x - 1)(x^2 + x - 2)$

9 Analýza zadání

Vzhledem k tomu, že existuje mnoho problémů při výpočtu s pohyblivou řádovou čárkou a mnoho z těch problémů nastává kvůli špatnému vstupnímu datu, rozhodla jsem si vytvořit knihovnu nového datového typu, který reprezentuje číslo s pohyblivou řádovou čárkou a umožňuje provádět základní aritmetické operace. Výsledný datový typ by měl umožnit větší rozsah čísel než existující formáty. Dalším cílem je vytvořit obslužnou aplikaci, která využívá tuto knihovnu pro svoji činnost a testovat jak reaguje při různých výpočtech numerických úloh.

9.1 Možné implementace

Existují několik možností jak implementovat nový datový typ s pohyblivou řádovou čárkou. Níže uvedu 3 možností, které se nejvíce používá.

- binární řádová čárka
- reprezentovat mantisu v soustavě BCD
- desetinná řádová čárka

První možnost binární řádová čárka, se spíš používá pro hardwarovou implementaci, když jde o to, dostat co nejpřesnější výsledek tak to není dobrý nápad. Totiž kromě čísel s násobky 2 ostatní čísla nemůžou být přesně uložena do binárních formátů. A navíc implementace binární čárky na softwaru se mi zdálo pomalé a složité, i když provádění aritmetické operace s nimi může být rychlejší. Další možnost BCD může být sice přesnější než čisté binární reprezentace, ale vyžaduje větší pamětní prostor, a opět vyžaduje konverzi mezi číselnými soustavami. Poslední možnost realizace řádové čárky mi přišlo jako nejvýhodnější. Přesnost a správnost s desetinnou čárkou je vyšší než předchozí dvě. Z tohoto důvodu i mnoho finančních softwarů používá právě tu desetinnou čárku. Nevýhodou softwarové implementace je, že sto až tisíc krát pomalejší než hardwarové implementace.

9.2 Výsledek

Po pečlivém zvážení všech faktorů jsem si nakonec rozhodla pro implementace desetinné pohyblivé čárky. Pro testování a porovnání správnosti nového datového typu jsem si rozhodla obslužné aplikaci umožnit pracovat ještě s datovým typem **double**.

9.3 Základní požadavky aplikace

Zde jsou uvedené základní funkcionality, které by měla umožnit výsledná aplikace:

- Kalkulátor – výpočet se základními aritmetickými operacemi a navíc umožnit libovolně závorkovat výraz.
- Maticové výpočty – unární a binární operace.
- Řešit soustavu rovnic – na výběr několik metod.
- Vyhodnotit polynom – normální vyčíslení, pomocí hornerova schématu.

10 Použité technologie

Pro vytvoření aplikace byly zvolené jazyk Java a vývojové prostředí Netbeans[9] IDE 6.7.1. Pro Javu jsem se rozhodla kvůli její velké rozšířenosti, kvalitní dokumentaci, objektovému orientovanému přístupu a hlavně její přenositelnosti. K vizuálnímu vyjádření analýzy a grafické znázornění jednotlivých tříd je používán jazyk UML.

10.1 Java

Java je objektově orientovaný programovací jazyk nezávislý na platformě, který vyvinul společnost SUN Microsystems. Nezávislost na operačním systému a na hardwaru počítače zajišťuje jeho způsob kompilace. Zdrojové kódy programu jsou předzpracovávány do tzv. bytecode, který je dále interpretován a spojen s příslušnými Java knihovnami. Při spuštění programu je bytecode převeden na strojový kód daného procesoru, to provádí tzv. Java Virtual Machine (JVM) [25]. A navíc na webových stránkách lze použít Javu v podobě appletů. Java applet je aplikace, které je na stránce vyhrazen obdélníkový prostor používaný pro komunikaci s uživatelem.

10.2 UML

UML neboli Unified Modeling Language je jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů. UML zahrnuje širokou řadu dostupných diagramů [10], z nichž jsem využila pouze některé, a to Class, Package a Use Case Diagram. K vytvoření těchto diagramů bylo využito UML PlugIn pro Netbeans IDE.

Relace

Vzhledem k tomu, že v grafech je zapotřebí předměty různým způsobem navzájem propojovat, jsou v jazyku UML specifikovány i relace, tj. vztahy mezi různými předměty [28].

- závislost (*dependency*) znázorňuje vztah, kdy změnou v jednom elementu je ovlivněn jiný (závislý) element
- asociace (*association*) je nejobecnější vztah mezi třídami, kdy objekty jedné třídy posílají zprávy objektům druhé třídy. Speciální variantou asociace jsou tzv. kompozice a agregace.
- zobecnění (*generalization*) je stav, kdy jeden element je specializací jiného elementu a je často používána v OOP, implementuje se pomocí dědičnosti.

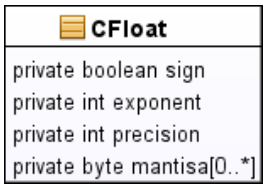
Diagramy

- Diagram tříd (*Class diagram*) patří do skupiny UML diagramů, které popisují strukturu systému, vyjadřuje třídy, které se v systému vyskytují a vztahy mezi nimi.
- Diagram balíčku (*Package diagram*) patří také mezi strukturní diagramy a ukazuje, jak jednotlivé třídy jsou rozděleny do balíčků.
- Diagram užití (*Use case diagram*) patří do skupiny diagramů chováni. Tento diagram zachycuje interakce mezi systémem a aktéry (nejčastěji uživateli), kteří od systému vyžadují nějakou akci.

[9] Více informace na http://netbeans.org/

11 Popis implementace

V této kapitole věnuju popisu datového typu CFloat, který jsem vytvořila. Tento typ umožňuje vytvořit čísla s proměnlivou přesností. Třída CFloat obsahuje konstanty určující speciální hodnoty: *CMINUS_INFINITY*, *CPLUS_INFINITY*, *CZero*, *CNaN*. Dále obsahuje statické metody pro vytvoření nuly a jedniček: *zero(), one(boolean sign)*. Každé číslo typu CFloat se skládá ze čtyř částí.



Obrázek 4 - CFloat

- *sign* – znaménko daného čísla (true pro záporné číslo, false pro kladné číslo)
- *exponent* – exponent daného čísla (max = 2147483647, min = 2147483647)
- *precision* – vyjadřuje maximální počet nenulových číslic v mantise (kromě vedoucích a koncových nul)
- *mantisa* – uchová ciferné hodnoty daného čísla a je vždy normalizovaná

Kromě základních aritmetických operací (sčítání, odčítání, násobení a dělení), umožňuje i některé další metody:

- **abs()** – vrací absolutní hodnotu daného čísla
- **negate()** – vrací hodnotu s opačným znaménkem
- **increment()** – inkrementuje dané číslo o jedno
- **round(int count)** – zaokrouhlí číslo na ***`count`*** platných cifer
- **roundD(int countDec)** – zaokrouhlí číslo na ***`countDec`*** desetinných míst
- **compareTo(CFloat val)** – porovnává toto číslo s číslem ***`val`***

11.1 Existující implementace desetinné pohyblivé čárky

Dnes už existuje mnoho alternativ s desetinnou pohyblivou čárkou. Většina z nich je podporovaná v softwaru. Firma IBM podporuje několik hardwarových implementací. Některé softwarové implementace:

- **BigDecimal** je balík pro jazyk Java a používá pevnou řádovou čárku.
- **Dfp**[10] je knihovna napsaná v Javě a používá bázi 10000.
- **System.Decimal** je knihovna pro Microsoft C# nebo .NET. Rozsah čísel v tomto formátu je **±1.0 × 10−28 až ±7.9 × 1028** a přesnost je 28-29 číslic.
- **Decimal** [11] je modul pro Python. Umožňuje nastavit IEEE trapy a příznaky.

