ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA ELEKTROENERGETIKY A EKOLOGIE

DIPLOMOVÁ PRÁCE

Porovnání výkonu numerického software

Bc. Lukáš Fuchman **2018**

Abstrakt

Předkládaná diplomová práce se zabývá výkonem numerických softwarů. V práci jsou představeny použité numerické programy a zhodnoceny z uživatelských a syntaktických aspektů. Následně jsou navrhnuty skripty pro testování náročných matematických operací a numerické programy, které jsou na skriptech otestovány a zhodnoceny z výkonového hlediska.

Klíčová slova

Numerický software, porovnání výkonu, MATLAB, GNU Octave, Scilab, FreeMat, JMathLib, pohyblivá desetinná čárka, pevná desetinná čárka, Fourierova transformace, maticové operace.

Abstract

The presented master thesis deals with the performance of numerical software. In master these work we present the used numerical software and evaluated based on user and syntax aspect. Afterwards we created scripts for testing difficult math operations and numerical programs on script tested and evaluated on performance basis.

Key words

Numerical software, performance comparing, MATLAB, GNU Octave, Scilab, FreeMat, JMathLib, Floating-point arithmetic, fixed-point arithmetic, Fourier transform, matrices operation.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

.....

podpis

V Plzni dne 22.5.2018 Bc. Lukáš Fuchman

Poděkování

Rád bych poděkoval svému vedoucímu diplomové práce doc. Ing Karlu Hruškovi, Ph.D. za přínosné rady, poskytnutí testovacích prostředků a vedení práce.

Obsah

Seznam symbolů a zkratek

CeCILL ..... CEA CNRS INRIA Logiciel Libre

GPL ..... GNU General Public License

FT ..... Fourierova transformace

Tic ..... Funkce pro měření výpočtového času

NaN ..... Not a number

OS ..... Operační systém

Úvod

Numerické softwary jsou výpočetní systémy pro vědeckotechnické numerické výpočty. Ovládání probíhá pomocí interaktivního rozhraní a skriptovacího jazyka. Oproti ostatním programovacím jazykům u numerických software odpadají některé složité zápisy a postupy a lze tak vytvářet numerické výpočty jednodušeji. Stejně tak grafické výstupy, které mají numerické programy již hotové a lze je jen jednoduše modifikovat a vykreslovat.

Využití numerických softwarů je velmi široké. Umožnují práci s maticemi, vykreslování 2D, 3D a dalších grafů, implementaci algoritmů, počítačovou simulaci, analýzu, zpracovávání, vytváření a prezentaci dat. V neposlední řadě pak i vytváření uživatelských aplikací včetně grafického rozhraní a další.

Výkon numerických programů, kterým se zabývá práce je významný z mnoha hledisek. Výkon programů má vliv na dobu výpočtu numerického problému, zejména u náročnějších výpočtů, které je nutné vyřešit v nejkratším čase. Dále je výkon důležitým aspektem při řízení systémů nebo zpracování dat. Především pokud řízení či zpracování probíhá v reálném čase, očekáváme od numerických programů nejvyšší rychlost. Rovněž má výkon vliv při používání numerického software na uživatelský komfort.

Cena numerických softwarů je různá, některé jsou šířeny zcela zdarma, jiné jsou placené a to i za velmi vysokou částkou. Placené programy by tedy měly nabízet vyšší funkčnost a především výkon. Práce porovnává výkon i z tohoto hlediska. Mezi porovnávanými programy je obsažen jeden placený program a čtyři neplacené. Porovnání tak může sloužit jak k finanční analýze, tak pro vybrání nejvhodnějšího programu pro dané výpočty.

1 Použitý numerický software

Kapitola se věnuje základním informacím a faktům o použitém numerickém softwaru, jako je použitý programovací jazyk, země původu, počet vývojářů apod. Slouží pro získání základního přehledu o použitých programech, ale i o úsilí, které bylo vynaloženo na jeho vývoj. Dané vynaložené úsilí by mělo mít zásadní vliv na pokročilost a kvalitu programů, tudíž i dopad na jejich výkon.

Numerické softwary, které bude práce hodnotit, jsou vybrané na základě rozšířenosti a oblíbenosti. Otestovat všechny dostupné numerické programy není z rozsahových i finančních důvodu možné. Z tohoto důvodu bylo vybráno následujících pět programů: MATLAB, GNU Octave, Scilab, JMathLib a FreeMat. Jejich zhodnocením bychom měli získat základní přehled a být schopni vybrat nejrychlejší program pro dané výpočty.

Základní přehled numerických softwarů je uveden v tabulce, více jich pak lze najít na Wikipedii, kde je zmíněno více než třicet numerických softwarů [1]. Přesto například program JMathLib v seznamu zmíněn není, tudíž lze přepokládat, že seznam neobsahuje veškeré numerické programy.

Název programu	*Tvůrce*	*Začátek vývoje*	*První publikovaná verze*	*Poslední stabilní verze*	*Datum stabilní verze*	*Cena*	*Licence*
MATLAB	*MathWorks*	*koncem 70. let*	*1984*	*R2017b*	*Září 2017*	*$2150 standardní / $500 vzdělávací / $49 studentská*	*Proprietární*
GNU Octave	*John W. Eaton*	*1988*	*1993*	*4.4.0*	*1.5.2018*	*Zdarma*	*Svobodná*
Scilab	*Scilab Enterprises*	*1990*	*1994*	*6.0.1*	*15.1.2018*	*Zdarma*	*CeCILL*
FreeMat	*Samit Basu*	*-*	*2004*	*4.2*	*30.6.2013*	*Zdarma*	*GPL*

Tab. 1.1 Tabulka použitých numerických softwarů

1.1 MATLAB

Nejznámější numerický software založený roku 1984. Nyní je licencovaný firmou MathWorks sídlící v Massachusetts v USA, která zaměstnává více než 4000 lidí. Obrat firmy za rok 2017 byl 900 milionů dolarů [2], přičemž uživatelů programu MATLAB jsou tři milióny po celém světě [3]. Rozhraní programu MATLAB je vyvinuto v programovacím jazyce Java a výpočetní operace v C nebo C++ [4], [5].

Produkt je šířen pod placenou licencí, kde se ceny různí podle prostředí využití, konkrétně na standardní, vzdělávací, domácí a studentské licence [6].

1.2 GNU Octave

Numerický program, jehož vývoj začal roku 1988. Nyní se na vývoji podílí i komunita vývojářů, kteří mohou pomáhat opravovat chyby, vyvíjet novou funkčnost, nebo jen odpovídat na otázky v e-mailové konferenci [7], [8]. Vývoj programu probíhá v programovacích jazycích C, C++ a Fortran [9]. Zdrojový kód je poskytnutý veřejně v repositáři pomocí verzovacího nástroje Mercurial [7].

Produkt je šířen pod volnou licencí GNU General Public License, tudíž lze program využívat zcela zdarma [9].

1.3 Scilab

Další z numerických programů, jehož vývoj započal v roce 1980 ve Francii na institutu CACSD (Computer Aided Control System Design), později v roce 1990 se pak přesunul na institut ENPC (École Nationale des Ponts et Chaussées), kde byl vytvářen šesti výzkumnými pracovníky [10]. Nyní je vyvíjen firmou Scilab Enterprises [11]. Programovací jazyky použité pro vývoj jsou C, C++, Java, XML, Scilab language a Fortran [12].

Produkt je šířen pod volnou licencí CeCILL, která je kompatibilní s GPL, tudíž lze program taktéž využívat zcela zdarma [13].

1.4 JMathLib

Vývoj numerického programu začal roku 2001 dvěma vývojáři o několik měsíců později se přidal třetí vývojář. V sestavě tří vývojářů probíhá vývoj dodnes. Program je vytvořený pouze v programovacím jazyce Java, vývojáři se záměrně vyhnuli jazykům C a C++ a program tak udržují na jediné platformě [14], [15]. Poslední stabilní vydaná verze je z roku 2009, která je i využita pro výpočty v této práci [16].

Produkt je licencován pod GNU Lesser General Public License, tudíž lze opět program využívat zcela zdarma [15].

1.5 FreeMat

První vydání stabilní verze bylo na začátku roku 2004, nicméně přesný počátek vývoje programu nikde není uveden [17]. Od té doby až dosud probíhá vývoj, nyní je program ve verzi 4.2. Na projektu se k dnešnímu datu podle verzovací a vývojové služby podílí osm osob [18]. Vytvořený je v programovacích jazycích C, C++ a Fortran [19].

Produkt je šířen pod licencí GPL, která je otevřená, a je tudíž taktéž možné používat software zcela zdarma [20].

2 Metodika návrhu testovacích algoritmů

2.1 Postup při návrhu

Každý matematický algoritmus testující výkon byl nejdříve napsán v programu MATLAB v odpovídající syntaxi. Většina numerických programů je programem MATLAB inspirována, proto kód napsaný v tomto prostředí je nejvíce univerzální a slibuje tak nejlepší přenositelnost do ostatních numerických softwarů.

Poté co byl algoritmus vyvinut tak, aby splňoval odpovídající matematické výpočty, byl přenesen do ostatních numerických softwarů, přičemž při přenosu byl upraven kód tak, aby odpovídal syntaxi daného numerického programu.

Protože každý běh testovacího algoritmu trvá rozdílnou dobu v závislosti na jeho náročnosti, samotný výkonný kód je zabalen do cyklu, kde počet jeho průchodů je definován konstantou na začátku programu. Při prvotním návrhu pro program MATLAB nebyl zatím znám výkon ostatních numerických programů, proto byla v začátku zvolena konstanta udávající počet průběhů neexaktně a spíše nižší. Při samotném testování bude pak konstanta volena tak, aby doba vykonávání algoritmu probíhala v optimálním čase ve všech programech.

Pro měření času je v algoritmech využitý časovač měřící výpočtový čas, se syntaxí `toc` pro MATLAB a jeho odpovídající alternativy pro ostatní programy [21]. Časovač je použitý pouze na samotné matematické operace, tudíž neměří čas potřebný pro deklarace a výpisy proměnných a grafů. Pro některé algoritmy je časovač použitý dvakrát a vypisuje tak dva výsledky. To z důvodu zajímavých operací, které jsou odlišné a lze je oddělit. Programy totiž vykazují zajímavé výkonové chování, které se stále týká jednoho testovacího algoritmu.

2.2 Syntaxe a specifika, rozdíly a porovnání numerických software i z uživatelského hlediska

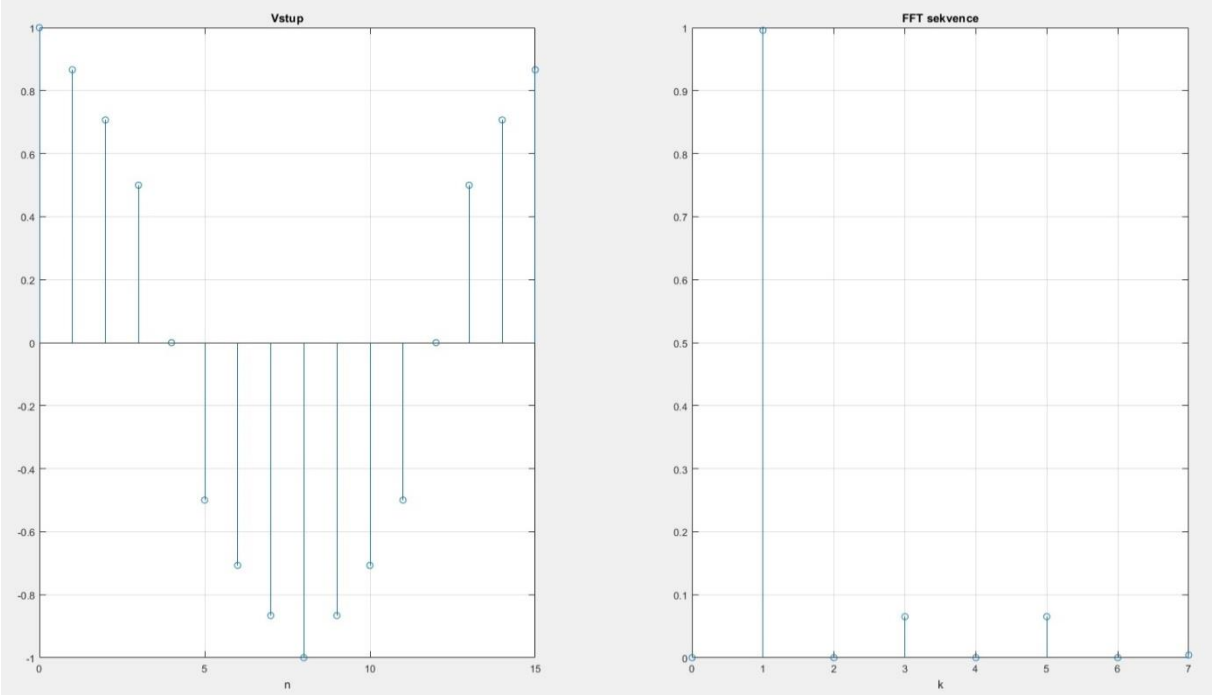
V následujícím rozdělení jsou popsány veškeré syntaktické rozdíly numerických programů, které se vyskytly při přenášení algoritmů z programu MATLAB. Syntaktických rozdílů numerické programy obsahují nepřeberné množství, nicméně práce se zabývá především těmi, které byly rozdílné při přenášení algoritmů.

Ke každému numerickému programu bude popsaná i uživatelská zkušenost, jako je jeho intuitivnost a funkcionalita. S výjimkou programu MATLAB, který je mezi uživateli velice rozšířený, a proto jej není nutné dále popisovat.

Pro každý numerický program je přiložena ukázka grafu. Pokud to bylo možné, parametry grafů v jednotlivých programech jsou totožné nebo velice podobné, především z důvodu, aby bylo možné snadno porovnat rozdíly.

2.2.1 MATLAB

Numerický program, na kterém byly navrhnuty veškeré algoritmy, je proto možné ho považovat za referenční. Ostatní programy jsou porovnávány z hlediska syntaxe opět s programem MATLAB.



Obr. 2.1 *Ukázka grafu FT z programu MATLAB*

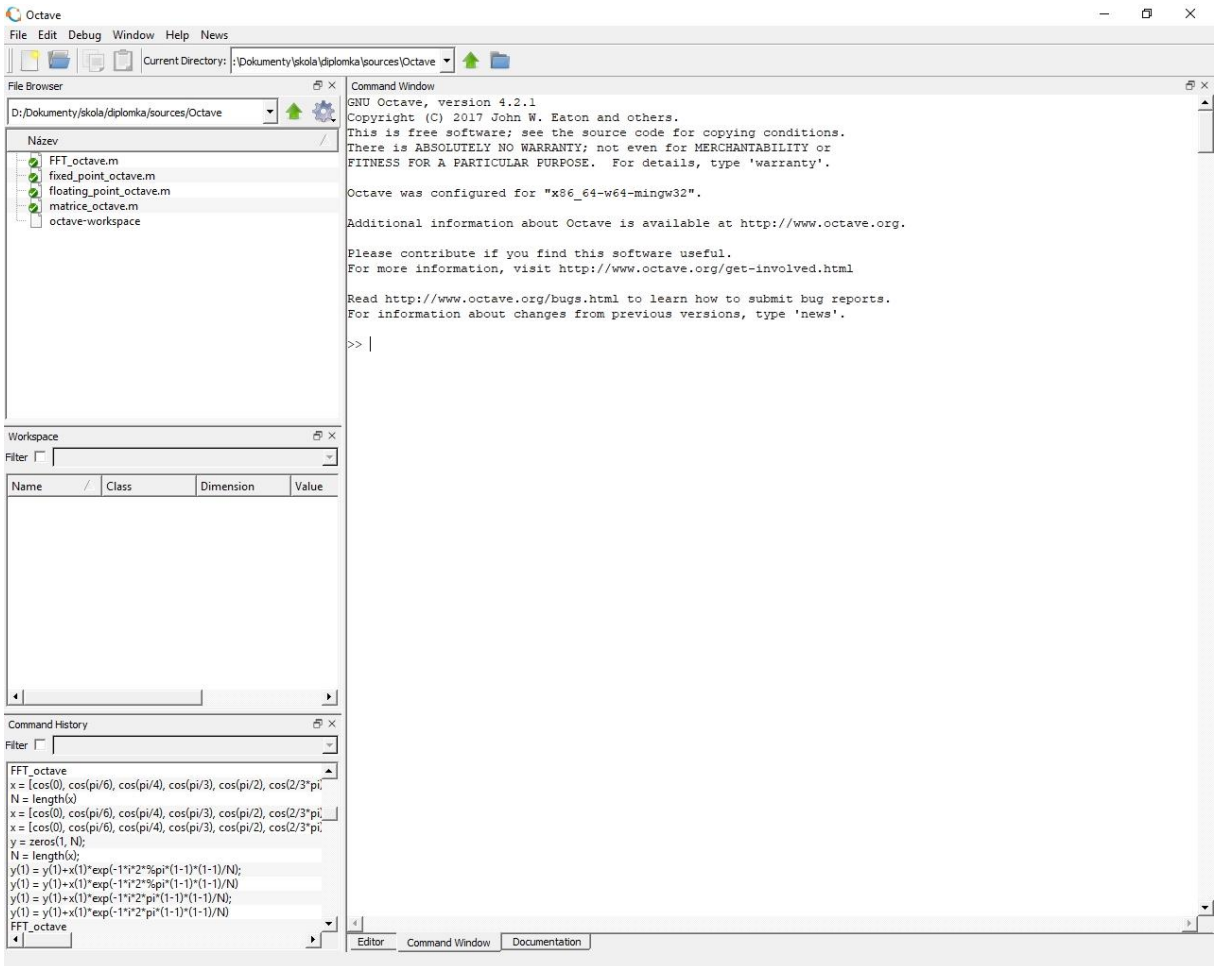
2.2.2 GNU Octave

Veškeré přenesené skripty byly spuštěny bez chyb se správnými výsledky a nemusely být nijak upraveny. Je tudíž zajištěna plná přenositelnost všech testovacích algoritmů bez potřeby změn.

