Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Využití opensource prostředků Octave pro simulace zpracování signálů

Digital Signal Processing Using Opensource GNU OCTAVE

2013 **Bc. Jaromír Továrek**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Jaromír Továrek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Využití opensource prostředků OCTAVE pro simulace zpracování signálů
Digital Signal Processing Using Opensource GNU OCTAVE

Zásady pro vypracování:

Cílem práce je vytvoření uživatelské příručky pro pokročilou práci s výpočetním systémem GNU Octave s ohledem na jeho využití při výuce předmětů, zabývajících se zpracováním číslicových signálů a obrazu.

1. Popis instalace Octave na platformách MS Windows a Linux.
2. Popis instalace a konfigurace grafických rozhraní pro práci se systémem Octave.
3. Vytvoření vzorových výukových scriptů a funkcí a otestování jejich funkčnosti.

Seznam doporučené odborné literatury:

[1] Octave - Český průvodce programem. URL: http://www.octave.cz/
[2] John W. Eaton. *Gnu Octave Manual.*

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Skapa, Ph.D.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

prof. RNDr. Vladimír Vašinek, CSc.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

„Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.“

Dne: 25.4.2013

Tomáš
.....
podpis studenta

Poděkování

Rád bych poděkoval Ing. Janu Skapovi, Ph.D. za rady, připomínky a odborné vedení při vypracování mé diplomové práce.

Abstrakt

Cílem této práce je poskytnout studentům předmětů zabývajících se zpracováním signálů alternativu výpočetního softwaru, který je volně dostupný na rozdíl například od licencovaného programu Matlab. K tomuto účelu je vytvořena pokročilá uživatelská příručka práce v programu Octave, ve které jsou probírány jednotlivé kapitoly týkající se zpracování signálů. V kapitolách jsou uvedena úskalí realizace v Octave a nejdůležitější změny oproti programu Matlab. K většině kapitol jsou vytvořeny programy s podrobným komentářem, ve kterých je prakticky ukázána problematika dané kapitoly.

Klíčová slova

Octave, balíček, terminál, funkce, příkaz, m-soubor, matice, 2D graf, 3D graf, ladění zdrojového kódu, animace, filtr, přesnost

Abstract

The aim of this thesis is to offer the students of the subjects dealing with signals processing an alternative of a computing software, that is free of charge compared to i.e. the programme Matlab. For this purpose it was evolved an advanced user manual of how to work in the background of the programme Octave, in which individual chapters about signals processing are being addressed. In the chapters, both, the issue of realization in the programme Octave and the most important changes compared to the programme Matlab, are stated. For most of the chapters programmes with detailed narratives were created, in which the problematics of each chapter is practically explained.

Key words

Octave, package, terminal, function, command, m-file, matrix, 2D graph, 3D graph, debugging, animation, filter, precision

Seznam použitých symbolů

Symbol	Jednotky	Význam symbolu
A_p	-	Chyba aproximace v propustném pásmu IIR filtru v Octave
a_p	-	Koeficienty jmenovatele přenosové funkce *H(p)* v Octave
A_s	-	Chyba aproximace v závěrném pásmu IIR filtru v Octave
a_s	-	Koeficienty jmenovatele přenosové funkce *H(s)* v Octave
a_z	-	Koeficienty jmenovatele přenosové funkce *H[z]* v Octave
A_p	-	Chyba aproximace v propustném pásmu IIR filtru
a_p	-	Koeficienty jmenovatele přenosové funkce *H(p)*
A_s	-	Chyba aproximace v závěrném pásmu IIR filtru
a_s	-	Koeficienty jmenovatele přenosové funkce *H(s)*
a_z	-	Koeficienty jmenovatele přenosové funkce *H[z]*
b_p	-	Koeficienty čitatele přenosové funkce *H(p)* v Octave
b_s	-	Koeficienty čitatele přenosové funkce *H(s)* v Octave
b_z	-	Koeficienty čitatele přenosové funkce *H[z]* v Octave
b_p	-	Koeficienty čitatele přenosové funkce *H(p)*
b_s	-	Koeficienty čitatele přenosové funkce *H(s)*
b_z	-	Koeficienty čitatele přenosové funkce *H[z]*
delta_p	-	Rozkmit v propustném pásmu FIR filtru v Octave
delta_s	-	Rozkmit v závěrném pásmu FIR filtru v Octave
f	Hz	Frekvence
f_c	Hz	Mezní frekvence v Octave
f_cd	Hz	Dolní mezní frekvence v Octave
f_ch	Hz	Horní mezní frekvence v Octave
f_N	Hz	Nyquistova frekvence v Octave
f_p	Hz	Mezní frekvence propustného pásma v Octave
f_pd	Hz	Dolní mezní frekvence propustného pásma v Octave
f_ph	Hz	Horní mezní frekvence propustného pásma v Octave
f_s	Hz	Mezní frekvence závěrného pásma v Octave