[10] Více informace na http://dfp.sourceforge.net/
[10] Více informace na http://pydoc.org/2.4.1/decimal.html
[11] Více informace na http://docs.python.org/library/decimal.html

12 Závěr

Výsledná aplikace je plně funkční téměř na každé myslitelné platformě, bez nutnosti její nové kompilace nebo jiných změn pro dané prostředí. Splňuje všechny požadavky, které jsou uvedené v zadání. Při tvorbě aplikace jsem zaměřila na to aby, vznikla aplikace s vysokou přesností. Nešlo mi tedy o výkon, jakého se dosahují profesionální aplikace.

Během vytvoření aplikace se vyskytlo několik problémů, které se podařilo úspěšně odstranit. Hlavním problémem byl výpočet exponentu u operace dělení ve třídě CFloat. Důvodem bylo to, že u této operace nelze odhadnout závislost mezi přesností výsledku a exponentem jako u ostatních operací. Ostatní problémy byly jen drobné chyby, které vznikly při implementaci.

Celá aplikace je napsána v jazyce Java. Nevýhodou použití jazyka Java je v tom, že neumožňuje přetěžovat operátory. Další nevýhodou je nutnost instalovat na počítač Java Runtime Environment, který umožňuje výše uvedenou přenositelnost mezi různými platformami. Aplikace má také o trochu vyšší hardwarové nároky na počítač, na rozdíl od aplikace, která se překládá přímo do spustitelného kódu.

Jako přínos této práce bych označila relativně jednoduché ovládání a vysoká přesnost výpočtu oproti výpočtu s datovým typem double.

Při tvorbě bakalářské práce jsem se naučila mnoho nových programátorských dovedností a rozšířila si teoretické i praktické znalosti o počítačové aritmetice.

V případném dalším vývoji nového datového typu *CFloat* by bylo možné implementovat další metody, jako jsou mocnina, odmocnina, atd. U *rovnice* bych doplnila metody umožňující parametrizace a tím řešit soustavy s více řešeními. Rozšířením části *polynomu* by mohlo být automatické vypočítání všech řešení nebo vykreslování grafů z polynomů.

Literatura

[1] *Floating Point* In Wikipedia: the free encyclopedia [online]. Poslední změna: 26. 7. 2010 [cit. 2010-07-31]. Anglická verze. Dostupné z WWW: <http://en.wikipedia.org/wiki/Floating_point>.

[2] *Vytvoření a rozvoj týmu pro náročné technické výpočty na paralelních počítačích na TU v Liberci* [online]. [2009?] [cit. 2010-6-20]. Dostupné z WWW: <http://www.fp.vslib.cz/kmd/EN/ESF1615/anotace.pdf>.

[3] IBM Corporation. *Decimal Arithmetic FAQ Part2 - Definitions* [online]. Poslední změna: 19. 4. 2010 [cit. 2010-07-30]. Dostupné z WWW: <http://speleotrove.com/decimal/decifaq2.html#precision>.

[4] **CÍCHA, Tomáš.** *Řídké matice a jejich použití v numerické matematice* [online]. 1. 6. 2009 [cit. 2010-07-30]. Dostupné z WWW: <http://is.muni.cz/th/207863/prif_b/sparse_matrices.pdf>.

[5] *IEEE 754-2008* In Wikipedia: the free encyclopedia [online]. Poslední změna: 26. 7. 2010 [cit. 2010-07-26]. Dostupné z WWW: <http://en.wikipedia.org/wiki/IEEE_754-2008>.

[6] **TIŠŇOVSKÝ, Pavel**. *Fixed point arithmetic* [online]. 24. 5. 2006 [cit. 2010-07-04]. Dostupné z WWW: <http://www.root.cz/clanky/fixed-point-arithmetic/>.

[7] IBM Corporation. *Arithmetic operations* [online]. 7. 4. 2009 [cit. 2010-08-05]. Dostupné z WWW: <http://speleotrove.com/decimal/daops.html>.

[8] **GOLDBERG, David**. *What Every Computer Scientist Should Know About Floating-Point Arithmetic* [online]. 1991 [cit. 2010-08-05]. Dostupné z WWW: <http://docs.sun.com/source/806-3568/ncg_goldberg.html>.

[9] **BAYER, Tomáš**. *Výpočet hodnoty aritmetického výrazu* [online]. [200?] [cit. 2010-08-05]. Dostupné z WWW: <http://web.natur.cuni.cz/~bayertom/Prog2/prog2_3.pdf>.

[10] *Unified Modeling Language* Wikipedia: otevřená encyklopedie [online]. Poslední změna: 15. 7. 2010 [cit. 2010-08-05]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Unified_Modeling_Language>.

[11] **PAJONK, Tomáš.** *UML-Pajonk* [online]. 2001 [cit. 2010-08-05]. Dostupné z WWW: <http://users.fs.cvut.cz/~jurajaku/pis/materialy/UML-Pajonk.doc>.

[12] *Single precision floating-point format* In Wikipedia: the free encyclopedia [online]. Poslední změna: 17. 5. 2010 [cit. 2010-08-05]. Anglická verze. Dostupné z WWW: <http://en.wikipedia.org/wiki/Single_precision_floating-point_format>.

[13] *Zaokrouhlení* Wikipedia: otevřená encyklopedie [online]. Poslední změna: 28. 6. 2010 [cit. 2010-08-05]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Zaokrouhlení>.

[14] **RÁK, Josef**. *Základy programování – Mantisa a Pole* [Přednáška]. Pardubice: Univerzita Pardubice. 2008 [cit. 2010-07-31].

[15] *Gaussova eliminace* [online]. 2002 [cit. 2010-07-05]. Dostupné z WWW: <http://www.karlin.mff.cuni.cz/~tuma/2002/NLinalg2.pdf>.

[16] **MARTINÁKOVÁ, Miroslava.** *Choleského metoda* [online]. 2008 [cit. 2010-08-05]. Dostupné z WWW: <http://mant.upol.cz/soubory/OdevzdanePrace/B08/b08-10-mm.pdf>.

[17] **ČAMBOR, Michal.** *Elementární procesor v aritmetice pevné a pohyblivé řádové čárky* [online]. 2009 [cit. 2010-08-05]. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=9087>.

[18] *Matice a maticová algebra, soustavy lineárních rovnic, kořeny polynomu a soustava nelin. rovnic* [online]. [200?] [cit. 2010-08-05]. Dostupné z WWW: <http://kfes-16.karlov.mff.cuni.cz/~standa/matlab/matlab4.pdf>.

[19] **COWLISHAW, Michael.** *Decimal Floating-Point: Algorism for Computers* [online]. 2003 [cit. 2010-08-05]. Dostupné z WWW: <http://speleotrove.com/decimal/IEEE-cowlishaw-arith16.pdf>.

[20] **KUBÁTOVÁ.** *Čísla se znaménkem a aritmetické operace pevná a pohyblivá řádová čárka* [online]. 2007 [cit. 2010-08-05]. Dostupné z WWW: <http://michal.myserver.cz/cvut-fel/SAP/sap-6-oper.pdf>.

[21] **NETRVALOVÁ, A.** *Zobrazení čísel v počítači – příklady* [online]. [200?] [cit. 2010-08-05]. <Dostupné z WWW: <http://www.kiv.zcu.cz/~netrvalo/vyuka/ppa1-06/cviceni/materialy/PrikladyZobrazeniCisel.pdf>.

[22] **LIMPOUCH, Jiří.** *Úvod do numerické matematiky* [online]. 1. 3. 1999 [cit. 2010-08-05]. Dostupné z WWW: <http://kfe.fjfi.cvut.cz/~limpouch/numet/foluvux/>.