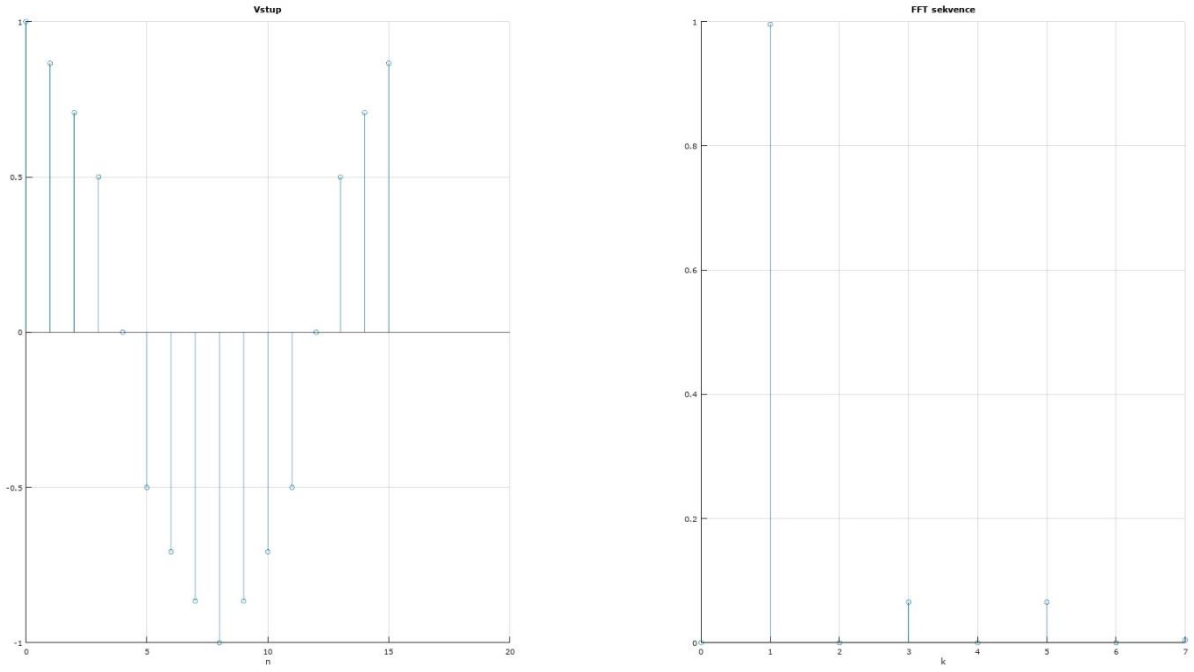
Úplná přenositelnost mezi programy však není zajištěna. Rozdíl je například mezi proměnnými `boolean` nebo příkazem `printf`, které jsou odlišené [22]. Přesto je zajištěna vysoká kompatibilita a skripty tak lze přenášet podle toho, který program je vypočte rychleji.

Uživatelské rozhraní programu je dobře strukturované, obsahuje veškeré užitečné prostředky jako seznam deklarovaných proměnných, editor, historie příkazů nebo adresářová struktura. Samotný editor kódu je na velmi vysoké úrovni. Přehledně zvýrazňuje syntaxi, páruje těla bloku a poskytuje jejich skrývání. Dále umožnuje do skriptu přidat breakpointy. Tudíž lze program krokovat, což je praktické pro orientaci především při vývoji složitějších programů.

Prostředí programu GNU Octave na uživatele působí přehledně, ovládání je veskrze intuitivní a v programu se uživatel rychle zorientuje i při přechodu z jiného numerického programu.



Obr. 2.2 *Ukázka rozhraní programu GNU Octave*



Obr. 2.3 *Ukázka grafu FT z programu GNU Octave*

2.2.3 Scilab

Ze všech testovaných programů má program Scilab nejvíce odlišnou syntaxi. Žádný z přenesených skriptů se nepodařilo beze změny spustit.

Rozdíly v syntaxi:

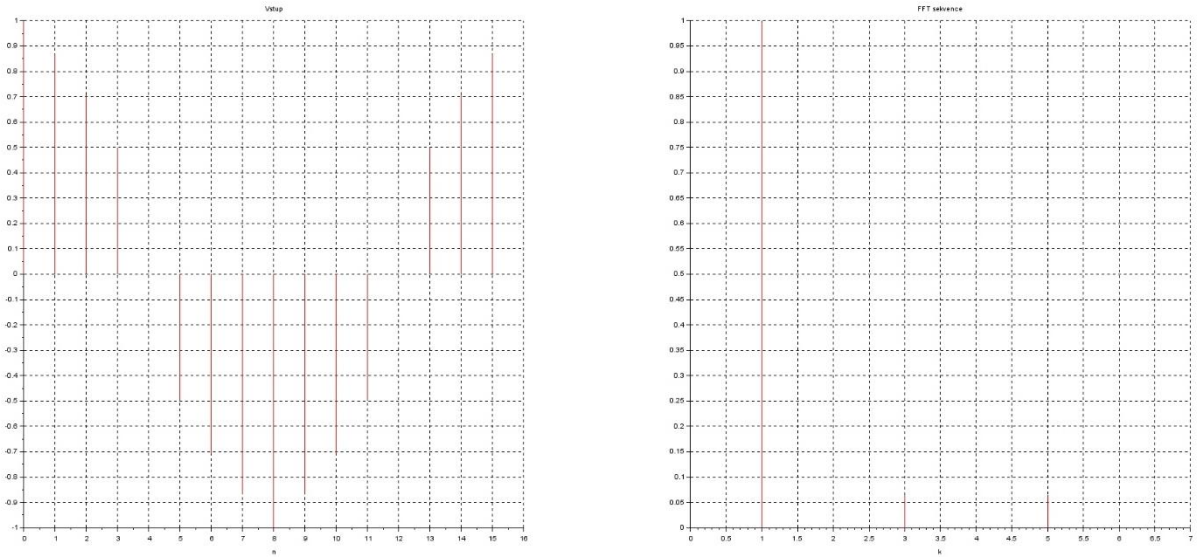
- Komentáře se zapisují pomocí dvojitých lomítek `//`, stejně jako ve vyšších programovacích jazycích, např. C++. V ostatních programech se zapisují pomocí znaku procenta.
- Za časovač `tic` vyžadováno psát kulaté závorky, stejně jako při volání metod a funkcí, tudíž `tic()` a `toc()` [23].
- Transpozice matice se zapisuje jako `B'`, oproti programu MATLAB, který podporuje dva formáty zápisu: `transpose(B)` nebo `B = B.'` [24], [25], kde proměnná `B` je transponovaná matice [24], [25].
- Před konstantou `pi` nebo imaginárním číslem `i` je nutné vložit znak procenta. Zapisujeme je tudíž jako `%pi` nebo `%i`. Pokud není procento uvedeno, program Scilab pracuje s konstantou jako s běžnou, uživatelsky deklarovanou proměnnou. Je tudíž potřeba dát pozor při přenosu skriptů, protože není vypsána žádná chyba v překladu, ale výsledek výpočtu může být chybný.
- Pro vykreslení diskrétní sekvence se používá funkce `plot2d3`, místo funkce `stem`,

kterou používá program MATLAB [26], [27].

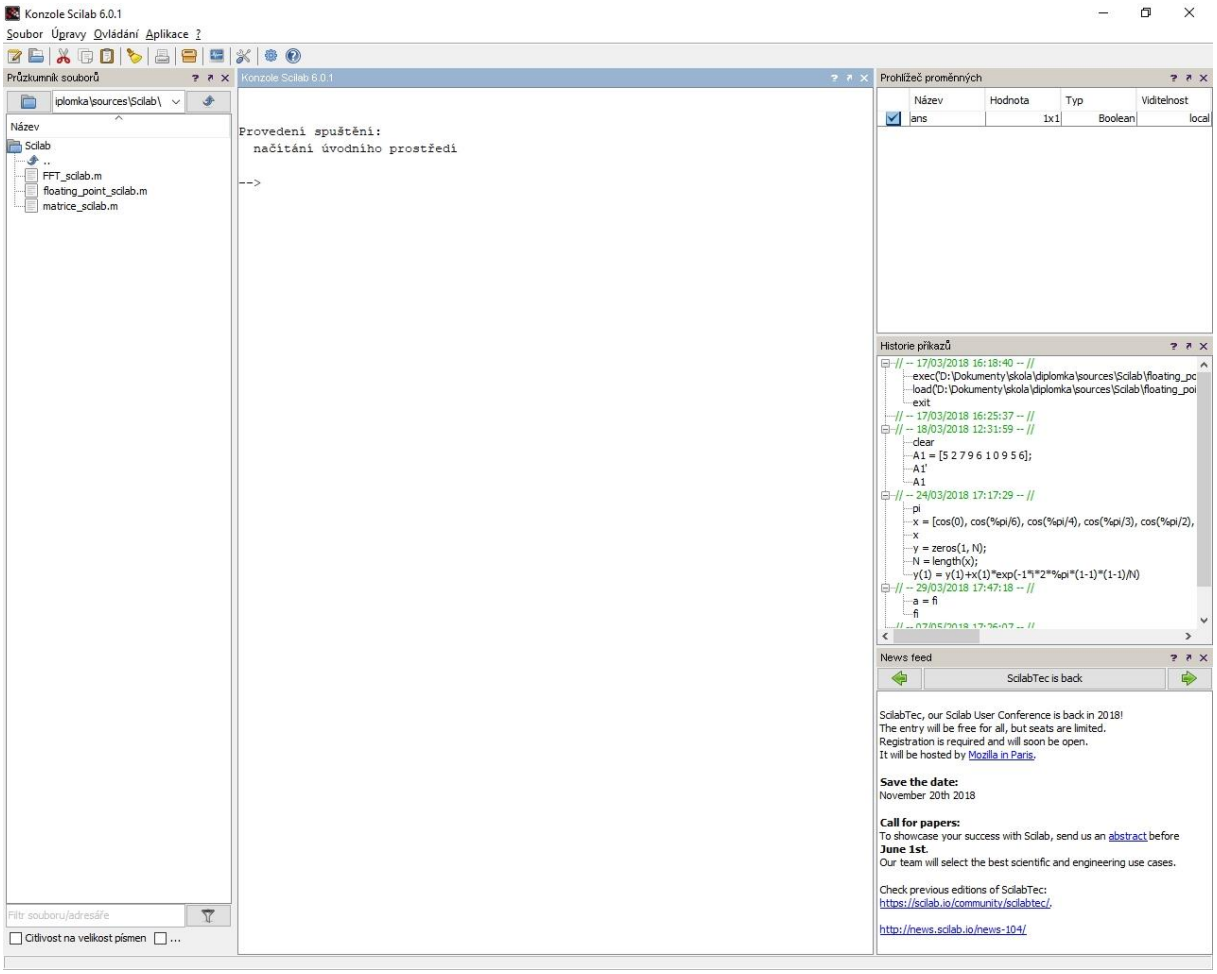
- Pro vykreslení mřížky do grafu se používá příkaz `xgrid`, místo příkazu `grid`, který používají ostatní numerické programy [29].
- Barvy jsou definovány odlišně. Pro rozlišení os v grafu byla použita funkce `color()`, kde jejím vstupem je řetězec s názvem barvy, např. `color('red')`.

Uživatelské rozhraní programu Scilab je velmi dobře strukturované, taktéž nabízí veškeré užitečné a prakticky totožné prostředky jako program GNU Octave. Pro editaci skriptů program využívá vlastní vestavěný textový editor nazvaný SciNotes. Editor poskytuje výbornou přehlednost kódu, zvýrazňování syntaxe a další pokročilou funkčnost. Jako nevýhodu lze považovat neumožnění grafického zadávání breakpointů. Debugging se provádí pomocí příkazů přímo v kódu [30].

Prostředí programu Scilab je na velice dobré úrovni. Přehledné, intuitivní a snadné na pochopení. Z testovaných numerických softwarů jej hodnotím jako nejlépe zpracované a nejvíce pokročilé, rozdílnou syntaxi lze vyřešit například převodníkem, který Scilab přímo obsahuje. V porovnání s programem MATLAB neumožňuje takovou funkčnost, ale v přehlednosti a intuitivnosti působí lépe.



Obr. 2.4 *Ukázka grafu FT z programu Scilab*



Obr. 2.5 *Ukázka rozhraní programu Scilab*

2.2.4 JMathLib

Umožňuje spouštět skripty přímo z formátu .m, tak jak je uloží program MATLAB, čehož při spouštění testovacích algoritmů bude využito. Přenositelnost skriptů se jeví jako průměrná, žádný ze skriptů nešel spustit přímo, ale potřebné změny v syntaxi jsou malé.

Rozdíly v syntaxi:

- Proměnné `i` a `j` nelze použít jako klasické uživatelsky definované proměnné. Je nutné je pojmenovat například velkým písmenem `I` a `J`. Proměnné jsou použité v cyklech a obvykle se tak i využívají. Protože program nevypisuje, pozici chyb ve skriptu, jejich nalezení bylo velice obtížné. Nefunkčnost je pravděpodobně způsobená konfliktem s imaginárními čísly.
- Řídící struktury `if`, `while` a další musí mít podmínku v závorkách. Například `if (k ~= Asize)`.
- Příkaz pro vymazání deklarovaných proměnných `clear` je nutno zapisovat ve formátu

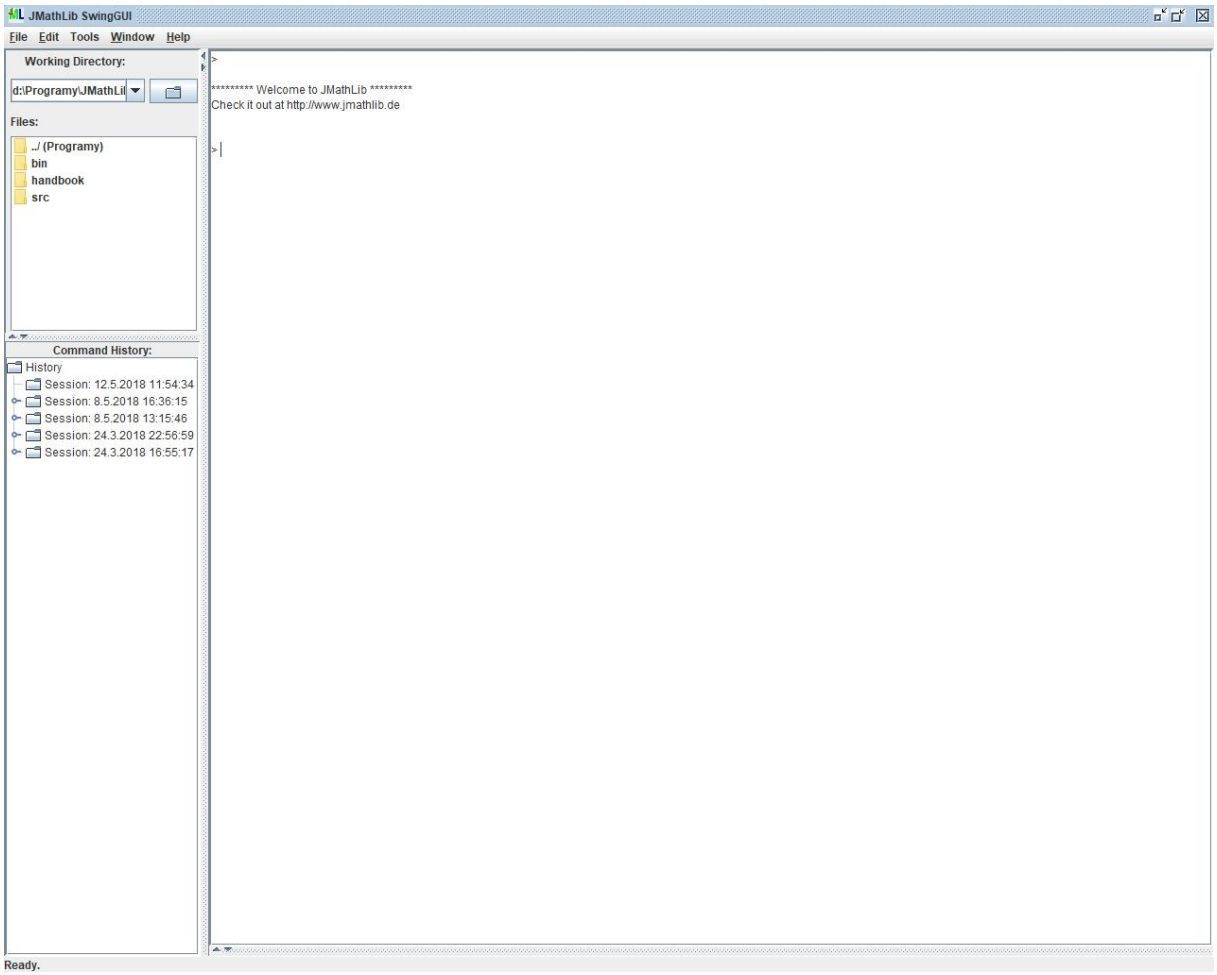
```
clear("variables").
```

- Nastavení grafu probíhá pomocí parametrů, které jsou zcela odlišné od ostatních programů. Přesné použití nastavení je popsáno v programové dokumentaci [31].