f_sd	Hz	Dolní mezní frekvence závěrného pásma v Octave
f_sh	Hz	Horní mezní frekvence závěrného pásma v Octave
f_vz	Hz	Vzorkovací frekvence v Octave
f_c	Hz	Mezní frekvence
f_{cd}	Hz	Dolní mezní frekvence
f_{ch}	Hz	Horní mezní frekvence
f_N	Hz	Nyquistova frekvence
f_p	Hz	Mezní frekvence propustného pásma
f_{pd}	Hz	Dolní mezní frekvence propustného pásma
f_{ph}	Hz	Horní mezní frekvence propustného pásma
f_s	Hz	Mezní frekvence závěrného pásma
f_{sd}	Hz	Dolní mezní frekvence závěrného pásma
f_{sh}	Hz	Horní mezní frekvence závěrného pásma
f_{vz}	Hz	Vzorkovací frekvence
k_p	-	Násobný koeficient přenosové funkce *H(p)* v Octave
k_s	-	Násobný koeficient přenosové funkce *H(s)* v Octave
k_z	-	Násobný koeficient přenosové funkce *H[z]* v Octave
k_p	-	Násobný koeficient přenosové funkce *H(p)*
k_s	-	Násobný koeficient přenosové funkce *H(s)*
k_z	-	Násobný koeficient přenosové funkce *H[z]*
L	-	Vektor koeficientů impulzní odezvy
meze_f	-	Vektor frekvenčních bodů v rozsahu od 0 do 1
meze_m	-	Vektor obsahující požadované velikosti přenosu
N	-	Řád filtru
N_NDP	-	Řád normované dolní propusti v Octave
N_{NDP}	-	Řád normované dolní propusti
Omega	-	Mezní úhlová frekvence normované dolní propusti v Octave
Omega_n	-	Charakteristická frekvence normované dolní propusti v Octave
p_0	-	Nulové body přenosové funkce *H(p)* v Octave
p_infty	-	Póly přenosové funkce *H(p)* v Octave

p_0	-	Nulové body přenosové funkce *H(p)*
p_∞	-	Póly přenosové funkce *H(p)*
s_0	-	Nulové body přenosové funkce *H(s)* v Octave
s_infty	-	Póly přenosové funkce *H(s)* v Octave
s_0	-	Nulové body přenosové funkce *H(s)*
s_∞	-	Póly přenosové funkce *H(s)*
T_vz	s	Vzorkovací perioda v Octave
tilde_omega	-	Normovaná mezní úhlová frekvence v Octave
w	s^{-1}	Normovaná frekvence v Octave
w_n	s^{-1}	Normovaná charakteristická frekvence v Octave
w_nd	s^{-1}	Dolní normovaná frekvence v Octave
w_nh	s^{-1}	Horní normovaná frekvence v Octave
w_p	-	Váhová posloupnost tvořená příslušným oknem v Octave
w_p	-	Váhová posloupnost tvořená příslušným oknem
z_0	-	Nulové body přenosové funkce *H[s]* v Octave
z_infty	-	Póly přenosové funkce *H[s]* v Octave
z_0	-	Nulové body přenosové funkce *H[s]*
z_∞	-	Póly přenosové funkce *H[s]*
δ_p	-	Rozkmit v propustném pásmu FIR filtru
δ_s	-	Rozkmit v závěrném pásmu FIR filtru
ϖ	-	Normovaná mezní úhlová frekvence
Ω	-	Mezní úhlová frekvence normované dolní propusti
ω	s^{-1}	Normovaná frekvence
Ω_n	-	Charakteristická frekvence normované dolní propusti
ω_n	s^{-1}	Normovaná charakteristická frekvence
ω_{nd}	s^{-1}	Dolní normovaná frekvence
ω_{nh}	s^{-1}	Horní normovaná frekvence