[23] **SEDGEWICK, Robert, WAYNE, Kevin.** *Floating Point* [online]. [200?] [cit. 2010-08-05]. Dostupné z WWW: <http://www.cs.princeton.edu/introcs/91float/>.

[24] **VOJÁČEK, Jakub.** *Hornerovo schéma* [online]. 14. 3. 2009 [cit. 2010-08-05]. Dostupné z WWW: <http://maths.cz/clanky/hornerovo-schema.html>.

[25] **EUBANKS, Brian.** *Java na maximum.* Brno: Computer Press, a.s., 2006. ISBN 80-251-1111-3

[26] **Hollasch, Steve.** *IEEE Standard 754 Floating Point Numbers* [online]. Poslední změna: 24. 2. 2005 [cit. 2010-08-05]. Dostupné z WWW: <http://steve.hollasch.net/cgindex/coding/ieeefloat.html>.

[27] **SEDGEWICK, Robert, WAYNE, Kevin.** *Numerical linear algebra* [online]. [200?] [cit. 2010-08-05]. Dostupné z WWW: <http://www.cs.princeton.edu/introcs/95linear/>.

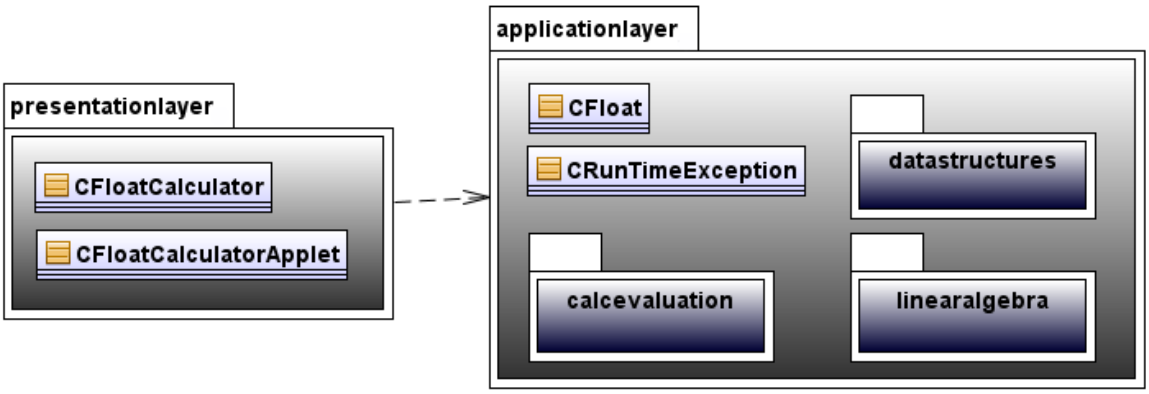
[28] **TIŠŇOVSKÝ, Pavel.** *Nástroje pro tvorbu UML diagramů* [online]. 17. 4. 2005 [cit. 2010-08-05]. Dostupné z WWW: <http://www.root.cz/clanky/nastroje-pro-tvorbu-uml-diagramu/#k04>.

[29] *Computational complexity of mathematical operations* In Wikipedia: the free encyclopedia [online]. Poslední změna: 18. 3. 2010 [cit. 2010-08-05]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations>.

[30] **KOTLÍKOVÁ, Michala.** *Zobrazení čísel v počítači* [online]. [200?] [cit. 2010-08-05]. Dostupné z WWW: <http://www.kiv.zcu.cz/~netrvalo/vyuka/ppa1-04/ZobrazeniCisel.pdf>.

Příloha A – Programátorská dokumentace

Zde uvedu jen krátký popis balíčků a tříd, které jsou v nich obsažené. Pro podrobný popis všech metod a tříd viz generovány JavaDoc v adresáři `Calculator/doc`. Celá aplikace je rozdělená do dvou balíčků **applicationlayer** a **presentationlayer**.



Obrázek 5 - Diagram balíčku celé aplikace, zdroj: vlastní

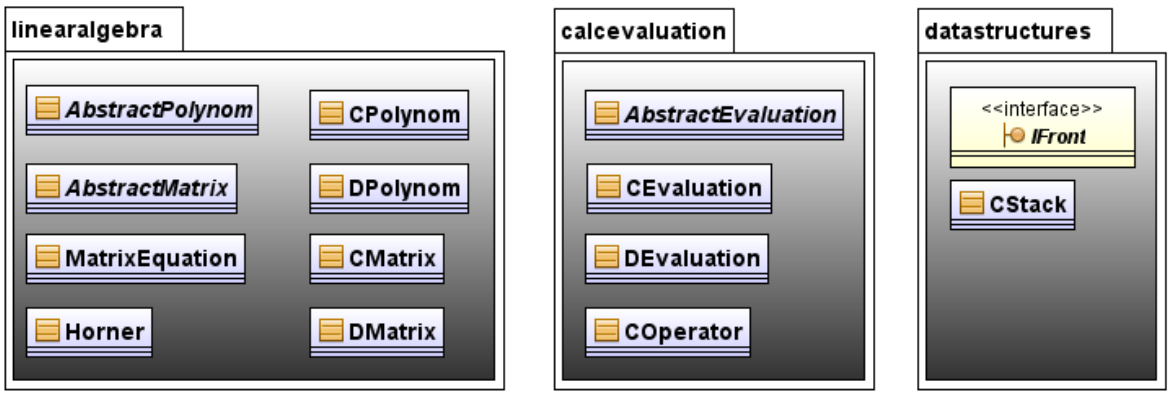
Presentationlayer

Prezentační vrstva neboli balíček **presentationlayer** obsahuje dvě třídy – *CFloatCalculator* (JFrame) a *CFloatCalculatorApplet* (JApplet). Jednotlivé třídy pak obsahují instance tříd z balíčku applicationlayer, kromě toho obsahují obrovské množství metod pro zobrazování GUI. V presentationlayer příjme výraz a data od uživatele a předává aplikační vrstvě k vyhodnocení. Po vyhodnocení výrazu zobrazí výsledek, případně chybu, pokud byl výraz nekorektní.

Applicationlayer

Aplikační vrstva neboli balíček **applicationlayer**, obsahuje v sobě dvě třídy a další tři balíčky. Význam těch tříd a balíčků je uveden níže.