Program lze spustit ve dvou režimech. První je JMathLib GUI, který obsahuje konzoli a je ve vizuálu Windows. V něm lze provádět jednotlivé příkazy, avšak neumožňuje spouštět skripty. Druhý je JMathLib SwingGUI ve vizuálu Java Swing [32]. Rozhraní obsahuje základní prvky, pracovní složku, historii příkazů a konzoli. Editor skriptů se nepodařilo spustit nebo vůbec není obsažen, skripty proto byly upravovány v externím editoru Notepad++ [33]. Složka s upravenými skripty musela být přidána do adresářové cesty programu, poté mohly být skripty v konzoli spuštěny z důvodu nefunkční sekce s pracovním adresářem. V konzolovém okně je možné vepisovat, mazat i editovat text i mimo sekci pro zadávaní, takže obsluha programu je matoucí. Naopak značná výhoda programu spočívá v jeho možnosti embedování do externí aplikace. Je tudíž možné bez velkého úsilí ve své aplikaci využít hotový numerický nástroj, umožňující spouštět složité výpočty.

Prostředí programu JMathLib je značně nedokončené. Nefungující agendy, prázdné položky v menu, editovatelná konzole apod. Zorientování a spuštění skriptů je zdlouhavé a neintuitivní. Po vypnutí programu nejsou zapamatovány položky z předchozího běhu, což z programu společně s ostatními nedostatky dělá těžko použitelný program pro náročnější projekty. Nutno ovšem dodat, že program ještě není ve verzi 1.0 a necelých deset let nebyla vydána nová verze, z čehož lze usoudit, že už neprobíhá jeho aktivní vývoj.

Ukázka grafu není přiložena, protože v prostředí JMathLib SwingGUI je chyba, která okno s grafem sice vytvoří, ale graf není vykreslen.



Obr. 2.6 *Ukázka rozhraní programu JMathLib*

2.2.5 FreeMat

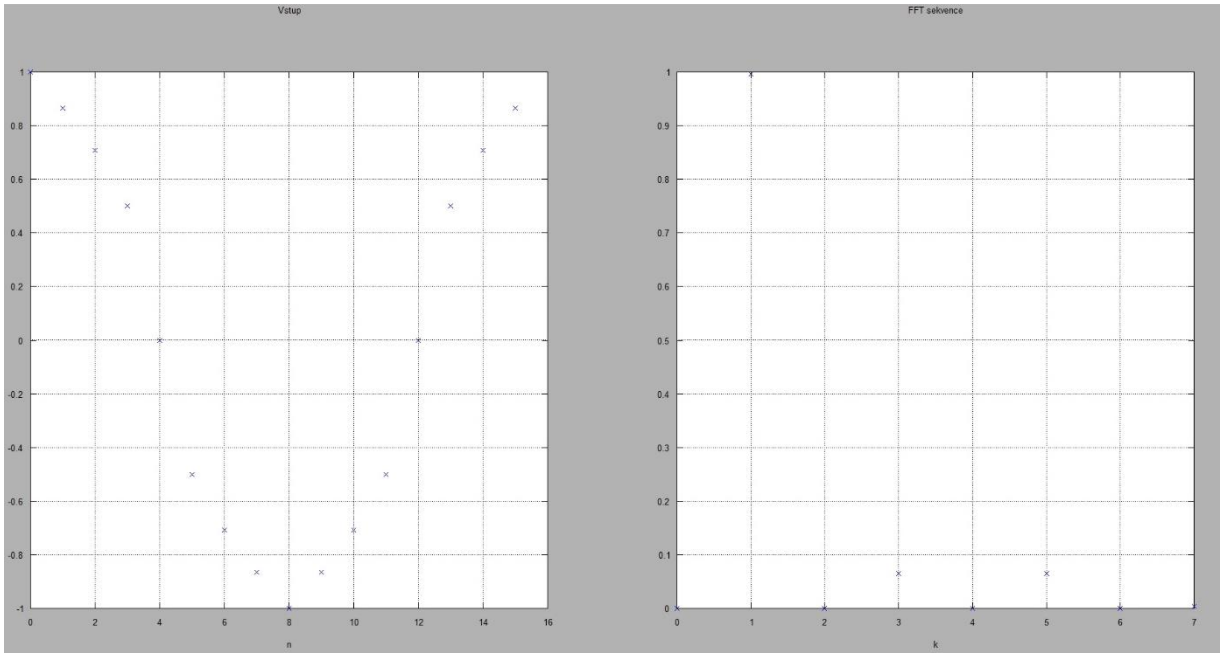
Poskytuje vysokou kompatibilitu mezi přenášenými skripty. Většina přenesených skriptů funguje bez potřeby úpravy.

Rozdíly v syntaxi:

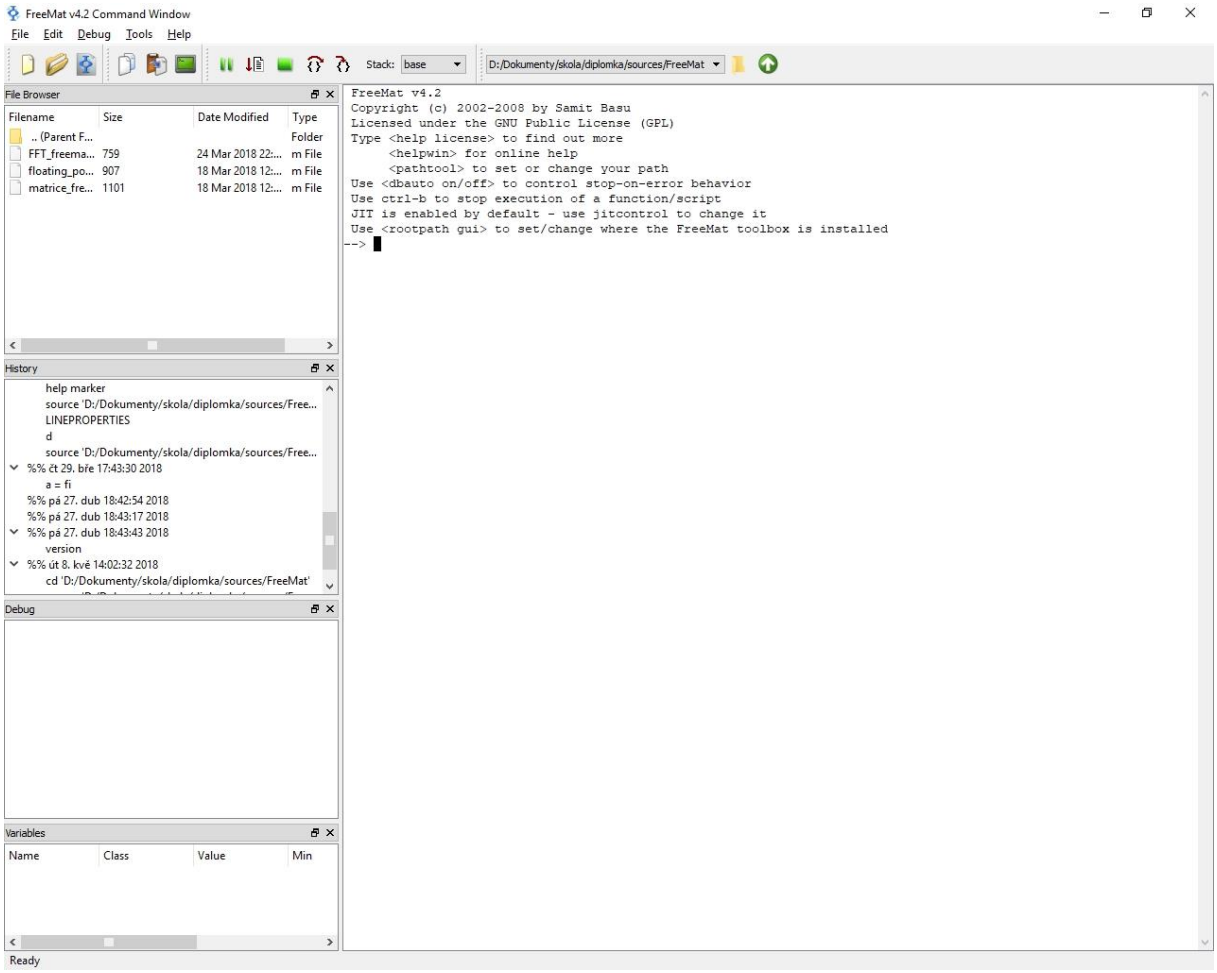
- Příkaz `stem` pro vykreslení diskrétní sekvence není programem podporován a nepodařilo se dohledat náhradu. V dokumentaci programu v sekci zabývající se výpočty FT je vizualizována pomocí klasického grafu. Pro vizualizaci skriptu je použit totožný přístup v přizpůsobeném nastavení. Body jsou změněny na znaky x a spojovací čára vypnuta.

Uživatelské rozhraní programu FreeMat obsahuje veškeré standardní nástroje analogicky jako program Scilab. Umožňuje vkládat breakpointy do skriptu a debutovat. Taktéž obsahuje pokročilý editor skriptů a další.

Prostředí programu FreeMat je intuitivní, snadné pro zorientování a jeho rozložení se drží standardů numerických programů. Pro své přehledné prostředí a vysokou kompatibilitu skriptů je velice vhodný jako alternativa.



Obr. 2.7 *Ukázka grafu FT z programu FreeMat*



Obr. 2.8 *Ukázka rozhraní programu FreeMat*

3 Návrh algoritmů

Při návrhu algoritmů byl držen jednotný koncept skriptování tak, aby algoritmy byly přehledné, měly stejný styl a konzistentní názvosloví. Například podobné proměnné nebo proměnné o totožném významu pojmenované stejně. Při přenosu skriptu na jiný numerický program a při změně syntaxe byla zachována posloupnost příkazů. Skripty tudíž vykazují vzájemnou podobnost, která přispívá k přehlednosti a rychlé orientaci.

3.1 Struktura algoritmu

První příkaz každého algoritmu je vymazání deklarovaných proměnných z předchozího běhu programu příkazem `clear`. Vyčištění paměti je důležité kvůli možnému zkreslení výsledků výkonu.

Prvním blokem každého skriptu jsou deklarace proměnných a jejich konstant. Zde je obsažena konstanta `loops` udávající počet běhů výpočtů. Dále se provádí alokování vektorů a jejich plnění nulovými hodnotami a ostatních proměnných potřebných pro výpočet.

Druhý blok obsahuje výpočet daného matematického problému. Probíhá v cyklu, který je obalen časovačem měřícím čas celého průchodu. Časovač je jen nad tímto blokem a měří pouze čas numerického problému, zbytek kódu (výpisy, grafy a deklarace) není relevantní.

Třetí a poslední blok obsahuje informační výpisy. Nejdůležitější je čas výpočtu, který udává výkon programu. Při vývoji skriptů byly vypisovány výsledky výpočtů hlavně pro kontrolu správnosti skriptů. Nyní se vykreslují jen snadno rozlišitelné výsledky, například grafy.

3.2 Zásady návrhu

V algoritmech nejsou využité žádné pokročilejší funkce numerických programů, jakou je například funkce `fft` pro výpočet rychlé Fourierovy transformace a podobné [34]. Veškeré operace jsou zapsány pomocí základních funkcí a příkazů, které podporují všechny numerické softwary. V případě využití pokročilejší funkce, která v sobě obsahuje pokročilejší algoritmy, by nebyl testován samotný výkon numerických programů, ale spíše pokročilost těchto algoritmů.

Níže uvedené algoritmy jsou v syntaxi pro program MATLAB z důvodu největší kompatibility a tudíž i názornosti. Algoritmy pro ostatní numerické programy jsou uvedené v příloze.

3.3 Určení počtu cyklů algoritmu

Pro určení počtu cyklů byl každý algoritmus nejdříve spuštěn jednou a byl změřen čas běhu. Poté byl čas vynásoben číslem tak, aby se doba pohybovala řádově v sekundách, u pomalejších programů v desítkách sekund. Číslo pak udává počet cyklů. Pokud některý z numerických softwarů vykazoval nízký výkon, počet cyklů se snížil a algoritmy se opět spustily na všech numerických programech. Tímto, spíše experimentálním způsobem byl nalezen optimální počet průchodů cyklů.

3.4 Výpočty s pohyblivou desetinnou čárkou

Inicializační blok deklaruje dvě proměnné `x` a `y` a nastavuje jejich přesnost desetinné čárky na dvojitou. Dvojitá přesnost proměnných je v programu MATLAB výchozí volba, přesnost je tedy nastavena redundantně pro předejití chybných výsledků [35].

```
clear;

x = 0;
y = 0;

double(x);
double(y);

loops = 100;
```

Na proměnné `x` a `y` jsou aplikovány operace pracující s desetinnou čárkou. Jsou to goniometrické funkce, sinus, cosinus a jejich hyperbolické varianty, mocnina, odmocnina, exponenciála, logaritmus, Pythagorova věta – analogická k funkci `hypot` a převrácená hodnota odmocniny. Veškeré aplikované operace jsou doporučené matematické operace s desetinnou čárkou podle standardu IEEE 754 [36].

V cyklu je do proměnné `y` vždy přiřazena hodnota, která odpovídá počtu aktuálních průchodů cyklu. Proměnné `x` není na začátku cyklu přiřazována žádná hodnota, zůstává uložená z předchozího běhu cyklu. Tato přiřazení zajišťují, že při každém průběhu cyklu jsou aplikovány matematické operace na jinou hodnotu, což zvyšuje složitost a komplexnost testu.

```
tic
for i = 1:loops;
    y = i;

    x = sin(y);
    y = cos(x);

    x = cosh(y);
    y = sinh(x);

    x = exp(x);
    y = log(y);

    x = x^abs(y);

    y = abs(y)^(1/abs(x));
    x = sqrt(x^2+y^2);

    y = sqrt(1/abs(x));
end
toc
```

Blok pro výpis výsledku u algoritmu není potřeba. Je zde jen pro kontrolu, zdali se hodnoty po provedení výpočtu nedostaly do nežádoucích rozsahů. Blok sloužil hlavně při vývoji algoritmu.

```
format long;
disp(x);
disp(y);
```

3.4.1 IEEE 754

Norma IEEE 754 je technický standard pro výpočty v pohyblivé desetinné čárce, které jsou implementovány v matematických koprocesorech nebo ve specializovaných číslicových obvodech [37].