Seznam použitých zkratek

Zkratka	Anglický význam	Český význam
DP	Low pass	Dolní Propust
FIR	Finite Impulse Response	Konečná impulzní odezva
GUI	Graphical User Interface	Grafické uživatelské rozhraní
HP	High pass	Horní Propust
IIR	Infinite Impulse Response	Nekonečná impulzní odezva
MS	Microsoft	Microsoft
NDP	Normalized Low-pass	Normovaná Dolní Propust
OS	Operating System	Operační Systém
PP	Band pass	Pásmová Propust
PZ	Band stop	Pásmová Zádrž
2D	Two-dimensional	Dvojrozměrný
3D	Three-dimensional	Trojrozměrný

Obsah

1 Úvod

V dnešní době se neustále zvyšují požadavky na přesnost výpočtů, simulací a jejich následnou vizualizaci. Z tohoto důvodu se k těmto požadavkům přistupuje výhradně pomocí výpočetních a vizualizačních programů. Mezi nejznámější výpočetní software patří bezesporu Matlab. Ovšem jeho největší nevýhoda spočívá v tom, že je licencovaný a tudíž není k dispozici zdarma. Z tohoto důvodu vznikl výpočetní software Octave, který byl sice původně určen pouze pro výuku návrhu chemických reaktorů, ale postupným vývojem, který zahrnuje vznik grafických rozhraní, nebo možnost doinstalování rozšiřujících balíčků se stal Octave komplexním výpočetním softwarem, který lze využít v mnoha oblastech zpracování signálů. Octave zatím nedosáhl a ani snad nedosáhne kvalit Matlabu, ale jeho největší výhoda spočívá v tom, že je volně k dispozici všem uživatelům.

Cílem mé práce je seznámit uživatele s tímto programem a jeho možnostmi zpracování signálů a poskytnout jim tak alternativu k výpočetnímu softwaru Matlab. V první části práce popisuji instalaci Octave a rozšiřujících balíčků na platformách MS Windows a Linux. V druhé části se zabývám instalací dvou základních grafických rozhraní (QtOctave a GUI Octave), která nám umožňují přehlednější a snadnější práci s programem. Instalace je opět popsána pro platformy MS Windows a Linux. Poslední část je rozdělena do osmi kapitol. V jednotlivých kapitolách se zabývám zpracováním signálů a jejich vizualizací. První kapitola je zaměřena na základní práci s maticemi. V druhé a třetí kapitole popisuji možnosti vykreslení a nastavení grafů. Čtvrtá kapitola je zaměřena na problematiku ladění zdrojového kódu v Octave. Popisuji zde možnost ladění zdrojového kódu jak pomocí grafických rozhraní tak přímo zadáváním příkazů do terminálového okna. V páté kapitole se okrajově zabývám prací se symbolickou proměnou. Šestá kapitola je zaměřena na tvorbu animací. V sedmé kapitole popisuji návrh digitálních číslicových filtrů. Popisuji jak návrh filtrů s konečnou impulzní odezvou, tak filtrů s nekonečnou impulzní odezvou. V poslední osmé kapitole se zabývám prací v programu s definovanou přesností. K většině kapitol jsou vytvořeny programy s podrobným komentářem, ve kterých je prakticky ukázáno řešení dané problematiky.

2 Instalace GNU Octave

Program Octave vznikl jako open-source. Je tedy volně šiřitelný, a proto si jej můžeme kdekoliv stáhnout například z domovské stránky projektu [1]. Na této stránce nalezneme veškeré dostupné verze programu včetně variant pro různé platformy, jako jsou například Linux, Mac OS či MS Windows. Já se v této práci budu zabývat instalací pod systémy MS Windows a Linux (distribuce Ubuntu). Instalace programu na ostatní platformy je detailně popsaná na domovské stránce projektu. Nové verze programu se objevují vícekrát za rok, takže nelze zaručit aktuálnost popisované verze, ale z vlastních zkušeností vím, že se instalace jednotlivých verzí příliš neliší. Nyní je aktuální verze Octave 3.6.1, takže si probereme instalaci právě této verze.