Diagram balíčků v applicationlayer

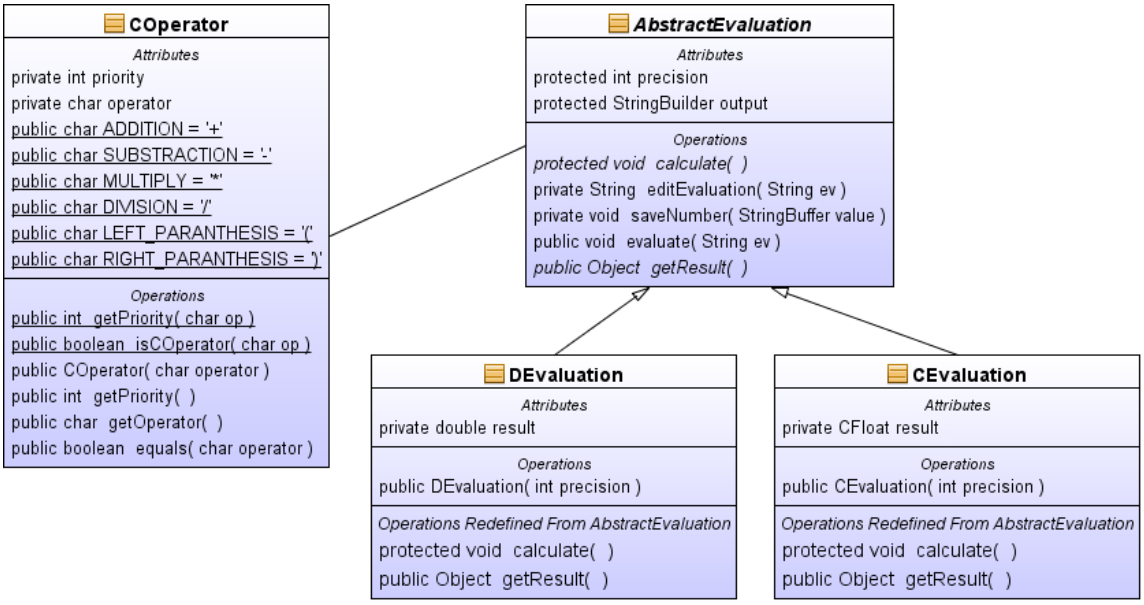


Obrázek 6 - Diagram balíčků v applicationlayer, zdroj: vlastní

Diagram tříd v balíčku „presentationlayer“. Třída ***CFloat*** představuje nově vytvořený datový typ. Třída ***CRunTimeExpcetion*** je potomkem výjimky *RunTimeException*.

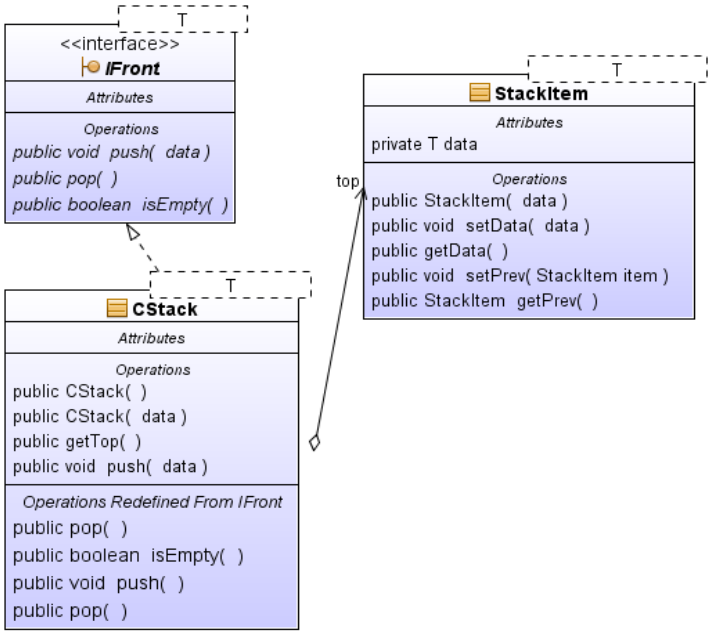
Obrázek 7 - Diagram tříd v balíčku applicationlayer, zdroj: vlastní

Diagram tříd v balíčku „applicationlayer.calcevaluation“. Třídy v tomto balíčku slouží k vyhodnocení aritmetických výrazů. Abstraktní třída ***AbstractEvaluation*** příjme výraz v infixové notaci a převede na postfixovou[12] notaci a výsledný výraz nastaví na **output.** Pak třídy ***DEvaluation*** (pro double) a ***CEvaluation*** (pro CFloat) vypočítá hodnoty výrazu z output a výsledek nastaví na **result.**



Obrázek 8 - Diagram tříd v balíčku calcevaluation, zdroj: vlastní

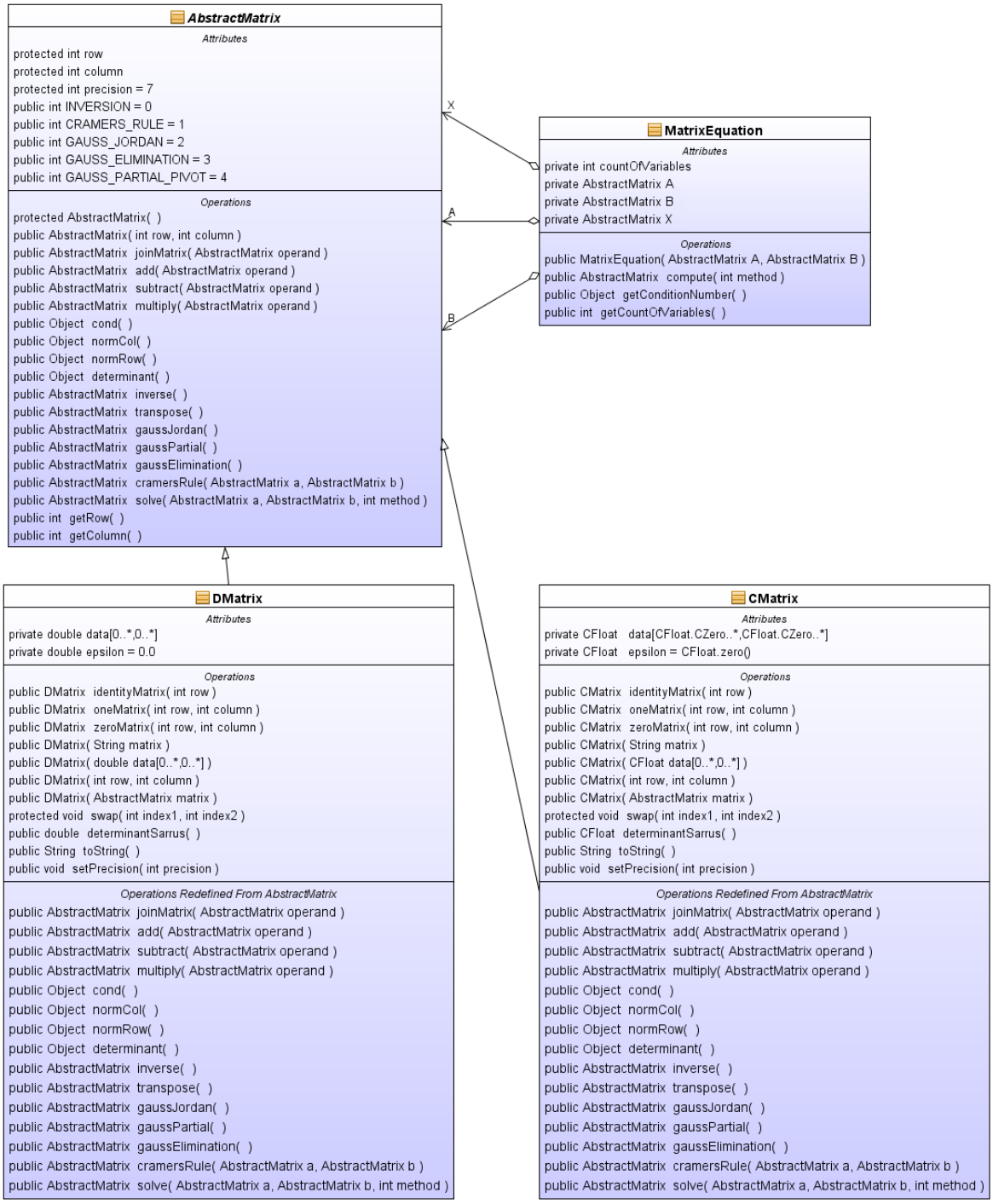
Diagram tříd v balíčku „applicationlayer.datastructures“. Balíček obsahuje rozhrání **IFront**, které implementuje třída ***CStack***. ***StackItem*** je vnitřní třídou třídy CStack. Třída CStack představuje zásobník a slouží k uložení hodnot výrazů.