Standard byl založen roku 1985 Institutem pro elektrotechnické a elektronické inženýrství (IEEE). Současná verze pochází z roku 2008 pod celým názvem IEEE 754-2008, která rozšiřuje původní standard IEEE 754-1985.

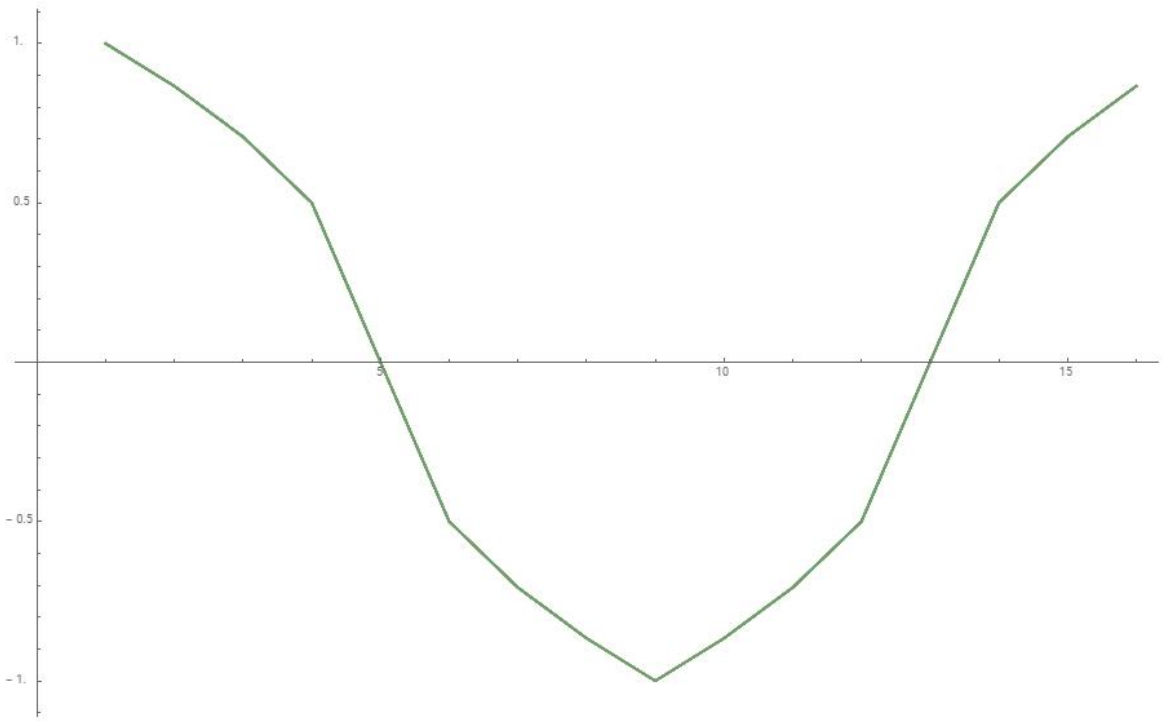
Norma definuje pět základních formátů pro uložení numerických hodnot, ale i pravidla pro implementaci aritmetických operací na tyto hodnoty. Základní formáty jsou pojmenovány podle jejich numerické základny a počtu bitů, například binary64 – dvojitá přesnost.

V normě jsou dále obsažena pravidla konverze mezi celočíselnými formáty a formáty s pohyblivou desetinnou čárkou, dále konverze mezi formáty s pohyblivou desetinnou čárkou a mezi základním formátem s pohyblivou desetinnou čárkou na řetězec číslic. Nakonec pak práci s hodnotami NaN a výjimkami [38].

3.5 Fourierova transformace

Inicializační blok deklaruje proměnnou x jako vektor, který obsahuje vstupní hodnoty pro FT. Pro porovnání výkonu numerického software nemají vstupní hodnoty vliv, byly proto zvoleny.

Vstupní hodnoty pro FT jsou vzorky z funkce cos(x) v intervalu <0, 2pi>. Vektor x obsahuje šestnáct hodnot, které leží na funkci. Na obrázku 3.1 je vektor pro ukázku vykreslen.



Obr. 3.1 *Vykreslený vektor x, který je vstupem FT*

Dále je deklarována proměnná `N`, která obsahuje délku vektoru potřebného pro následující výpočty. Dále je vektor `y` deklarován na nuly o stejné délce jako vektor `x`. Vektor `y` slouží pro pomocné výpočty.

```
clear;

loops = 100;

x = [cos(0), cos(pi/6), cos(pi/4), cos(pi/3),
    cos(pi/2), cos(2/3*pi), cos(3/4*pi), cos(5/6*pi),
    cos(pi), cos(7/6*pi), cos(5/4*pi),
    cos(4/3*pi), cos(3/2*pi), cos(5/3*pi),
    cos(7/4*pi), cos(11/6*pi)];

N = length(x);

y = zeros(1, N);
```

V bloku s výpočtem je naprogramovaná Fourierova transformace podle vzorce

$$X_k = \sum_{k=0}^{N-1} x_k e^{\frac{-2\pi kn}{N}} \qquad (3.1),$$

kde je pro jeho výpočty zapotřebí dvou vnořených cyklů, jejichž pomocné proměnné jsou `k` a `n` korespondující s názvy ve vzorci [39], [40]. Ze vzorce je patrné, že jeden cyklus je zapotřebí pro výpočet sumace a druhý pak pro `N`. Na řádku uvnitř cyklu s pomocnou proměnnou cyklu `n` je naprogramován vzorec pro `y(k)`.

Proměnná `mag` znamená v překladu z angličtiny magnitude, v češtině je to maximální amplituda výstupního signálu.

Ve vnějším cyklu opět probíhá opakování stejného výpočtu pro vytížení numerického programu. Význam přiřazení `y(k) = 0` je pro opakované výpočty, kdy se z pomocného vektoru y mažou vypočítané hodnoty z předchozího cyklu.

```
tic
for i = 100:loops;
    for k = 1:N
        y(k) = 0;
```

```
        for n = 1:N
            y(k) = y(k)+x(n)*exp(-1i*2*pi*(k-1)*(n-1)/N);
        end
    end
    mag = 2*abs(y)/N;
end
toc
```

Výstupem algoritmu jsou grafy vstupní a výstupní funkce FT vykreslené v diskrétní podobě. Na blok s informačními výpisy není nastaven časovač, protože se jedná o grafický výstup ve kterém nejsou obsaženy výpočty. Grafy jsou vykresleny vedle sebe pomocí funkce `subplot` a pojmenovány.

Ukázky grafů pro každý numerický program jsou uvedeny v kapitole 2 a všechny odpovídají zmíněným výpočtům v této kapitole.

```
t = 0:N-1;
subplot(1, 2, 1);
stem(t, x);
xlabel('n');
title('Vstup');
grid on;

subplot(1,2,2);
stem(t(1:N/2), mag(1:N/2));
xlabel('k');
title('FT sekvence');
grid on;
```

3.6 Maticové operace

Inicializační blok obsahuje proměnnou `size` jako konstantu, která představuje velikost vytvářených matic a určuje tak zásadním způsobem náročnost algoritmu. Při deklaraci matic, která bude popsaná níže, se matice násobí referenčním vektorem, takže skutečná velikost matice je ještě větší, v závislosti na délce vektoru. Konstanta tak představuje majoritní parametr velikosti matic. Pro optimální výsledky výpočtu byla zvolena na hodnotu 100.

Vektory `A1` a `A2` jsou označovány jako referenční. Pomocí nich se vytváří pseudonáhodné matice, přičemž jejich délkou lze regulovat velikost matic. Referenční vektory jsou naplněné čísly, která jsou zcela náhodná. Generování čísel neprobíhá softwarově, ale čísla byla zvolena předem kvůli konzistenci výpočtu. Při vygenerování nových náhodných čísel při spuštění

skriptů by se mohla měnit náročnost výpočtu, tudíž i čas. Pro předpoklad správného výpočtu je nutná totožná velikost referenčních vektorů `A1` a `A2`.

Proměnná `Asize` počítá délku referenčního vektoru `A1`. Proměnná `matriceSize` počítá skutečnou velikost matice. Obě proměnné jsou potřebné pro výpočtový blok. Proměnné `rotA1` a `rotA2` jsou pomocnými vektory pro inicializace matic. Inicializační blok nastavuje jejich hodnoty na nuly.

Proměnné `B1` a `B2` jsou samotné matice, na kterých se provádějí výpočty. Předpoklad pro korektní výpočty je čtvercový charakter matic. Jejich horizontální a vertikální velikost je nastavena na dříve vypočítanou velikost `matriceSize`. Se zvolenými parametry mají matice velikost 1000 na 1000 položek, což je odpovídající velikost pro výpočty.

```
clear;

size = 100;

loops = 10;

A1 = [5 2 7 9 6 1 0 9 5 6];
A2 = [3 3 7 6 0 8 0 1 2 5];

Asize = length(A1);
matriceSize = Asize * size;

rotA1 = zeros(1, Asize);
rotA2 = zeros(1, Asize);

B1 = zeros(matriceSize, matriceSize);
B2 = zeros(matriceSize, matriceSize);
```

Výpočtový blok byl rozdělen na dvě části. První sestavuje pseudonáhodné matice, druhý provádí nad maticemi matematické operace. Protože jsou bloky odlišné, co se operací týče, měření výpočtového času je odděleno. Proto jsou v maticovém skriptu získávány informace o rychlosti výpočtu ze dvou odlišných problematik.

Následující blok kódu vytváří matice pomocí referenčních vektorů, ze kterých postupně skládá matice vkládáním referenčních vektorů vedle sebe. Pro matici `B1` je použit referenční

vektor `A1` a pro matici `B2` pak `A2`. První cyklus `while` probíhá tak dlouho, dokud nedosáhne skutečné velikosti matic. Dva vnořené cykly `for` prochází jednotlivé položky po řádcích a sloupcích. Referenční vektor o velikosti deseti položek s náhodnými hodnotami je vkládán vedle sebe, dokud není dosaženo maximálního počtu sloupců. Pro konstantu `size = 100` je vložen stokrát, vznikne tisíc sloupců. Poté je referenční vektor skládán pod sebe. Kvůli předpokladu čtvercové matice a výšce vektoru jedna je pod sebe vložen tisíckrát.

Aby bylo dosaženo rozmanitosti matic, každý nový řádek je posunut o jednu pozici do leva. Pro dočasné uložení referenčních vektorů jsou využity vektory `rotA1` a `rotA2`, které jsou rotovány v cyklu `k` a po doběhnutí cyklu přiřazeny do referenčních vektorů.

Níže je na ukázku vypsána matice B1 o velikosti konstanty `size = 1` vytvořená výše ukázaným algoritmem a referenčním vektorem.

```
 5     2     7     9     6     1     0     9     5     6
 2     7     9     6     1     0     9     5     6     5
 7     9     6     1     0     9     5     6     5     2
 9     6     1     0     9     5     6     5     2     7
 6     1     0     9     5     6     5     2     7     9
 1     0     9     5     6     5     2     7     9     6
 0     9     5     6     5     2     7     9     6     1
 9     5     6     5     2     7     9     6     1     0
 5     6     5     2     7     9     6     1     0     9
 6     5     2     7     9     6     1     0     9     5
```

```
tic
i = 1;
j = 1;
while i <= matriceSize
    for j = 1:matriceSize
        for k = 1:Asize
            B1(j, i+k-1) = A1(k);
            B2(j, i+k-1) = A2(k);

            if k ~= Asize
                incIndex = k + 1;
            else
                incIndex = 1;
            end

            rotA1(k) = A1(incIndex);
            rotA2(k) = A2(incIndex);
        end
        A1 = rotA1;
        A2 = rotA2;
```

```
    end
    i = i + Asize;
end
toc
```

Druhý měřený blok kódu vykonává operace nad maticemi. Matice se sčítají, násobí, transponují a vypočítává se jejich determinant. To celé opět ve smyčce pro maximální vytížení numerického softwaru. Vzhledem k charakteru matic vychází determinant matice `B` nulový.

```
tic
for i = 1:loops
    B = B1 + B2;
    B = B1 * B;
    B = transpose(B);
    d = det(B);
end
toc
```

3.7 Výpočty s pevnou desetinnou čárkou

Veškeré proměnné v numerických programech jsou automaticky deklarované s pohyblivou desetinnou čárkou. Pro výpočty s pevnou desetinnou čárkou byla pro program MATLAB použita funkce `fi`, která vytváří objekt, v našem případě číslo, s pevnou desetinnou čárkou [41].

V bloku deklarací je do proměnné `x` nastavena pevná desetinná čárka.

```
clear;

loops = 10000;

x = fi(1);
```

Výpočtový blok vytváří novou proměnou `y`, do které se přiřazuje počet aktuálního průchodu cyklu. Pokud je do proměnné s pevnou desetinnou čárkou přiřazena hodnota s proměnnou desetinnou čárkou, proměnná s pevnou desetinnou čárkou je změněna na pohyblivou. Proto musí být hodnota z proměnné `i` do proměnné `y` přiřazena pomocí funkce `fi`. Při každém výpočtu v cyklu je hodnota v proměnné `x` ponechána a dále jsou prováděny výpočty. Při každém průběhu jsou tudíž v proměnných jiné hodnoty, což zvyšuje variabilitu výpočtu.

Následně jsou na proměnné aplikované základní matematické operace, sčítání, odčítání, dělení, násobení. Poslední operace násobí číslo konstantou s desetinným místem. To z důvodu zavedení do výpočtu, kromě celých čísel, i reálná čísla. Typy použitých matematických operací, které pracují s pevnou desetinnou čárkou, byly převzaty z dokumentace pro program MATLAB [42], [43].

```
tic
for i = 1:loops;
    y = fi(i);

    x = x + y;
    x = x - y;

    x = x / y;
    x = x * y;

    x = y * 0.1415;
end
toc
```

Blok vypisující výsledek je zde pouze pro kontrolu, zdali se hodnoty po provedení výpočtu nedostaly do nežádoucích rozsahů. Blok tak sloužil hlavně při vývoji algoritmu.

```
format long;
disp(x);
disp(y);
```

3.7.1 Podpora v numerických softwarech

Při přenosu algoritmu na ostatní numerické softwary bylo zjištěno, že funkce `fi` není podporována a neexistuje pro ni alternativa, respektive ostatní programy pevnou desetinnou čárku nepodporují. Proto není možné spustit algoritmus mimo program MATLAB.

Podle dokumentu A Fixed-Point Type for Octave z dubna roku 2006 pocházejícího z Motorola laboratoří lze v numerickém softwaru GNU Octave pracovat s pevnou desetinnou čárkou pomocí toolboxu [44]. Autoři zmiňují, že toolbox byl šířen pod licencí GNU Public Licence, čímž se stal zdarma použitelný. Dále rozebírají jeho využití a uvádí příklady. Nicméně v seznamu dostupných toolboxů pro program GNU Octave již není k nalezení a nelze jej tudíž

do programu přidat [45]. Při zevrubném průzkumu proč tomu tak je, bylo zjištěno, že toolbox pro nové verze programu již není kompilován, a proto není dostupný [46].

Použití toolboxu je podle dokumentu relativně jednoduché. Pro vytvoření nové proměnné s pevnou desetinnou čárkou stačilo použít funkci `fixed`, která umožnovala vytvářet skalární, komplexní a maticově komplexní hodnoty.

Pro ostatní numerické softwary nebyla v dokumentacích nalezena jakákoliv zmínka o pevné desetinné čárce či podpoře funkce `fi` nebo `fixed`. Lze proto předpokládat, že operaci nepodporují. Z tohoto důvodu nebude výkon pevné desetinné čárky v numerických softwarech porovnáván.

4 Technické prostředky

Kapitola obsahuje veškeré technické prostředky a specifikace, které byly použity při testování výkonu algoritmů. Slouží především pro orientaci a kategorizování výsledků vzhledem k verzím programů a operačních systému. Dále pak pro správnou interpretaci výsledků, případně jejich reprodukci.

4.1 Hardware

Výkon byl testován na přenosném počítači – laptopu značky DELL. Laptop obsahoval 4 GB operační paměti a procesor od firmy Intel s označením Intel® Core™ i5-8250U. Tento procesor z řady osmé generace pod názvem Kaby Lake R má čtyři fyzická jádra o frekvenci 1,6 GHz, přičemž maximální turbo frekvenci o 3,6 GHz. Dále obsahuje osm vláken a 6 MB cache [47]. Vydán byl ve třetím kvartálu roku 2017.