2.1 Instalace GNU Octave pod systémem MS Windows v krocích

a) Stažení aktuální verze programu

Aktuální verzi programu můžeme stáhnout z domovské stránky projektu [2]. Na této adrese vybereme možnost Windows: installers, kde se nachází dostupné verze programu. Po vybrání aktuální verze stáhneme dva komprimované soubory *Octave3.6.1_gcc4.6.2_pkgs_20120303.7z* a *Octave3.6.1_gcc4.6.2_20120303.7z*.

b) Vytvoření složky s programem Octave a samotná instalace

Na místním disku si vytvoříme složku, kam chceme program umístit, například to bude složka Octave. Cesta k ní bude vypadat takto C:\Octave. Do takto vytvořené složky rozbalíme soubor *Octave3.6.1_gcc4.6.2_20120303.7z*, to můžeme provést pomocí programu 7-zip. Nyní je Octave nainstalovaný a je ještě potřeba doinstaloval příslušné balíčky. Opět to provedeme tak, že soubor *Octave3.6.1_gcc4.6.2_pkgs_20120303.7z* rozbalíme do námi vytvořené složky C:\Octave.

c) Spuštění programu

Program se spouští ze složky C:\Octave\Octave3.6.1_gcc4.6.2\bin pomocí aplikace octave.exe. Přes pravé tlačítko myši si také můžeme odeslat zástupce aplikace na plochu a později program spouštět právě z plochy.

d) Načtení nainstalovaných balíčků

Abychom nemuseli po každém spuštění programu balíčky načítat, můžeme při prvním spuštění programu zadat následující tři příkazy a balíčky se vždy po spuštění načtou automaticky.

```
pkg rebuild -auto
pkg rebuild -noauto ad windows
pkg rebuild -auto java
```

e) Instalace jednotlivých balíčků

Jaké balíčky aktuální verze obsahuje, je možné zjistit na adrese, odkud jsme program stáhli, nebo po instalaci balíčků stačí v programu zadat příkaz `pkg list` a na obrazovce se objeví výpis nainstalovaných balíčků. Pokud chceme nainstalovat novější verzi balíčku je opět potřeba příslušný balíček stáhnout z domovské stránky projektu [2], kde vybereme v horní liště záložku Packages. Balíček uložíme do našeho domovského adresáře C:\Octave a po spuštění Octave zadáme následující příkazy.

```
pkg install <jméno_balíčku>
pkg load < jméno_balíčku >
```

Pokud chceme balíček odinstalovat, stačí zadat.

```
pkg uninstall < jméno_balíčku >
```

f) Příklad instalace balíčku „symbolic“

Nejprve stáhneme soubor *symbolic-1.1.0.tar.gz*, který umístíme do adresáře C:\Octave. Po spuštění programu zadáme následující příkazy.

```
pkg install symbolic-1.1.0.tar.gz
pkg load symbolic
```

Nyní už můžeme s tímto balíčkem pracovat.

Pozn. Při zadávání příkazů se musíme nacházet ve složce, kam jsme umístili soubor *symbolic-1.1.0.tar.gz*, jinak instalaci neprovedeme. Proto se mi osvědčilo instalování balíčků z grafického rozhraní, kde je přímo vidět místo, kde se zrovna nacházíme.

Při inicializaci může program hlásit tato varování:

warning: gmsh does not seem to be present some functionalities will be disabled

warning: dx does not seem to be present some functionalities will be disabled

warning: function C:\Octave\Octave3.6.1_gcc4.6.2\share\octave\packages\statistics-1.1.0\fstat.m shadows a core library function

Tato varování můžeme ignorovat. Jejich přesný popis je na adrese projektu.