Obrázek 9 - Diagram tříd v balíčku datastructures, zdroj: vlastní

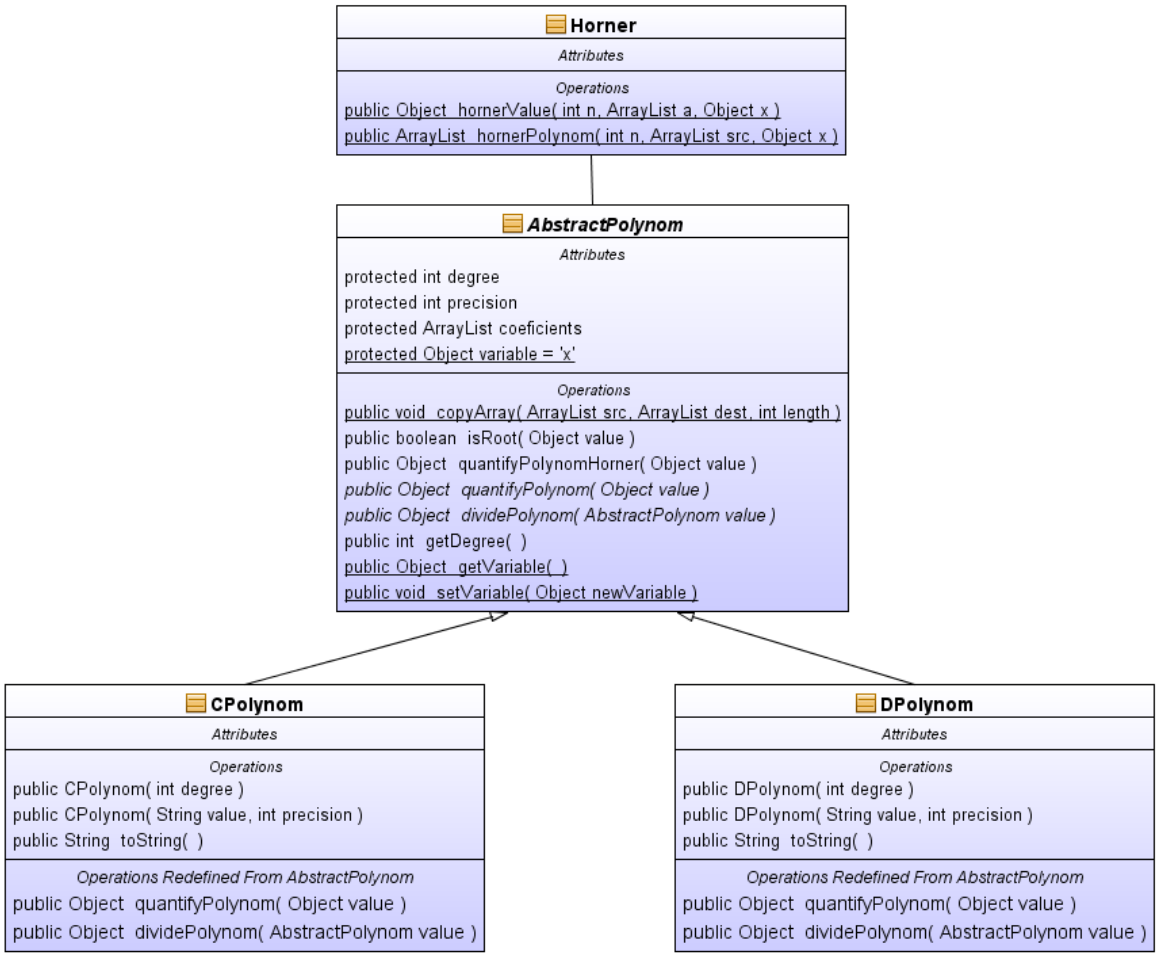
[12] Více informace na http://web.natur.cuni.cz/~bayertom/Prog2/prog2_3.pdf

Diagram tříd pracující s maticí v balíčku „applicationlayer.linearalgebra“. Abstraktní třída AbstractMatrix obsahuje základní strukturu matice a definice některých metod. Jeho potomek třída DMatrix představuje matici s double hodnoty a CMatrix představuje matici s CFloat hodnoty. Atribut epsilon slouží jako přesnost stroje a je vypočítán vůči nastavené přesnosti. Třída MatrixEquation představuje soustavu rovnic a zatím pracuje jen se soustavou, která má právě jedno řešení. V ostatním případě je volána výjimka. Pro výpočet řešení se používá metody z třídy AbstractMatrix.



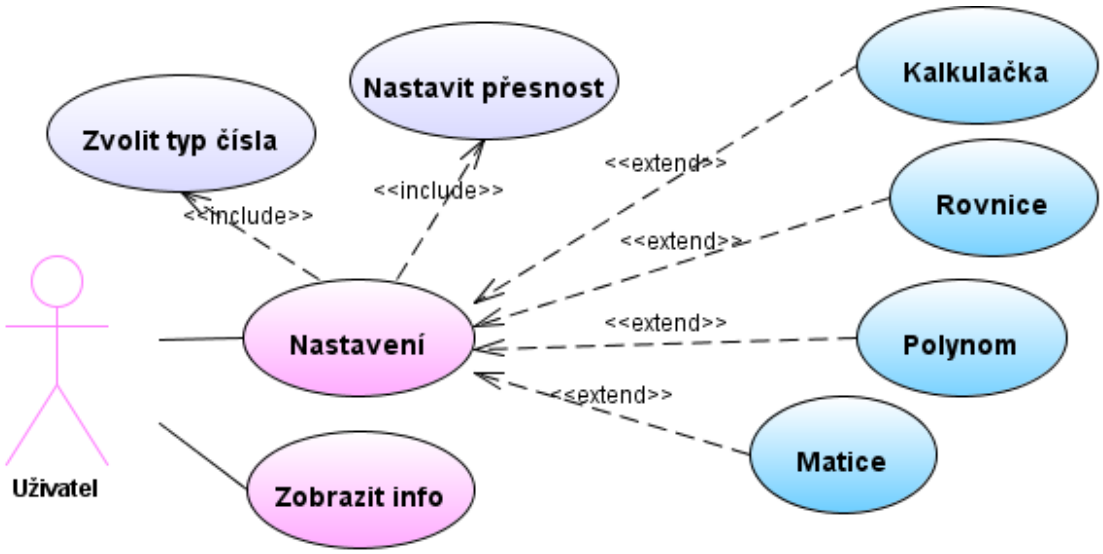
Obrázek 10 - Diagram tříd pro práci s maticí v balíčku linearalgebra, zdroj: vlastní

Diagram tříd pracující s polynomy v balíčku „applicationlayer.linearalgebra“. Finální třída ***Horner*** obsahuje jen dvě metody pro vyhodnocení polynomu pomocí hornerova schématu. Abstraktní třída ***AbstractPolynom*** tu opět představuje základní strukturu polynomu a obsahuje definice některých metod. Jeho potomek třída ***DPolynom*** pracuje s double hodnotami a ***CPolynom*** s *CFloat* hodnotami.