4.2 Systémy

Testovací hardware obsahoval operační systémy, Windows a Linux o verzích:

- Windows 10 pro 10.0.16299 build 16299
- Linux Debian Buster

4.3 Verze programů

Numerické programy byly pro oba systémy nainstalovány ve stejné verzi, aby bylo možné porovnat i vliv systému na rychlosti výpočtů. Jediný rozdíl byl ve verzi běhového prostředí Javy, která by ovšem na výkon neměla mít vliv z důvodu jejího použití převážně na grafické rozhraní programů.

Verze nainstalovaných numerických programů:

- MATLAB R2017b
- GNU Octave 4.2.2
- Scilab 6.0.0
- JMathLib 0.9.4
- FreeMat 4.2

Verze programu Java:

- Java SE Development Kit 10.0.1+10 pro Windows
- Java SE Development Kit 9.04+12-debian-4 pro Linux

5 Výsledky výpočtů

5.1 Principy při testování

Veškeré výpočty byly spouštěny duálně na dvou operačních systémech, konkrétně Windows a Linux. Algoritmy při přenosu mezi systémy zůstaly totožné a programy byly instalovány v totožné verzi. Lze tudíž porovnat vliv operačního systému na výkon.

Při určování počtu cyklů jednotlivých výpočtů z důvodu vysokého výkonu programu MATLAB bylo dosaženo relativně vysokých hodnot. Jednotlivé výpočty jsou tedy velice náročné.

Během testování výkonu pro numerický program JMathLib se ukázalo, že je velice pomalý. Jeden z důvodů bylo jeho logování do souboru a příkazové řádky. Avšak i přes vypnutí logování pomocí příkazu `debug(0)`, bylo zaznamenáno jeho zrychlení jen okolo dvaceti procent. Počty cyklů pro tento program byly tedy zmenšeny o několik řádů tak, aby výsledky řádově odpovídaly ostatním numerickým programům a až poté byla zaznamenána jeho rychlost.

5.2 Výsledky výpočtů algoritmů na numerických softwarech

5.2.1 Výpočty s pohyblivou desetinnou čárkou

Pro výpočty s pohyblivou desetinnou čárkou bylo experimentálně určeno 1 000 000 cyklů. Pro program JMathLib pak nastaveno pouze 1 000 cyklů. Porovnání trvání výpočtů ukazuje následující tabulka.

Numerický software	*Doba výpočtu pro Linux [s]*	*Doba výpočtu pro Windows [s]*
MATLAB	*0,452*	*0,579*
GNU Octave	*8,358*	*46,077*
Scilab	*8,026*	*13,125*
FreeMat	*48,702*	*151,673*
JMathLib	*30,700*	*106,239*

Tab. 5.1 Výsledky doby výpočtů s pohyblivou desetinnou čárkou

5.2.2 Fourierova transformace

Pro výpočty s Fourierovo transformací bylo experimentálně určeno 1 000 cyklů. Pro program JMathLib pak nastaveno pouze 10 cyklů. Výsledky doby výpočtů ukazuje následující tabulka.

Numerický software	*Doba výpočtu pro Linux [s]*	*Doba výpočtu pro Windows [s]*
MATLAB	*0,885*	*1,256*
GNU Octave	*6,919*	*42,531*
Scilab	*10,047*	*17,669*
FreeMat	*29,304*	*133,266*
JMathLib	*12,480*	*46,714*

Tab. 5.2 Výsledky doby výpočtů s Fourierovo transformací

5.2.3 Maticové operace

Pro výpočty s Maticovými operacemi bylo experimentálně určeno 10 cyklů. Pro program JMathLib pak nastaven pouze 1 cyklus a velikosti matice pro výpočty pouze o velikosti 10 x 10 (oproti velikosti 1 000 x 1 000 pro ostatní software). Následující tabulka v prvním sloupci ukazuje výsledky doby výpočtu sestavování matic, v druhém sloupci dobu trvání matematických operací nad sestavenými maticemi.

Numerický software	*Doba výpočtu pro Linux [s]*		*Doba výpočtu pro Windows [s]*	
	Sestavování	*Operace*	*Sestavování*	*Operace*
MATLAB	*0,050*	*0,635*	*0,614*	*0,702*
GNU Octave	*19,837*	*0,691*	*26,452*	*0,695*
Scilab	*3,450*	*0,251*	*6,256*	*0,484*
FreeMat	*5,578*	*6,244*	*2,734*	*31,535*
JMathLib	*0,346*	*0,428*	*0,344*	*0,641*

Tab. 5.3 Výsledky doby výpočtů s maticovými operacemi

Na operačním systému Linux v numerickém programu Scilab se vyskytla chyba v zápisu do operační paměti. Program byl při výpočtu ukončen operačním systémem chybou SEGFAULT, což znamená pokus programu zapsat data do nepovolené části operační paměti [48]. Bylo zjištěno, že pád aplikace způsobuje výpočet determinantu matice. Proto byl z algoritmu při testování odstraněn a výpočet proveden bez determinantu. Pro porovnání rychlosti výpočtu determinantu byl spuštěn stejný výpočet bez determinantu matice v programu

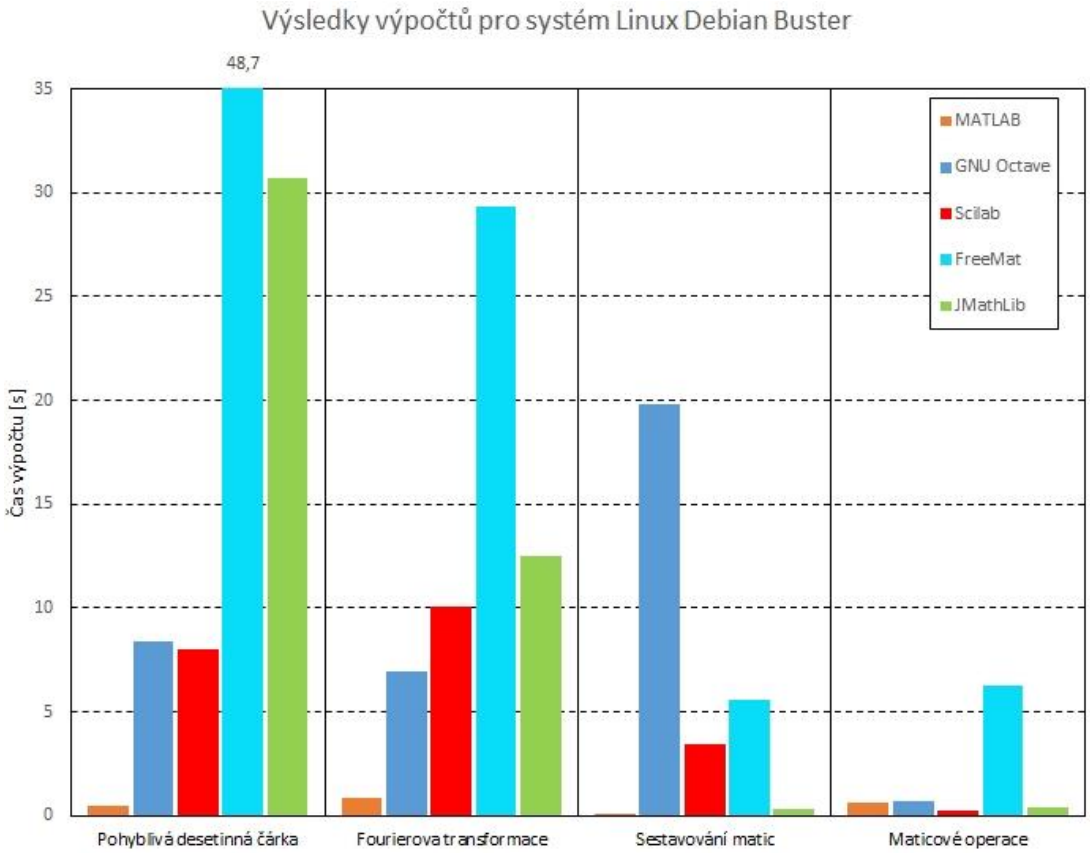
GNU Octave, který trval 0,298 oproti normálnímu výpočtu s časem 0,691 s. Výpočet vykazuje zrychlení o 57 %, což je značný rozdíl. O výpočtu determinantu matice proto lze říct, že jde o dominantní operaci z hlediska výpočetního výkonu. Výsledky pro operační systém Linux v programu Scilab proto nelze brát jako průkazný.

5.3 Porovnání výsledku výpočtů

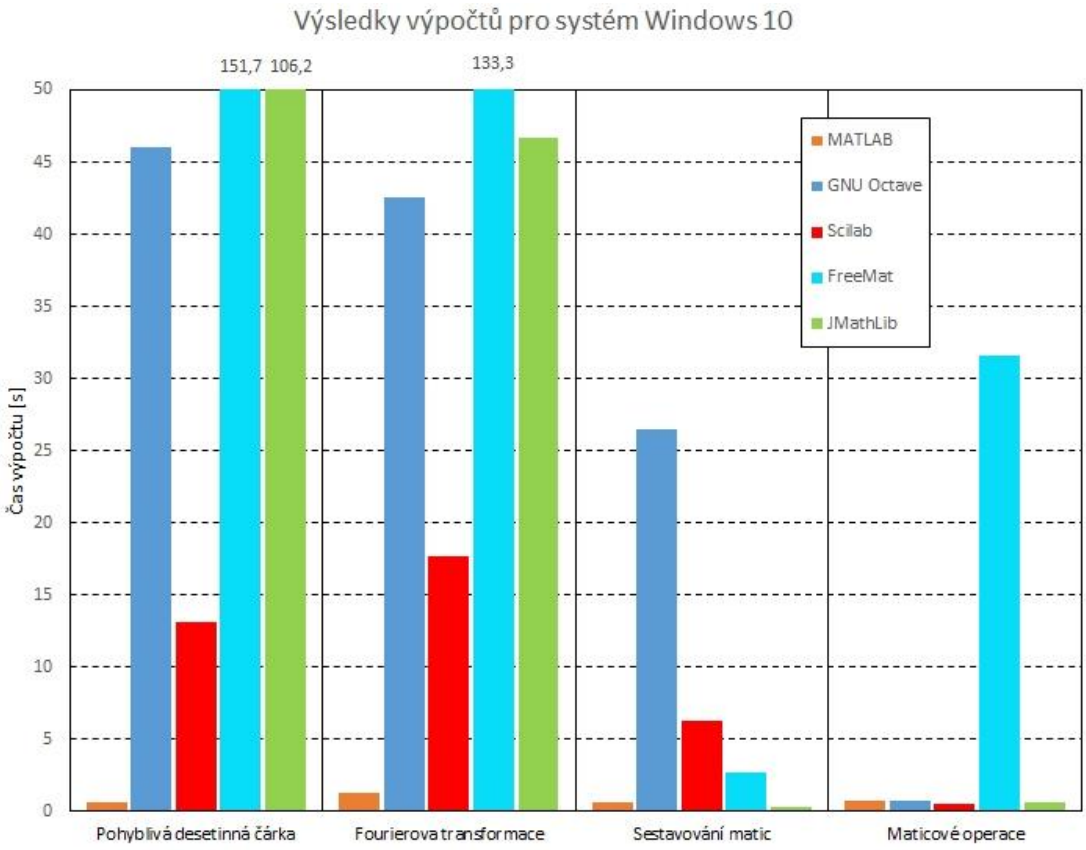
5.3.1 Porovnání výkonů numerických programů

Následující grafy interpretují výsledky výkonů numerický programů. První dva grafy porovnávají výkon jednotlivých algoritmů vztažené na OS. Ukazují časy potřebné k výpočtu algoritmu a lze tak vyčíst, který numerický program je pro danou problematiku nejrychlejší či nejpomalejší a podobně.

Programy jsou v grafu barevně rozlišeny. Příliš vysoké hodnoty nejsou v grafu vykresleny celé, ale z důvodů přehlednosti jsou vypsány číselně nad osou.



Obr. 5.4 *Výsledky výpočtů pro OS Linux Debian Buster*



Obr. 5.5 *Výsledky výpočtů pro OS Windows 10*

Jak je patrné z grafů, bezkonkurenčně nejrychlejší numerický program je MATLAB, a to na oba operační systémy. V operacích s pohyblivou desetinnou čárkou, Fourierovou transformací a sestavování matic dosahuje vyšší rychlosti o jeden a v některých případech až o dva řády. Co se týče maticových operací, jeho rychlost je srovnatelná s ostatními programy.

Rychlost programu GNU Octave pro OS Linux je v porovnání s ostatními neplacenými programy nadprůměrná. I pro OS Windows je délka výpočtu nadprůměrná, avšak trvají déle než pod Linuxem. Pro algoritmus sestavování matic si program vedl nejhůře, výpočty byly až šestkrát pomalejší než u ostatních programů.

Mezi neplacenými programy si nejlépe vedl numerický program Scilab. Kromě výpočtů Fourierovy transformace pod OS Linux a sestavování matic pod OS Windows je ve všech výpočtech nejrychlejší. Protože výpočet determinantu matice u maticových operací na OS Linux způsoboval pád, byl z výpočtu odstraněn, neboť výslednou hodnotu nelze brát jako průkaznou. Nicméně na OS Windows jsou maticové operace rychlejší než v programu MATLAB.

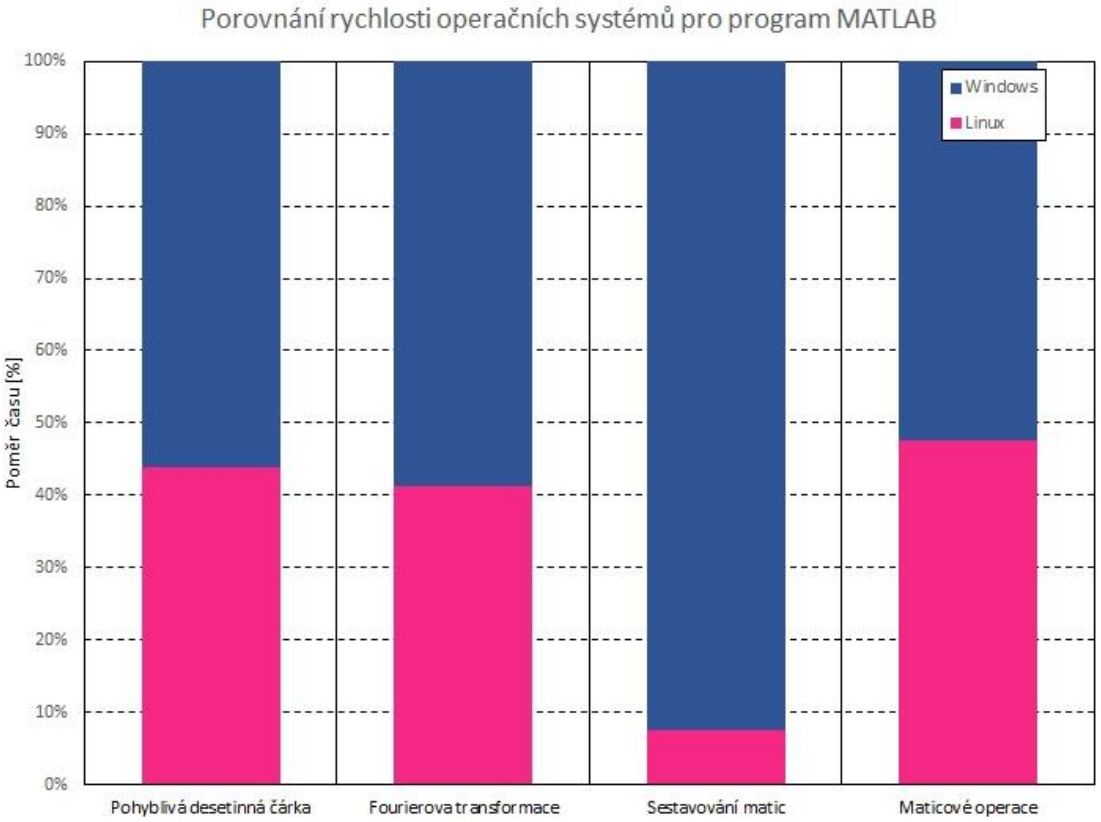
Numerický program FreeMat vykazuje spíše špatné výkonové výsledky. Oproti neplaceným programům GNU Octave a Scilab u algoritmů s pohyblivou desetinnou čárkou a Fourierovo transformací je program výrazně pomalejší. U algoritmu pro sestavování matic si program vedl dobře, nicméně naopak u maticových operací vykázal velmi špatné výsledky.