2.2 Instalace GNU Octave pod systémem Linux (distribuce Ubuntu) v krocích

a) Aktualizace balíčků Ubuntu

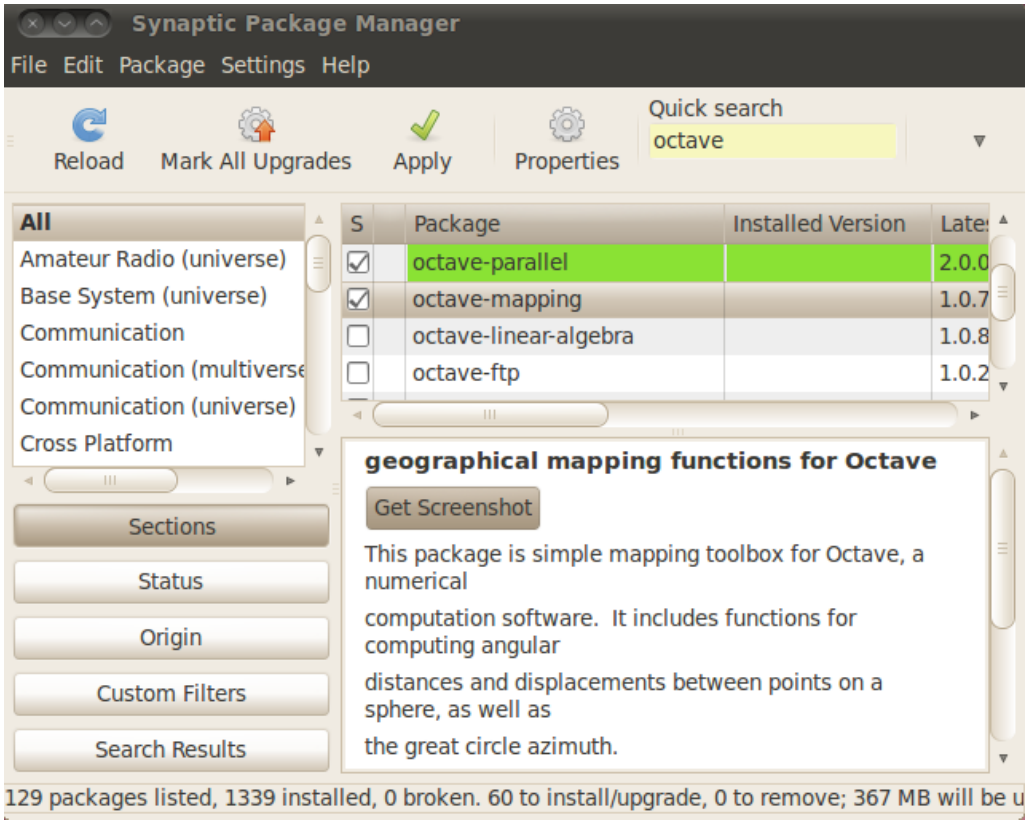
Po přihlášení do terminálu jako správce (root) zadáme příkaz `apt-get update`, pomocí kterého se aktualizují všechny příslušné balíčky distribuce Ubuntu, které máme k dispozici včetně balíčku Octave. Tím zajistíme aktuální dostupnou verzi.

b) Instalace programu Octave

Instalaci programu Octave můžeme provést dvěma způsoby. První způsob je přes terminál zadáním příkazu `apt-get install octave`, druhý způsob je pomocí správce balíčků Synaptic (Synaptic package manager), který nalezneme v sekci systém – administration - Synaptic package manager. Ve správci balíčků necháme vyhledat balíčky pro Octave a poté vybereme příslušnou verzi, kterou chceme nainstalovat. Instalaci provedeme stisknutím tlačítka apply. Ukázka je na obrázku 2.1.

c) Instalace jednotlivých balíčků

Opět můžeme provést instalaci dvěma způsoby jako v předchozím případě. Pomocí terminálu si nejprve můžeme zobrazit dostupné balíčky, to provedeme zadáním příkazu `aptitude search octave`. Samotnou instalaci provedeme příkazem `aptitude install název_balíčku`. V případě instalace pomocí správce balíčků Synaptic postupujeme obdobně jako u instalace samotného programu Octave viz obrázek 2.1. Doporučuji nainstalovat všechny dostupné balíčky k patřičné verzi programu.



Obrázek 2.1: *Ukázka instalace balíčků Octave pomocí správce balíčků Synaptic*

Při zpracování této kapitoly jsem čerpal z literatury [1], [2], [3].