Obrázek 11 - Diagram tříd pro práci s polynomem v balíčku linearalgebra, zdroj: vlastní

Diagram užití



Obrázek 12 - Diagram užití, zdroj: vlastní

Příloha B – Uživatelská dokumentace

B. 1 Úvod

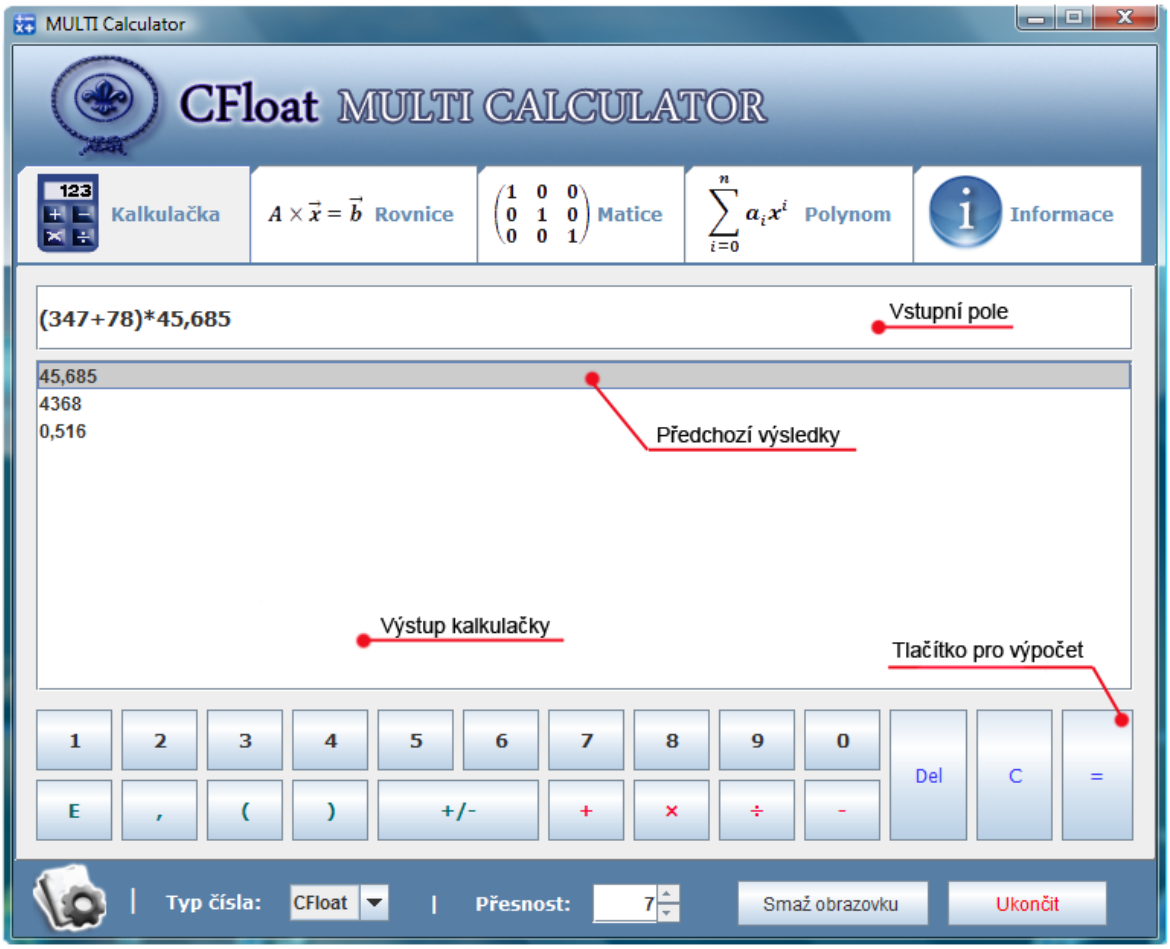
Tady budu věnovat popisu obsluhy programu z pohledu čtenáře bakalářské práce jako uživatele příslušné úrovně. Program je možné spustit na libovolné platformě, například na Microsoft Windows, většině distribucí Linuxu a dalších, jedinou podmínkou je instalace Javy. Na všech systémech je možné program spustit pomocí příkazu `java -jar Calculator.jar`, který zobrazí navíc konzoli systému. Na většině systémů by bylo možné spustit program také pomocí příkazu `javaw -jar Calculator.jar`, který již konzoli nezobrazuje. V případě správně nakonfigurovaného systému stačí poklepat na spustitelný *jar* soubor a program se spustí.

Celé okno lze rozdělit na dvě hlavní části: výpočetní část a část nastavení. Výpočetní část dále obsahuje pět položek. U všech výpočetních částí je možno nastavit přesnost a typ čísla, se kterými chceme pracovat.

Obrázek 13 - Nastavení, zdroj: vlastní

B. 2 Kalkulačka

V této části si můžeme zadávat libovolný aritmetický výraz, který kalkulačka podporuje a libovolně je uzavírat do kulatých závorek. Do vstupního pole si výraz můžeme zadat pomocí klávesnice nebo tlačítka v dolní části. Jednotlivé výsledky jsou uloženy do výstupního pole, které můžou sloužit jako operand v dalším výrazu. To se uskuteční vybráním dané hodnoty myší. Po zadání výrazu zahájíme výpočet tlačítkem „=“.



Obrázek 14 – Kalkulačka, zdroj: vlastní

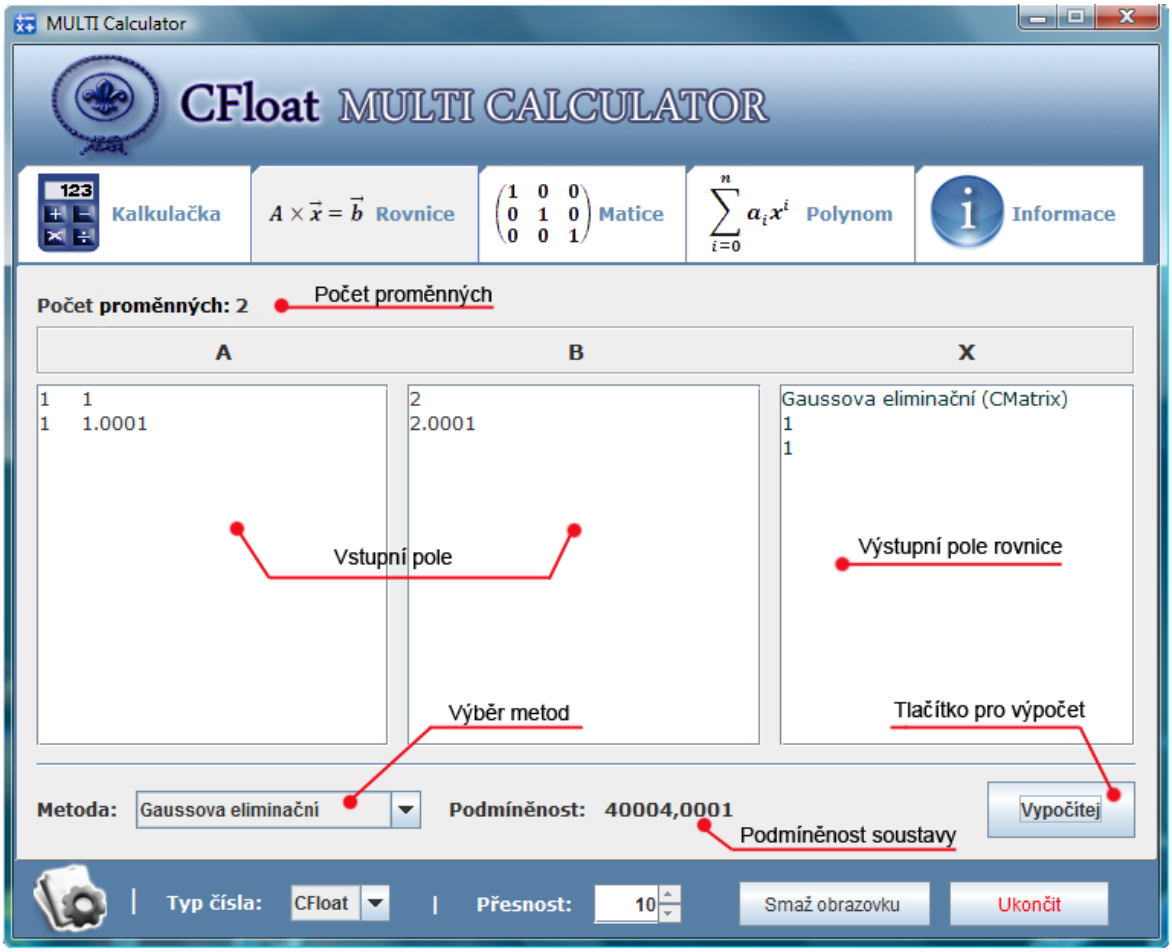
B. 3 Rovnice

Tato část je určená pro řešení soustav lineárních rovnic, kde soustava má právě jedno řešení. Hodnoty rovnice zadáme v maticovém tvaru. Tlačítkem **„Vypočítej“** si zahájíme výpočet. Po stisknutí tlačítka „Vypočítej“ se zobrazuje počet proměnných v levém horním rohu a podmíněnost soustavy je zobrazená na dolní části.