Pro numerický program JMathLib byly před spouštěním algoritmů upraveny počty cyklů, které jsou řádově nižší. Jeho výsledky v grafech tudíž nejsou v poměru a nemohou být posuzovány proti sobě. Kdybychom vynásobili čas, který vyšel oproti počtům cyklů ostatních programů, vyšel by nám například pro algoritmus s pohyblivou desetinnou čárkou čas výpočtu zhruba kolem 42 hodin. Program je tedy oproti ostatním extrémně pomalejší. Do grafu byl takto zanesen především pro získání základního přehledu o jeho výkonu, bez vztažení na ostatní numerické programy.

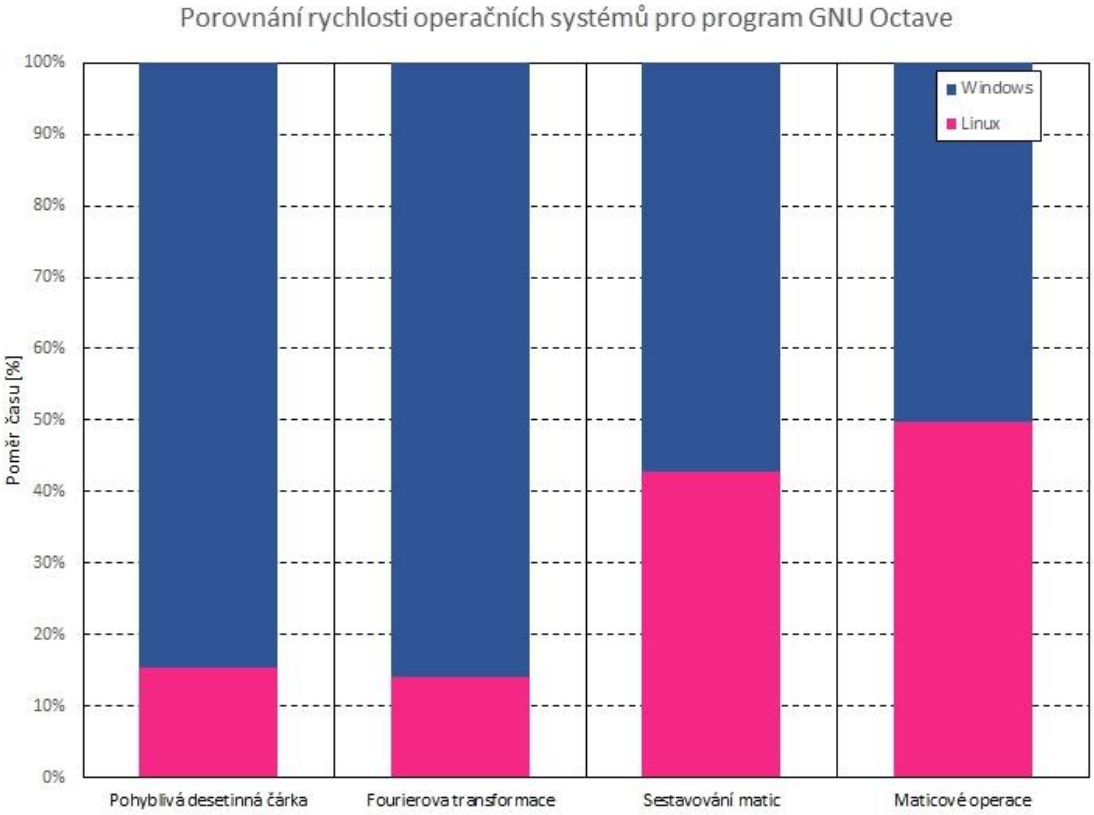
5.3.2 Porovnání výkonů použitých operačních systémů

Následující grafy ukazují poměr času, který byl potřeba pro vykonání algoritmu pro operační systém Windows a Linux. V každém grafu jsou vykresleny poměry pro všechny algoritmy daného numerického programu. Tudíž lze vyčíst jaký OS je pro výpočet vhodnější.

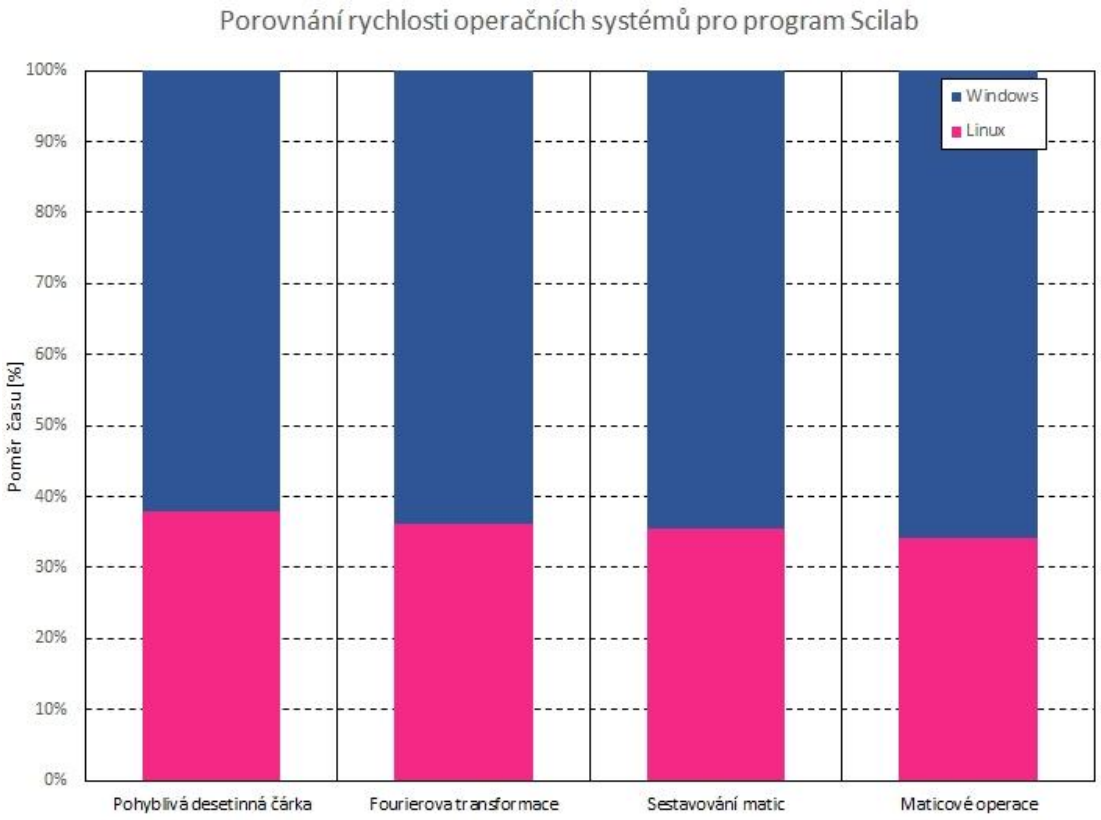
Hodnoty u grafů 50 % Windows a 50 % Linux znamenají, že algoritmus na obou systémech trval stejně dlouho. Posun nahoru znamená potřebu menšího času na výpočet pro OS Windows. Naopak při posunu dolů pak potřebu menšího času na výpočet pro OS Linux. Hodnoty jsou vztažené relativně k procentům.



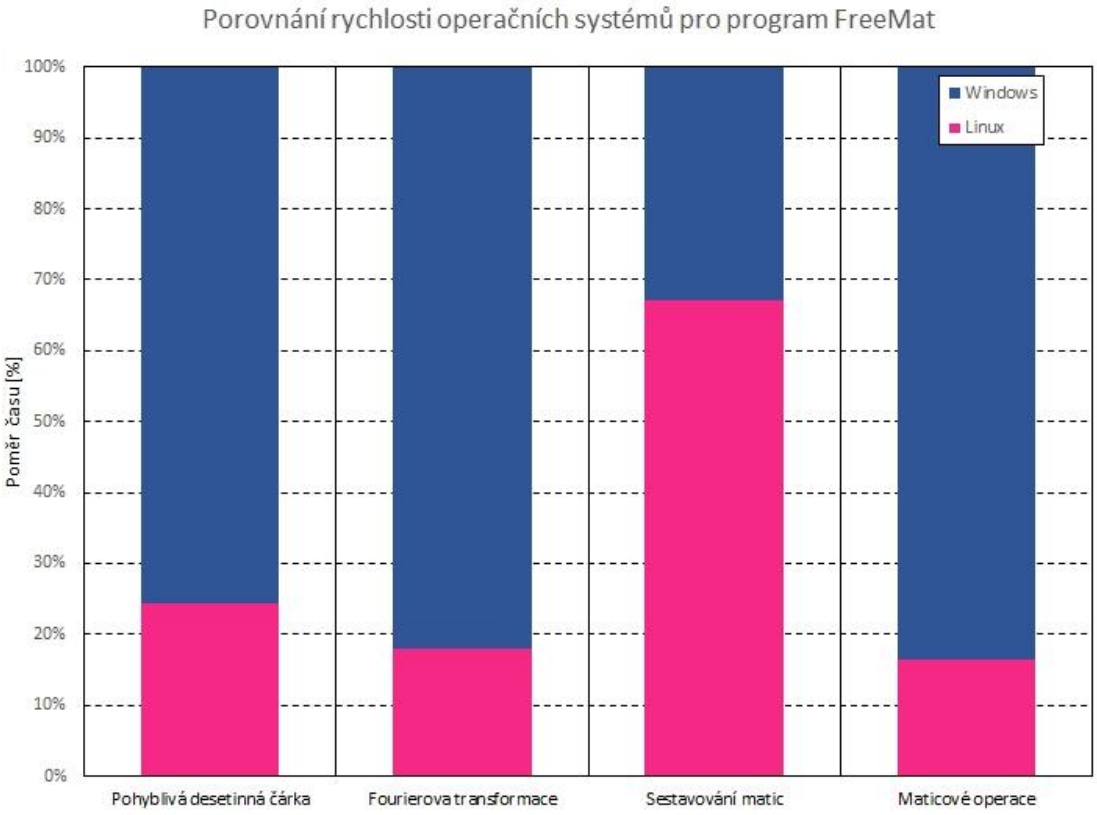
Obr. 5.6 *Porovnání rychlosti OS pro program MATLAB*



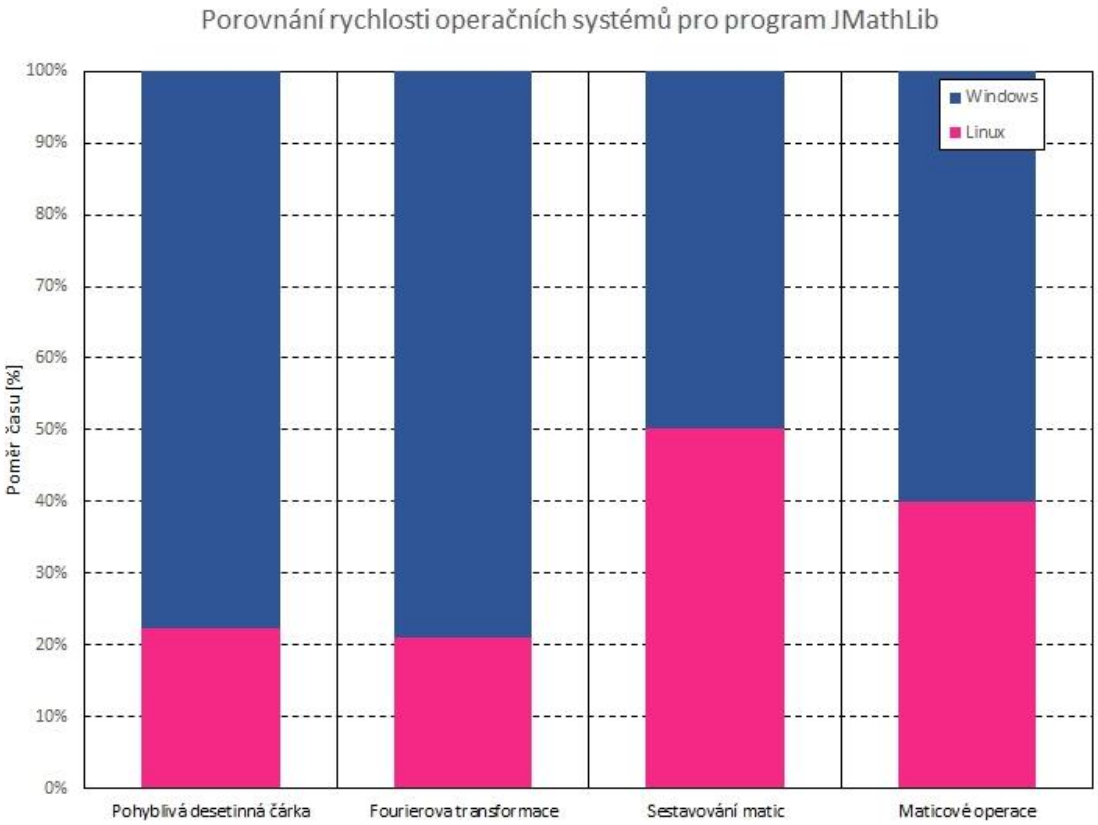
Obr. 5.7 *Porovnání rychlosti OS pro program GNU Octave*



Obr. 5.8 *Porovnání rychlosti OS pro program Scilab*



Obr. 5.9 *Porovnání rychlosti OS pro program FreeMat*



Obr. 5.10 *Porovnání rychlosti OS pro program JMathLib*

Z grafů plyne, že doba výpočtu algoritmů v program MATLAB je na operačních systémech vyrovnaná, přičemž v OS Linux jsou o něco rychlejší. Zásadní rozdíl přichází u sestavování matic, kde je MATLAB výrazně rychlejší v OS Linux.

Algoritmy s pohyblivou desetinnou čárkou a Fourierova transformace v programu GNU Octave je rychlejší, pokud jsou spuštěny v OS Linux. Rychlosti algoritmů sestavování matic a maticové operace jsou pro oba operační systémy podobné, přičemž sestavování matic je o něco rychlejší v OS Linux.

Pro numerický program Scilab jsou výsledky pro všechny algoritmy vyrovnané. Rychlejší OS je opět Linux. Poměr algoritmu maticových operací je opět zkreslen s výhodou pro OS Linux, protože nebyl vypočítáván determinant matice, viz výše.

Numerický program FreeMat vykazuje zásadně vyšší rychlost, pokud je spouštěn pod OS Linux, kromě algoritmu pro sestavování matic, který je rychlejší pod OS Windows.

Numerický program JMathLib je opět zásadně rychlejší pod OS Linux, kromě sestavování matic, kde byla rychlost stejná. Nicméně velikost sestavované matice byla pouze o rozměrech 10 x 10. Časy výpočtu vyšly malé, proto lze předpokládat, že při sestavování větší matice budou výsledky jiné. Nutno zmínit, že program běží čistě na programovacím jazyce Java, a tudíž i jeho výpočty. Nicméně pro každý operační systém byla naistalována jiná verze běhového prostředí Javy. Je možné, že i tento rozdíl má vliv na výsledky.

5.4 Zhodnocení výsledku výpočtů

Z výsledku vyšlo najevo, že numerický program MATLAB je nejrychlejší ze všech testovaných softwarů. Pro všechny testované skripty měl nejvyšší výkon a to až o několik řádů. Jen v jedné variantě byl pomalejší, konkrétně při maticových operacích v OS Windows, avšak jedná se jen o malý rozdíl.

Mezi numerickými programy, které jsou zdarma, má nejlepší výsledky výkonu program Scilab a to pro všechny testované algoritmy. Jako nevýhodu lze považovat nestabilitu, která se projevila pádem aplikace v OS Linux. Nicméně nebylo určeno, kde vzniká chyba. Je tedy možné, že chyba byla způsobena externím zdrojem.

Třetí pomyslnou příčku lze přiřadit, co se výkonu týče, programu GNU Octave. Některé výsledky výkonů algoritmů jsou srovnatelné s programem Scilab, většinou jsou však spíše horší.

Výkon programu FreeMat v porovnání s ostatními je spíše nižší. Časové výsledky výpočtů algoritmů byly oproti ostatním softwarům výrazně delší, avšak stále ve stejném řádu. Tudíž lze říci, že se na složitější výpočty příliš nehodí.

Program JMathLib je z testovaných programů nejpomalejší a má značně nízký výkon. Oproti ostatním programům se jeho výkon pohybuje o několik řádů níže. Na složité, ale i na jednodušší výpočty se příliš nehodí.

Pro většinu výpočtů, kromě algoritmu sestavování matic v programu FreeMat, vykazovaly numerické programy značně vyšší výkon na OS Linux. Ačkoliv byly dodrženy stejné verze

programů a nainstalovány jako x64. Náročné výpočty je tedy mnohem výhodnější spouštět pod operačním systémem Linux a dosáhnout tak značně lepších výkonových výsledků.