3 Instalace grafických rozhraní

Hlavním úkolem grafických rozhraní je usnadnit práci uživateli s programem Octave. Grafických rozhraní pro Octave se začíná objevovat celá řada (QtOctave, GUI Octave, Xoctave, QOcTerm atd.). Já jsem zvolil popis dvou nejrozšířenějších a to QtOctave a GUI Octave. Zatímco GUI Octave lze nainstalovat pouze pod systémem Windows, tak QtOctave je dostupný i pro systém Linux. Já zde budu popisovat instalaci QtOctave pod systémem Windows a Linux a instalaci GUI Octave pod systémem Windows. Abychom mohli grafická rozhraní využívat, je potřeba mít nainstalovaný Octave.

3.1 Instalace QtOctave pod systémem MS Windows v krocích

a) Stažení programu

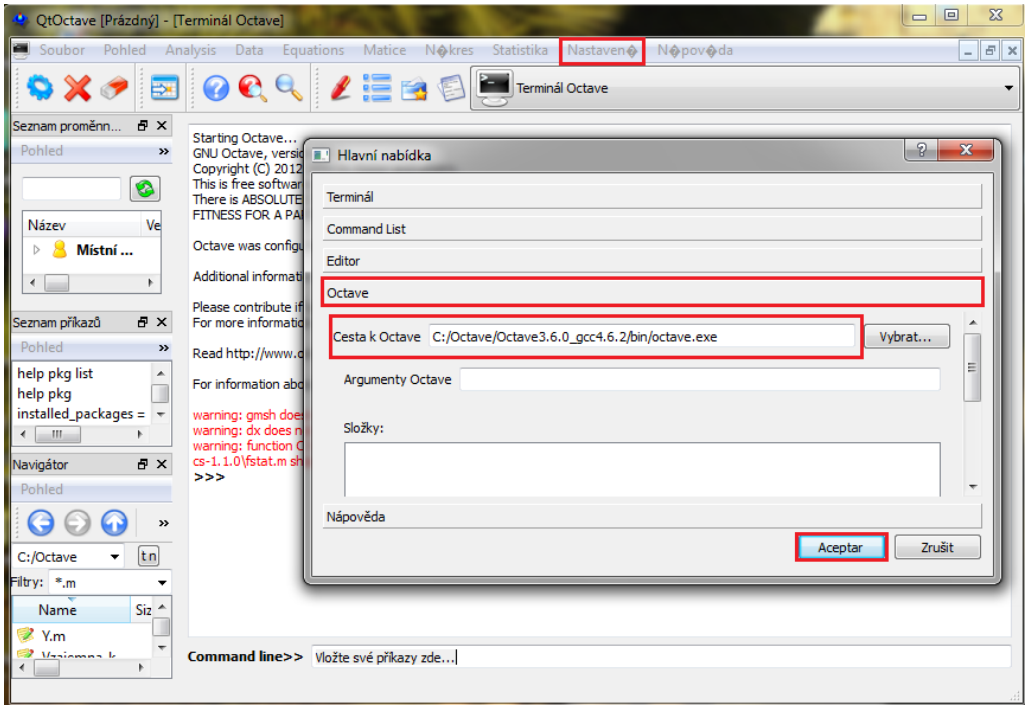
Program si můžeme stáhnout z domovské stránky projektu [4]. Bohužel novějších verzí už se nedočkáme, protože vývoj QtOctave byl ukončen. Stránky projektu jsou v současné době mimo provoz, proto je instalační soubor obsažen v příloze B na dvd.

b) Instalace programu

Pro instalaci využijeme již vytvořenou složku pro samotný Octave, tedy složku C:\Octave. Samozřejmě je možné si vytvořit samostatnou složku, ale já jsem zvolil stejnou jako pro Octave, abych měl oba programy pohromadě. Do složky C:\Octave rozbalíme stažený soubor. To provedeme stejně jako při instalaci Octave. Nyní už je rozhraní nainstalováno a je už jen potřeba ho správně nastavit.

c) Spuštění programu QtOctave a jeho nastavení

Program spustíme ze složky C:\Octave\qtoctave-0.10.1\bin pomocí aplikace qtoctave.exe. Opět je dobré vytvořit si zástupce na ploše pro snadnější spouštění. Aby rozhraní správně pracovalo, je potřeba nastavit cestu k programu Octave. To provedeme přes tlačítko Nastavení v horní liště programu. Poté zvolíme možnost Obecná nastavení, záložku Octave a zde nastavíme cestu k aplikaci octave.exe C:\Octave\Octave3.6.1_gcc4.6.2\bin\octave.exe. Na závěr potvrdíme nastavení tlačítkem Aceptar. Aby se změna nastavení projevila, je potřeba program restartovat. Postup je také zobrazen na obrázku 3.1.