Jestli je výpočet proveden úspěšně, budou zobrazené **Typ metody (Typ matice)** a **výsledek** ve výstupním poli. Jinak se podle počtu řešení zobrazuje chybová hláška ve tvaru: **Typ matice (metoda) – soustava nemá řešení** nebo **soustava má nekonečně mnoho řešení.**

Navíc máte možnost si vybrat metodu pro výpočet řešení. Implicitně je nastavena na Gaussovou eliminační metodu. Seznam metod:

- pomocí inverzní matice;
- Cramerovo pravidlo;
- Gauss-Jordanova eliminace;
- Gaussova eliminace;
- Gaussova eliminace s parciální pivotací.



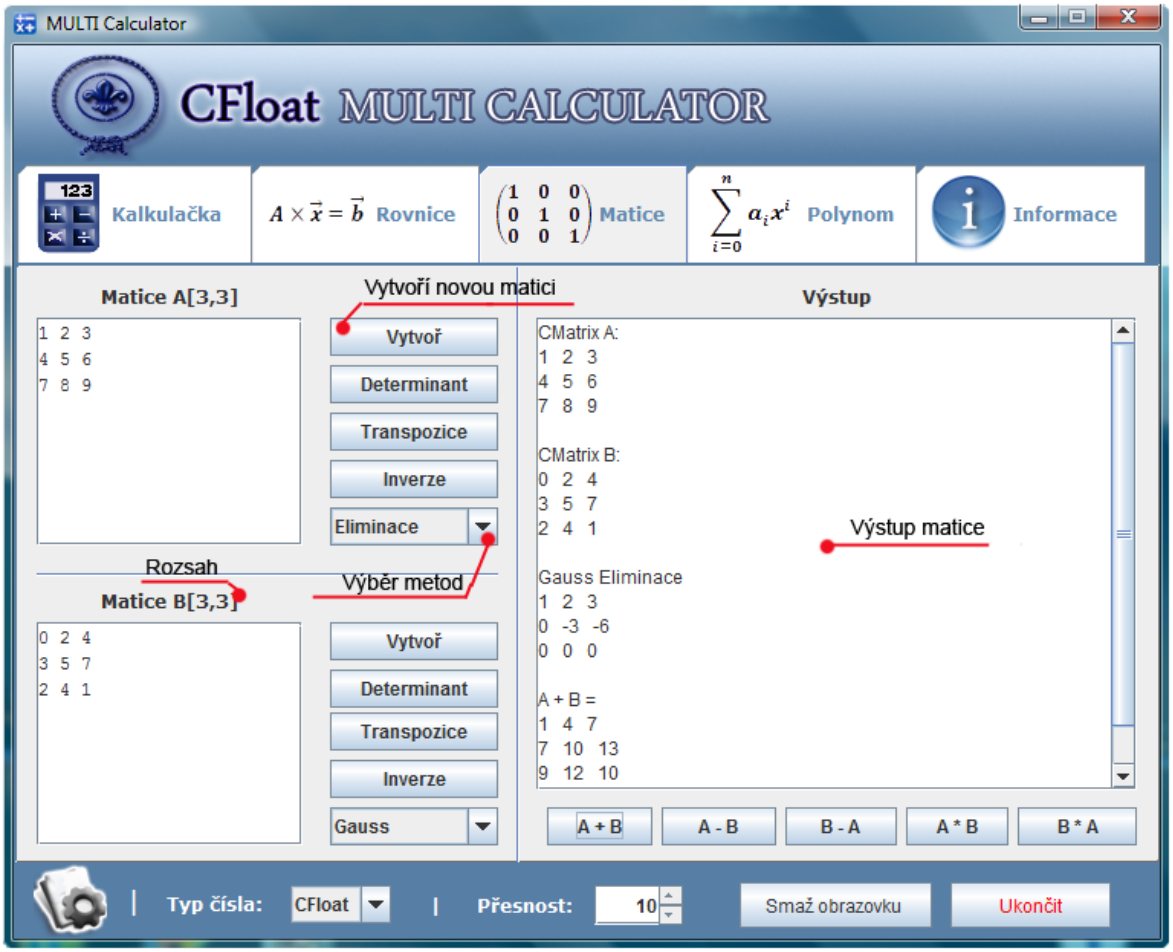
Obrázek 15 – Rovnice, zdroj: vlastní

B. 4 Matice

Zde si můžeme pracovat s dvěma maticemi. K vytvoření matic slouží tlačítko **„Vytvoř“,** po úspěšné vytvoření se zobrazují **Typ** matice a **prvky** dané matice ve výstupním poli. Počet sloupců a řádků matice se zobrazují nad vstupním polí dané matice. U každé matice jsou tlačítka sloužící k práci s tou maticí. Všechny výstupy včetně chybových hlášek jsou zobrazené ve **výstupním poli**. Kromě vypočítání determinantu, inverzní matice si navíc můžeme upravit matici **s Gaussovými metodami**. Tyto úpravy můžou sloužit jako pomůcky pro další úkoly, například pro řešení soustav, kde soustava rovnic má nekonečně mnoho řešení.

Seznam metod:

- Gaussova eliminace;
- Gauss-Jordanova eliminace;
- Gaussova eliminace s parciální pivotací.



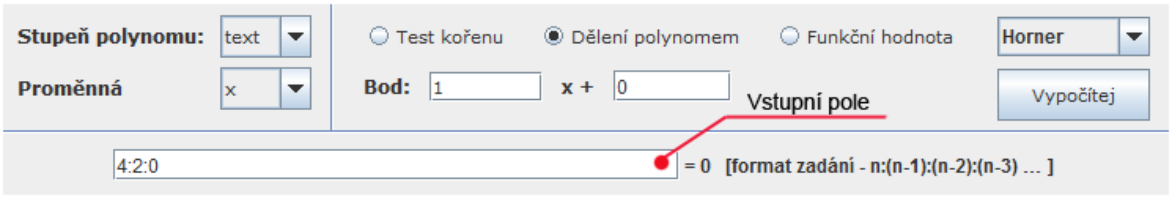
Obrázek 16 – Matice, zdroj: vlastní

B. 5 Polynom

Další část programu je určená pro práci s polynomy. U vyčíslení polynomu máme dvě možností – normální, pomocí hornerova schématu. Navíc si můžeme vybrat znak proměnné použitý pro textové reprezentace polynomu.