Závěr

Vybrané numerické programy pro porovnání numerického výkonu byly zhodnoceny ve více kategoriích. Široce zaměřené hodnocení nám poskytlo dostatečný nadhled a usnadnilo vybrání konkrétního software, který je nejvhodnější a nejvíce odpovídá dané specifikaci a situaci.

Při hodnocení numerických programů bylo cíleno na jejich uživatelské rozhraní a intuitivnost, pokročilost vývoje, dostupnost, přenositelnost skriptů a výkon. Pro testování výkonu byly navrženy algoritmy s pohyblivou a pevnou desetinnou čárkou, Fourierova transformace a algoritmy pro maticové operace. Výpočty s pevnou desetinnou čárkou se povedlo realizovat jen v numerickém programu MATLAB, ostatní programy pevnou desetinnou čárku nepodporují a nebylo tak možné pro algoritmus porovnat výkon.

Testování výkonu proběhlo na laptopu, na kterém byly nainstalovány dva operační systémy, Linux a Windows. Na obou operačních systémech byly numerické programy naistalovány ve stejné verzi. Následně byly postupně spuštěny totožné skripty a změřena doba potřebná k jejich výpočtům.

Program MATLAB je placený, cena standardní licence je 2 150 dolarů, avšak program nabízí pokročilé funkce, intuitivní rozhraní a vysokou rychlost. Ze všech porovnávaných softwarů byl program MATLAB nejrychlejší a to v některých případech i o několik řádů. Srovnatelná rychlost s ostatními je u maticových operací.

Další z testovaných programů byl program Scilab, který je šířen zdarma, nabízí dobré a intuitivní rozhraní s veškerou potřebnou funkčností. Syntaxe numerického softwaru je nejvíce odlišná a každý skript se musí buď přepsat, nebo překonvertovat, není možno tudíž jednoduše spouštět stejný skript. Z programů, které jsou poskytovány zdarma, měl nejlepší výsledky testovacích algoritmů, program má tedy nejvyšší výkon. Přesto nemusí být ideální variantou vzhledem ke složitější přenositelnosti skriptů.

Známý program GNU Octave, který je šířen zdarma, taktéž poskytuje dobré a intuitivní rozhraní poskytující veškerou funkčnost. Syntaxe je velice podobná programu MATLAB a zajištuje vysokou přenositelnost. Při přenosu algoritmů nemusely být skripty upravovány.

Výkon programu je lehce nadprůměrný. Pro svoji syntaktickou přenositelnost se hodí jako bezplatná alternativa pro MATLAB, avšak o menším výkonu.

Program FreeMat, šířený zdarma, nabízí podobné rozhraní a funkce jako programy Scilab a GNU Octave. Syntaxe je velice podobná programu MATLAB, kromě jedné funkce nemusely být skripty nijak upravovány. Výkon programu je nízký, zejména u maticových operací. Program se tudíž hodí spíše pro nenáročné operace.

Poslední program JMathLib, taktéž šířený zcela zdarma, poskytuje velice špatné a neintuitivní rozhraní. V programu nefunguje například výběr adresáře obsahující programy, není dostupný editor kódu apod. Práce s programem je tedy velice složitá. Syntaxe je podobná programu MATLAB, nicméně obsahuje rozdíly a při přenášení musel být každý skript upraven. Výkon programu je velice špatný a až o několik řádu pomalejší než ostatní programy. Pro porovnání výkonu musely být skripty upraveny tak, aby bylo možné zaznamenat doběhnutí programu. Program se nehodí na složité ani na méně náročné skripty, pouze na jednoduché výpočty. Jeho výhoda může být možnost implementace do externí aplikace a využití jako hotové řešení pro matematické výpočty.

Pro většinu výpočtů, vykazovaly numerické programy značně vyšší výkon na operačním systému Linux. Náročné výpočty je tudíž mnohem výhodnější místo operačního systému Windows spouštět pod operačním systémem Linux a dosáhnout tak značně lepších výkonových výsledků.

Seznam literatury a informačních zdrojů

[1] *Comparison of numerical analysis software - Wikipedia* [online]. Poslední změna 24.02.2018 [cit. 27.04.2018]. Dostupné z: https://en.wikipedia.org/wiki/Comparison_of_numerical_analysis_software

[2] *MathWorks - Company Overview* [online] ©2018. [cit. 23.04.2018]. Dostupné z: https://www.mathworks.com/company/aboutus.html

[3] *MathWorks - Company Fact Sheet* [online] ©2018. [cit. 23.04.2018]. Dostupné z: https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/c/company-fact-sheet-8282v18.pdf

[4] *MathWorks - MATLAB Answers: Which computer programming language are MATLAB R2015b and R2016b written in?* [online]. ©2018. [cit. 23.04.2018]. Dostupné z: https://www.mathworks.com/matlabcentral/answers/280083-which-computer-programming-language-are-matlab-r2015b-and-r2016b-written-in

[5] *Wikipedia – MATLAB* [online]. Poslední změna 05.04.2018 [cit. 23.04.2018]. Dostupné z: https://en.wikipedia.org/wiki/MATLAB

[6] *Pricing and Licensing - MATLAB & Simulink* [online]. ©2018. [cit. 24.04.2018]. Dostupné z: https://www.mathworks.com/pricing-licensing.html?prodcode=ML&intendeduse=comm

[7] *Get Involved. The GNU Operating System and the Free Software Movement* [online]. ©2018. [cit. 24.04.2018]. Dostupné z: https://www.gnu.org/software/octave/get-involved.html

[8] *FAQ - Octave* [online]. Poslední změna 18.04.2018 [cit. 24.04.2018]. Dostupné z: https://wiki.octave.org/FAQ#How_can_I_get_involved_in_Octave_development.3F

[9] *About. The GNU Operating System and the Free Software Movement* [online]. ©2018. [cit. 24.04.2018]. Dostupné z: https://www.gnu.org/software/octave/about.html

[10] *History / Scilab / Home - Scilab* [online]. ©2017. [cit. 26.04.2018]. Dostupné z: https://www.scilab.org/scilab/history

[11] *Development / Development / Home - Scilab* [online]. ©2017. [cit. 26.04.2018]. Dostupné z: https://www.scilab.org/en/development/quality_assurance

[12] *About. The GNU Operating System and the Free Software Movement* [online]. Poslední změna 20.08.2015 [cit. 26.04.2018]. Dostupné z: https://wiki.scilab.org/Programming%20languages%20in%20Scilab%20and%20their%20usage

[13] *License / Scilab / Home - Scilab* [online]. ©2017. [cit. 24.04.2018]. Dostupné z: https://www.scilab.org/en/scilab/license

[14] *JMathLib site – Developer Team* [online]. Poslední změna 27.04.2018 [cit. 27.04.2018]. Dostupné z: http://www.jmathlib.de/developers.php

[15] *JMathLib - A Java Clone of Octave, SciLab, Freemat and Matlab.* [online] Poslední změna 27.04.2018 [cit. 27.04.2018]. Dostupné z: http://www.jmathlib.de/

[16] *JMathLib - Free Software Directory* [online]. ©2017. [cit. 27.04.2018]. Dostupné z: https://directory.fsf.org/wiki/JMathLib#tab=Overview

[17] *SourceForge - FreeMat / News* [online]. ©2018. [cit. 27.04.2018]. Dostupné z: https://sourceforge.net/p/freemat/news/

[18] *SourceForge - FreeMat / Wiki / Home* [online]. ©2018. [cit. 27.04.2018]. Dostupné z: https://sourceforge.net/p/freemat/wiki/Home/

[19] *SourceForge - FreeMat / Code / [r4479]* [online]. ©2018. [cit. 27.04.2018]. Dostupné z: https://sourceforge.net/p/freemat/code/HEAD/tree/

[20] *FreeMat* [online]. [cit. 27.04.2018]. Dostupné z: http://freemat.sourceforge.net/

[21] *Start stopwatch timer - MATLAB tic.* [online]. ©2018. [cit. 05.05.2018]. Dostupné z: https://www.mathworks.com/help/matlab/ref/tic.html

[22] *MATLAB Programming/Differences between Octave and MATLAB* [online]. Poslední změna 26.04.2018 [cit. 07.05.2018]. Dostupné z: https://en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MATLAB

[23] *tic - Start a stopwatch timer. Scilab Online Help* [online]. Poslední změna 12.02.2018 ©2017 [cit. 07.05.2018]. Dostupné z: https://help.scilab.org/docs/6.0.1/en_US/tic.html

[24] *Transpose vector or matrix - MATLAB transpose .'. MATLAB & Simulink* [online]. Poslední změna 12.02.2018 ©2018 [cit. 07.05.2018]. Dostupné z: https://www.mathworks.com/help/matlab/ref/transpose.html

[25] *' (Matlab operator) - Transpose. Scilab Online Help* [online]. Poslední změna 12.02.2018 ©2017 [cit. 07.05.2018]. Dostupné z: https://help.scilab.org/docs/6.0.1/en_US/m2sci_transpose.html

[26] *Ratio of circle's circumference to its diameter - MATLAB pi. MATLAB & Simulink* [online]. ©2018 [cit. 07.05.2018]. Dostupné z: https://www.mathworks.com/help/matlab/ref/pi.html

[27] *Plot discrete sequence data - MATLAB stem. MATLAB & Simulink* [online]. ©2018 [cit. 07.05.2018]. Dostupné z: https://www.mathworks.com/help/matlab/ref/stem.html

[28] *plot2d3 - 2D plot (vertical bars). Scilab Online Help* [online]. Poslední změna 12.02.2018 ©2017 [cit. 07.05.2018]. Dostupné z: https://help.scilab.org/docs/6.0.1/ru_RU/plot2d3.html

[29] *xgrid - Add a grid on a 2D plot. Scilab Online Help* [online]. Poslední změna 12.02.2018 ©2017 [cit. 07.05.2018]. Dostupné z: https://help.scilab.org/docs/6.0.1/en_US/xgrid.html

[30] *Debugging. Scilab Online Help* [online]. Poslední změna 01.05.2015 ©2017 [cit. 07.05.2018]. Dostupné z: https://help.scilab.org/doc/5.5.2/en_US/section_f30f4937bdb6cfca519a6e632d13c988.html

[31] *JMathLib site – plot* [online]. Poslední změna 05.08.2018 [cit. 08.05.2018]. Dostupné z: http://www.jmathlib.de/docs/handbook/function_plot.php

[32] *A Visual Guide to Swing Components. MIT - Massachusetts Institute of Technology* [online]. ©2015 [cit. 08.05.2018]. Dostupné z: http://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html

[33] *Notepad++ Home. Notepad++ Home* [online]. © Don Ho 2018 [cit. 22.05.2018]. Dostupné z: https://notepad-plus-plus.org/

[34] *Fast Fourier transform - MATLAB fft. - MATLAB & Simulink* [online]. ©2018 [cit.

11.05.2018]. Dostupné z: https://www.mathworks.com/help/matlab/ref/fft.html

[35] *Double-precision arrays – MATLAB. MATLAB & Simulink* [online]. ©2018 [cit. 08.05.2018]. Dostupné z: https://www.mathworks.com/help/matlab/ref/double.html

[36] *Floating-Point Arithmetic -- from Wolfram MathWorld. Wolfram MathWorld: The Web's Most Extensive Mathematics Resource* [online]. ©2018 [cit. 11.05.2018]. Dostupné z: http://mathworld.wolfram.com/Floating-PointArithmetic.html

[37] *Discrete Fourier Transform -- from Wolfram MathWorld. Wolfram MathWorld: The Web's Most Extensive Mathematics Resource* [online]. ©2018 [cit. 11.05.2018]. Dostupné z: http://mathworld.wolfram.com/DiscreteFourierTransform.html

[38] *IEEE 754 - Wikipedia.* [online]. Poslední změna 16.05.2018 [cit. 18.05.2018]. Dostupné z: https://en.wikipedia.org/wiki/IEEE_754

[39] *Discrete Fourier Transform -- from Wolfram MathWorld. Wolfram MathWorld:* [online]. ©2018 [cit. 22.05.2018]. http://mathworld.wolfram.com/FourierTransform.html

[40] *IEEE - The world's largest technical professional organization dedicated to advancing technology for the benefit of humanity. IEEE - The world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.* [online]. ©2018 [cit. 18.05.2018]. Dostupné z: https://www.ieee.org/

[41] *Construct fixed-point numeric object - MATLAB fi. - MATLAB & Simulink* [online]. ©2018 [cit. 12.05.2018]. Dostupné z: https://www.mathworks.com/help/fixedpoint/ref/fi.html

[42] *Operations with Fixed-Point Data - MATLAB & Simulink* [online]. ©2018 [cit. 12.05.2018]. Dostupné z: https://www.mathworks.com/help/stateflow/ug/operations-with-fixed-point-data.html

[43] *Perform Fixed-Point Arithmetic - MATLAB & Simulink* [online]. ©2018 [cit. 12.05.2018]. Dostupné z: https://www.mathworks.com/help/fixedpoint/gs/fixed-point-arithmetic_bt25flf-1.html

[44] David Bateman, Laurent Mazet, Véronique Buzenac-Settineri, Markus Muck. *A FIXED-POINT TYPE FOR OCTAVE* [online]. Saint Aubin91193 Gif-Sur-Yvette Cedex – France, 1.4.2006 [cit. 13.05.2018]. Dostupné z: https://www.researchgate.net/publication/1851245_A_Fixed-Point_Type_for_Octave

[46] *Octave-Forge. Octave-Forge* [online]. [cit. 13.05.2018]. Dostupné z: https://octave.sourceforge.io/packages.php

[46] *octave fixed point package: from where to download ?- NARKIVE* [online]. [cit. 13.05.2018]. Dostupné z: http://octave-dev.narkive.com/XKTa98km/octave-fixed-point-package-from-where-to-download

[47] *Intel® Core™ i5-8250U Processor (6M Cache, up to 3.40 GHz) Product Specifications. Intel® Product Specifications* [online]. [cit. 16.05.2018]. Dostupné z: https://ark.intel.com/products/124967/Intel-Core-i5-8250U-Processor-6M-Cache-up-to-3_40-GHz

[48] *Segmentation fault - Wikipedia* [online]. Poslední změna 10.03.2018 [cit. 22.05.2018]. Dostupné z: https://en.wikipedia.org/wiki/Segmentation_fault

Přílohy

Příloha A – Výpočty s pohyblivou desetinnou čárkou pro program MATLAB

```
clear;

% Výpočty s pohyblivou desetinnou čárkou, doporučené matematické operace
% podle standardu IEEE 754

x = 0;
y = 0;

% Nastavuje dvojitou přesnost proměnných
% Ačkoliv jsou jako výchozí stav v MATLABu
double(x);
double(y);

% Konstanty
loops = 100000; % Udává počet cyklů

tic
for i = 1:loops;
    y = i;

    % Goniometrické funkce
    x = sin(y);
    y = cos(x);

    x = cosh(y);
    y = sinh(x);

    x = exp(x);
    y = log(y);

    % Mocnina
    x = x^abs(y);

    % Odmocnina, nutné absolutní hodnoty pro vyvarování komplexních čísel
    y = abs(y)^(1/abs(x));

    % Pythagorova věta, funkce hypot
    x = sqrt(x^2+y^2);

    % Převrácená hodnota odmocniny
    y = sqrt(1/abs(x));
end
toc

% Pro kontrolu vypsané hodnoty, zdali se výpočty nedostaly mimo nechtěné
% rozsahy
format long;
disp(x);
disp(y);
```

Příloha B – Fourierova transformace pro program MATLAB

```
clear;

% Konstanty
loops = 100000; % Udává počet cyklů
x = [cos(0), cos(pi/6), cos(pi/4), cos(pi/3), cos(pi/2), cos(2/3*pi),
cos(3/4*pi), cos(5/6*pi), cos(pi), cos(7/6*pi), cos(5/4*pi), cos(4/3*pi),
cos(3/2*pi), cos(5/3*pi), cos(7/4*pi), cos(11/6*pi)];
N = length(x);

% Allokování pole
y = zeros(1, N);

tic
for i = 1:loops;
    for k = 1:N
        y(k) = 0;

        for n = 1:N
            y(k) = y(k)+x(n)*exp(-1i*2*pi*(k-1)*(n-1)/N);
        end
    end
    mag = 2*abs(y)/N;
end
toc

% Graf vstupu
t = 0:N-1;
subplot(1, 2, 1);
stem(t, x);
xlabel('n');
title('Vstup');
grid on;

% Graf FFT sekvence
subplot(1,2,2);
stem(t(1:N/2), mag(1:N/2));
xlabel('k');
title('FFT sekvence');
grid on;
```

Příloha C – Maticové operace pro program MATLAB

```
clear;

% Inicializace proměnných

% Ve skutečnosti je velikost matice krát velikost referenčního vektoru
size = 100;

loops = 10;

% Referenční vektor stejné velikosti
A1 = [5 2 7 9 6 1 0 9 5 6];
A2 = [3 3 7 6 0 8 0 1 2 5];

Asize = length(A1);
matriceSize = Asize * size;

rotA1 = zeros(1, Asize);
rotA2 = zeros(1, Asize);
B1 = zeros(matriceSize, matriceSize);
B2 = zeros(matriceSize, matriceSize);

% Sestavování pseudonáhodných matic
tic
i = 1;
j = 1;
while i <= matriceSize
    for j = 1:matriceSize
        for k = 1:Asize
            B1(j, i+k-1) = A1(k);
            B2(j, i+k-1) = A2(k);

            if k ~= Asize
                incIndex = k + 1;
            else
                incIndex = 1;
            end

            rotA1(k) = A1(incIndex);
            rotA2(k) = A2(incIndex);
        end
        A1 = rotA1;
        A2 = rotA2;
    end
    i = i + Asize;
end
toc

% Základní operace s maticemi
tic
for i = 1:loops
    B = B1 + B2;
    B = B1 * B;
    B = transpose(B);
    d = det(B);
end
toc
```

Příloha D – Výpočty s pevnou desetinnou čárkou pro program MATLAB

```
clear;

% Výpočty s pevnou desetinnou čírkou, doporučené matematické operace
% Práce s unsigned typem

% Konstanty.
loops = 10000; % Udává počet cyklů

% Počáteční inicializace
x = fi(1);

tic
for i = 1:loops;
    y = fi(i);

    % Podporované binární operace
    x = x + y;
    x = x - y;

    x = x / y;
    x = x * y;

    % Násobení čísla s pevnou desetinnou čárkou jiným typem
    % Operace zanese do výpočtu desetinná místa
    x = y * 0.1415;
end
toc
% Pro kontrolu vypsané hodnoty, zdali se výpočty nedostaly mimo nechtěné
% rozsahy
format long;
disp(x);
disp(y);
```

Příloha E – Výpočty s pohyblivou desetinnou čárkou pro program GNU Octave

```
clear;

% Výpočty s pohyblivou desetinnou čárkou, doporučené matematické operace
% podle standardu IEEE 754

x = 0;
y = 0;

% Nastavuje dvojitou přesnost proměnných
% Ačkoliv jsou jako výchozí stav v MATLABu
double(x);
double(y);

% Konstanty
loops = 1000000; % Udává počet cyklů

tic
for i = 1:loops;
    y = i;

    % Goniometrické funkce
    x = sin(y);
    y = cos(x);

    x = cosh(y);
    y = sinh(x);

    x = exp(x);
    y = log(y);

    % Mocnina
    x = x^abs(y);

    % Odmocnina, nutné absolutní hodnoty pro vyvarování komplexních čísel
    y = abs(y)^(1/abs(x));

    % Pythagorova věta, funkce hypot
    x = sqrt(x^2+y^2);

    % Převrácená hodnota odmocniny
    y = sqrt(1/abs(x));
end
toc

% Pro kontrolu vypsané hodnoty, zdali se výpočty nedostaly mimo nechtěné
% rozsahy
format long;
disp(x);
disp(y);
```

Příloha F – Fourierova transformace pro program GNU Octave

```
clear;

% Konstanty
loops = 10000; % Udává počet cyklů
x = [cos(0), cos(pi/6), cos(pi/4), cos(pi/3), cos(pi/2), cos(2/3*pi),
cos(3/4*pi), cos(5/6*pi), cos(pi), cos(7/6*pi), cos(5/4*pi), cos(4/3*pi),
cos(3/2*pi), cos(5/3*pi), cos(7/4*pi), cos(11/6*pi)];
N = length(x);

% Allokování pole
y = zeros(1, N);

tic
for i = 1:loops;
    for k = 1:N
        y(k) = 0;

        for n = 1:N
            y(k) = y(k)+x(n)*exp(-1i*2*pi*(k-1)*(n-1)/N);
        end
    end
    mag = 2*abs(y)/N;
end
toc

% Graf vstupu
t = 0:N-1;
subplot(1, 2, 1);
stem(t, x);
xlabel('n');
title('Vstup');
grid on;

% Graf FFT sekvence
subplot(1,2,2);
stem(t(1:N/2), mag(1:N/2));
xlabel('k');
title('FFT sekvence');
grid on;
```

Příloha G – Maticové operace pro program GNU Octave

```
clear;

% Ininicializace proměnných

% Ve skutečnosti je velikost matice krát velikost referenčního vektoru
size = 100;

loops = 10;

% Referenční vektor stejné velikosti
A1 = [5 2 7 9 6 1 0 9 5 6];
A2 = [3 3 7 6 0 8 0 1 2 5];

Asize = length(A1);
matriceSize = Asize * size;

rotA1 = zeros(1, Asize);
rotA2 = zeros(1, Asize);
B1 = zeros(matriceSize, matriceSize);
B2 = zeros(matriceSize, matriceSize);

% Sestavování pseudonáhodných matic
tic
i = 1;
j = 1;
while i <= matriceSize
    for j = 1:matriceSize
        for k = 1:Asize
            B1(j, i+k-1) = A1(k);
            B2(j, i+k-1) = A2(k);

            if k ~= Asize
                incIndex = k + 1;
            else
                incIndex = 1;
            end

            rotA1(k) = A1(incIndex);
            rotA2(k) = A2(incIndex);
        end
        A1 = rotA1;
        A2 = rotA2;
    end
    i = i + Asize;
end
toc

% Základní operace s maticemi
tic
for i = 1:loops
    B = B1 + B2;
    B = B1 * B;
    B = transpose(B);
    d = det(B);
end
toc
```

Příloha H – Výpočty s pohyblivou desetinnou čárkou pro program Scilab

```
clear;

// Výpočty s pohyblivou desetinnou čárkou, doporučené matematické operace
// podle standardu IEEE 754

x = 0;
y = 0;

// Nastavuje dvojitou přesnost proměnných
// Ačkoliv jsou jako výchozí stav v MATLABu
double(x);
double(y);

// Konstanty
loops = 1000000; // Udává počet cyklů

tic()
for i = 1:loops;
    y = i;

    // Goniometrické funkce
    x = sin(y);
    y = cos(x);

    x = cosh(y);
    y = sinh(x);

    x = exp(x);
    y = log(y);

    // Mocnina
    x = x^abs(y);

    // Odmocnina, nutné absolutní hodnoty pro vyvarování komplexních čísel
    y = abs(y)^(1/abs(x));

    // Pythagorova věta, funkce hypot
    x = sqrt(x^2+y^2);

    // Převrácená hodnota odmocniny
    y = sqrt(1/abs(x));
end
disp(toc());

// Pro kontrolu vypsané hodnoty, zdali se výpočty nedostaly mimo nechtěné
// rozsahy
disp(x);
disp(y);
```

Příloha I – Fourierova transformace pro program Scilab

```
clear;

// Konstanty
loops = 10000; // Udává počet cyklů
x = [cos(0), cos(%pi/6), cos(%pi/4), cos(%pi/3), cos(%pi/2), cos(2/3*%pi),
cos(3/4*%pi), cos(5/6*%pi), cos(%pi), cos(7/6*%pi), cos(5/4*%pi),
cos(4/3*%pi), cos(3/2*%pi), cos(5/3*%pi), cos(7/4*%pi), cos(11/6*%pi)];
N = length(x);

// Alokování pole
y = zeros(1, N);

tic
for i = 1:loops;
    for k = 1:N
        y(k) = 0;

        for n = 1:N
            y(k) = y(k)+x(n)*exp(-1*%i*2*%pi*(k-1)*(n-1)/N);
        end
    end
    mag = 2*abs(y)/N;
end
disp(toc());

// Graf vstupu
t = 0:N-1;
subplot(1, 2, 1);
plot2d3(t, x, color('red'));
xlabel('n');
title('Vstup');
xgrid;

// Graf FFT sekvence
subplot(1,2,2);
plot2d3(t(1:N/2), mag(1:N/2), color('red'));
xlabel('k');
title('FFT sekvence');
xgrid;
```

Příloha J – Maticové operace pro program Scilab

```
clear;

// Ininicializace proměnných

// Ve skutečnosti je velikost matice krát velikost referenčního vektoru
size = 100;

loops = 10;

// Referenční vektor stejné velikosti
A1 = [5 2 7 9 6 1 0 9 5 6];
A2 = [3 3 7 6 0 8 0 1 2 5];

Asize = length(A1);
matriceSize = Asize * size;

rotA1 = zeros(1, Asize);
rotA2 = zeros(1, Asize);
B1 = zeros(matriceSize, matriceSize);
B2 = zeros(matriceSize, matriceSize);

// Sestavování pseudonáhodných matic
tic()
i = 1;
j = 1;
while i <= matriceSize
    for j = 1:matriceSize
        for k = 1:Asize
            B1(j, i+k-1) = A1(k);
            B2(j, i+k-1) = A2(k);

            if k ~= Asize
                incIndex = k + 1;
            else
                incIndex = 1;
            end

            rotA1(k) = A1(incIndex);
            rotA2(k) = A2(incIndex);
        end
        A1 = rotA1;
        A2 = rotA2;
    end
    i = i + Asize;
end
disp(toc());

// Základní operace s maticemi
tic()
for i = 1:loops
    B = B1 + B2;
    B = B1 * B;
    B = B';
    d = det(B);
end
disp(toc());
```

Příloha K – Výpočty s pohyblivou desetinnou čárkou pro program FreeMat

```
clear;

% Výpočty s proměnnou desetinnou čárkou, doporučené matematické operace
% podle standardu IEEE 754

x = 0;
y = 0;

% Nastavuje dvojitou přesnot promměných
% Ačkoliv jsou jako výchozí stav v MATLABu
double(x);
double(y);

% Konstanty
loops = 1000000; % Udává počet cyklů

tic
for i = 1:loops;
    y = i;

    % Goniometrické funkce
    x = sin(y);
    y = cos(x);

    x = cosh(y);
    y = sinh(x);

    x = exp(x);
    y = log(y);

    % Mocnina
    x = x^abs(y);

    % Odmocnina, nutné absolutní hodnoty pro vyvarování komplexních čísel
    y = abs(y)^(1/abs(x));

    % Pythagorova věta, funkce hypot
    x = sqrt(x^2+y^2);

    % Převrácená hodnota odmocniny
    y = sqrt(1/abs(x));
end
toc

% Pro kontolu vypsané hodnoty, zdali se výpočty nedostaly mimo nechtěné
% rozsahy
format long;
disp(x);
disp(y);
```

Příloha L – Fourierova transformace pro program FreeMat

```
clear;

% Konstanty
loops = 10000; % Udává počet cyklů
x = [cos(0), cos(pi/6), cos(pi/4), cos(pi/3), cos(pi/2), cos(2/3*pi),
cos(3/4*pi), cos(5/6*pi), cos(pi), cos(7/6*pi), cos(5/4*pi), cos(4/3*pi),
cos(3/2*pi), cos(5/3*pi), cos(7/4*pi), cos(11/6*pi)];
N = length(x);

% Alokování pole
y = zeros(1, N);

tic
for i = 1:loops;
    for k = 1:N
        y(k) = 0;

        for n = 1:N
            y(k) = y(k)+x(n)*exp(-1i*2*pi*(k-1)*(n-1)/N);
        end
    end
    mag = 2*abs(y)/N;
end
toc

% Graf vstupu
t = 0:N-1;
subplot(1, 2, 1);
plot(t, x, 'x');
xlabel('n');
title('Vstup');
grid on;

% Graf FFT sekvence
subplot(1,2,2);
plot(t(1:N/2), mag(1:N/2), 'x');
xlabel('k');
title('FFT sekvence');
grid on;
```

Příloha M – Maticové operace pro program FreeMat

```
clear;

% Inicializace proměnných

% Ve skutečnosti je velikost matice krát velikost referenčního vektoru
size = 100;

loops = 10;

% Referenční vektor stejné velikosti
A1 = [5 2 7 9 6 1 0 9 5 6];
A2 = [3 3 7 6 0 8 0 1 2 5];

Asize = length(A1);
matriceSize = Asize * size;

rotA1 = zeros(1, Asize);
rotA2 = zeros(1, Asize);
B1 = zeros(matriceSize, matriceSize);
B2 = zeros(matriceSize, matriceSize);

% Sestavování pseudonáhodných matic
tic
i = 1;
j = 1;
while i <= matriceSize
    for j = 1:matriceSize
        for k = 1:Asize
            B1(j, i+k-1) = A1(k);
            B2(j, i+k-1) = A2(k);

            if k ~= Asize
                incIndex = k + 1;
            else
                incIndex = 1;
            end

            rotA1(k) = A1(incIndex);
            rotA2(k) = A2(incIndex);
        end
        A1 = rotA1;
        A2 = rotA2;
    end
    i = i + Asize;
end
toc

% Základní operace s maticemi
tic
for i = 1:loops
    B = B1 + B2;
    B = B1 * B;
    B = transpose(B);
    d = det(B);
end
toc
```

Příloha N – Výpočty s pohyblivou desetinnou čárkou pro program JMathLib

```
clear("variables");

% Výpočty s proměnnou desetinnou čárkou, doporučené matematické operace
% podle standardu IEEE 754

x = 0;
y = 0;

% Nastavuje dvojitou přesnot promměných
% Ačkoliv jsou jako výchozí stav v MATLABu
double(x);
double(y);

% Konstanty
loops = 1000; % Udává počet cyklů

tic
for I = 1:loops;
    y = i;

    % Goniometrické funkce
    x = sin(y);
    y = cos(x);

    x = cosh(y);
    y = sinh(x);

    x = exp(x);
    y = log(y);

    % Mocnina
    x = x^abs(y);

    % Odmocnina, nutné absolutní hodnoty pro vyvarování komplexních čísel
    y = abs(y)^(1/abs(x));

    % Pythagorova věta, funkce hypot
    x = sqrt(x^2+y^2);

    % Převrácená hodnota odmocniny
    y = sqrt(1/abs(x));
end
toc

% Pro kontolu vypsané hodnoty, zdali se výpočty nedostaly mimo nechtěné
% rozsahy
format long;
disp(x);
disp(y);
```

Příloha O – Fourierova transformace pro program JMathLib

```
clear("variables");

% Konstanty
loops = 10; % Udává počet cyklů
x = [cos(0), cos(pi/6), cos(pi/4), cos(pi/3), cos(pi/2), cos(2/3*pi),
cos(3/4*pi), cos(5/6*pi), cos(pi), cos(7/6*pi), cos(5/4*pi), cos(4/3*pi),
cos(3/2*pi), cos(5/3*pi), cos(7/4*pi), cos(11/6*pi)];
N = length(x);

% Alokování pole
y = zeros(1, N);

tic
for I = 1:loops;
    for k = 1:N
        y(k) = 0;

        for N = 1:N
            y(k) = y(k)+x(N)*exp(-1i*2*pi*(k-1)*(N-1)/N);
        end
    end
    mag = 2*abs(y)/N;
end
toc
```

Příloha P – Maticové operace pro program JMathLib

```
clear("variables");

% Inicializace proměnných

% Ve skutečnosti je velikost matice krát velikost referenčního vektoru
size = 1;

loops = 1;

% Referenční vektor stejné velikosti
A1 = [5 2 7 9 6 1 0 9 5 6];
A2 = [3 3 7 6 0 8 0 1 2 5];

Asize = length(A1);
matriceSize = Asize * size;

rotA1 = zeros(1, Asize);
rotA2 = zeros(1, Asize);
B1 = zeros(matriceSize, matriceSize);
B2 = zeros(matriceSize, matriceSize);

% Sestavování pseudonáhodných matic
tic
I = 1;
J = 1;
while (I <= matriceSize)
    for J = 1:matriceSize
        for k = 1:Asize
            B1(J, I+k-1) = A1(k);
            B2(J, I+k-1) = A2(k);

            if (k ~= Asize)
                incIndex = k + 1;
            else
                incIndex = 1;
            end

            rotA1(k) = A1(incIndex);
            rotA2(k) = A2(incIndex);
        end
        A1 = rotA1;
        A2 = rotA2;
    end
    I = I + Asize;
end
toc

% Základní operace s maticemi
tic
for I = 1:loops
    B = B1 + B2;
    B = B1 * B;
    B = transpose(B);
    d = det(B);
end
toc
```