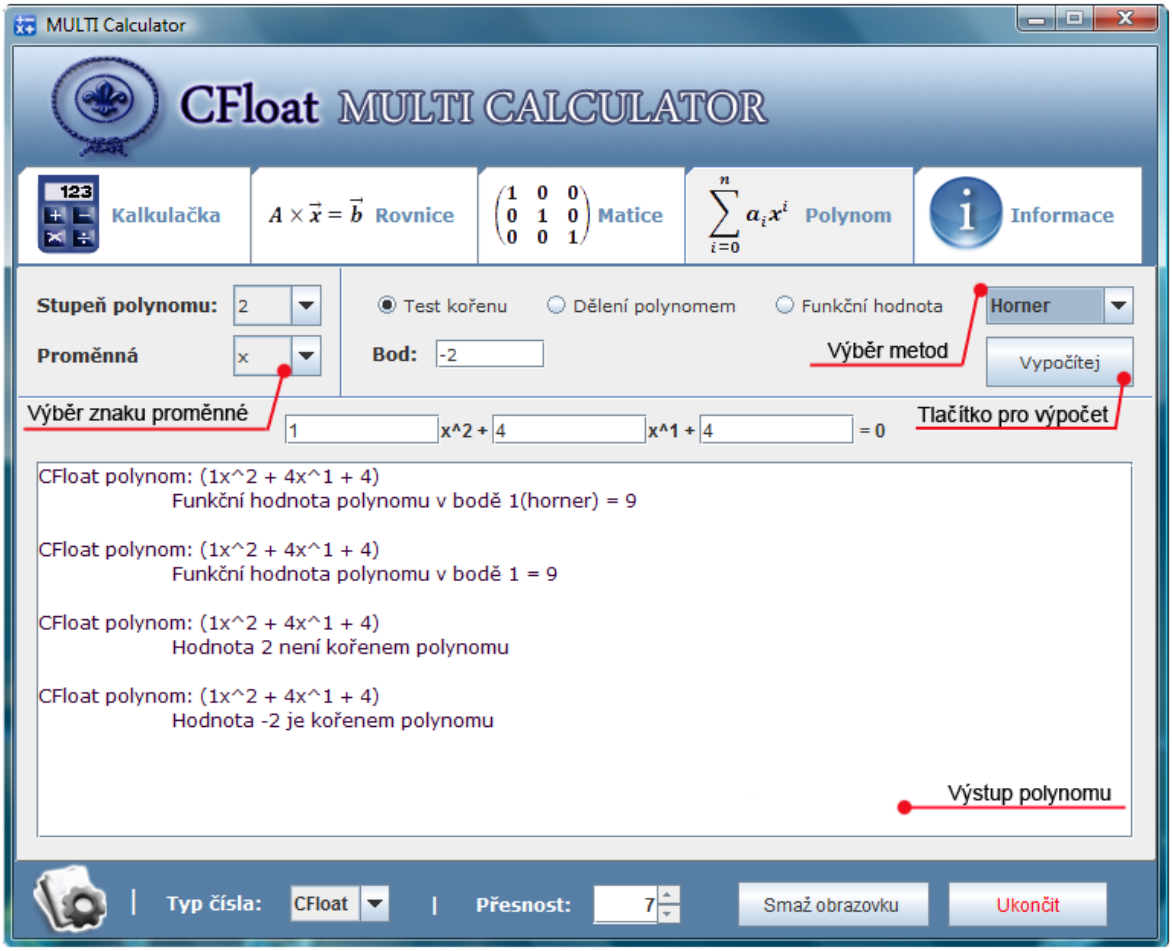
Koeficienty polynomu můžeme zadat dvěma způsoby:

1. První způsob je přímé psaní koeficientů ve formátu „$a_n : a_{n-1} : \ldots : a_0$“ do **vstupního pole**, kde musíme zadat všechny koeficienty polynomu (i s nulami).



Obrázek 17 - Polynom vstup, zdroj: vlastní

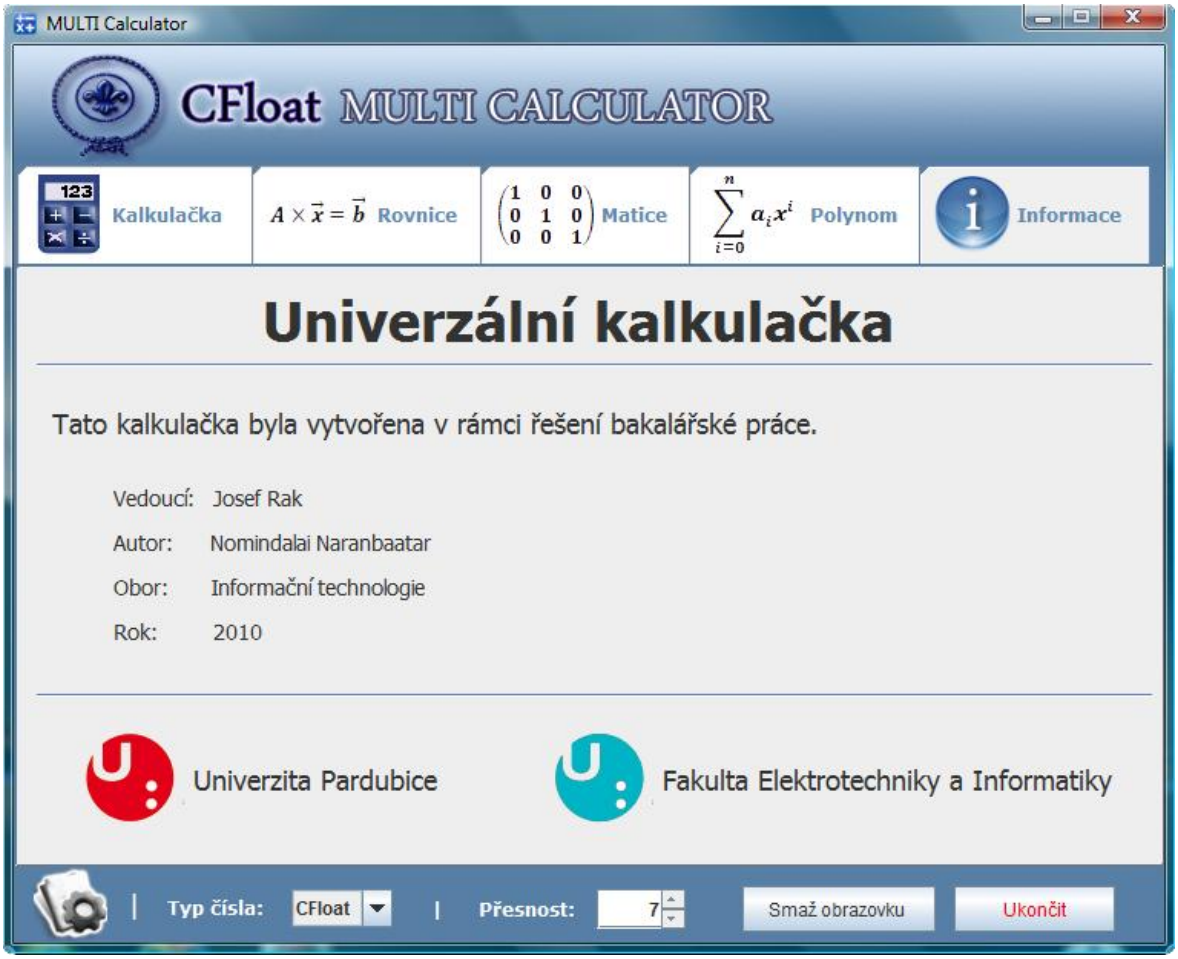
2. Další způsob je zobrazený na dolním obrázku. Zde máme možnost vybrat stupeň polynomu do deseti. Po vybrání se zobrazí vstupní pole, podle stupně polynomu. Zde nemusíme zadat všechny parametry, prázdné pole budou doplněné nulami.



Obrázek 18 – Polynom, zdroj: vlastní

B. 6 Informace

Poslední část obsahuje jen informace o autorovi.



Obrázek 19 – Informace, zdroj: vlastní