Obrázek 3.1: *Nastavení rozhraní QtOctave*

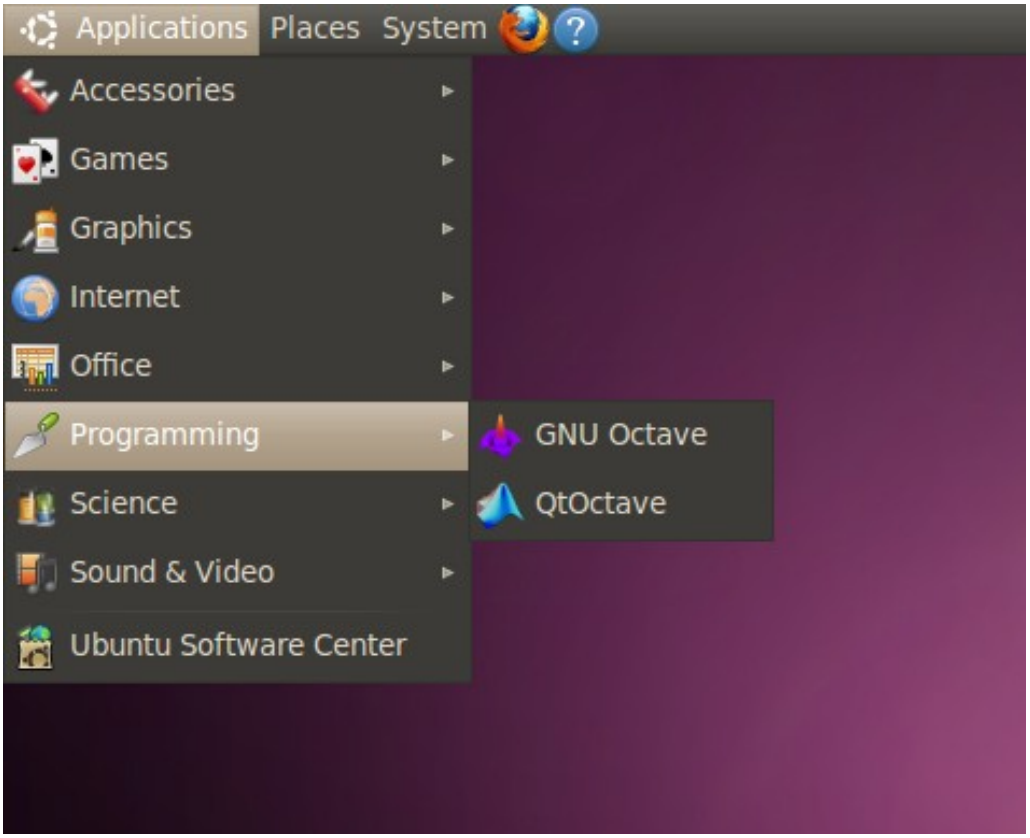
3.2 Instalace QtOctave pod systémem Linux (distribuce Ubuntu) v krocích

a) Instalace programu

Jako při instalaci čistého Octave můžeme postupovat dvěma způsoby. První způsob je instalace přes terminál. V tomto případě stačí zadat příkaz `apt-get install qtoctave`. Po provedení tohoto příkazu je program nainstalován. Druhý způsob je pomocí správce balíčku Synaptic. Postup instalace je stejný jako pro instalaci Octave nebo instalaci balíčků. Vyhledáme balíček qtoctave a poté spustíme instalaci tlačítkem apply.

b) Spuštění programu QtOctave a jeho nastavení

Program můžeme spustit přes terminál zadáním příkazu `qtoctave`, nebo můžeme program spustit z pracovní plochy aplikace – programování – QtOctave viz obrázek 3.2. Nastavení správné funkčnosti je stejné jako u nastavení ve Windows, pouze se bude lišit cesta k Octave. Nastavení je zobrazeno na obrázku 3.1.



Obrázek 3.2: *Spuštění QtOctave z pracovní plochy*

3.3 Instalace GUI Octave v krocích

a) Stažení aktuální verze programu

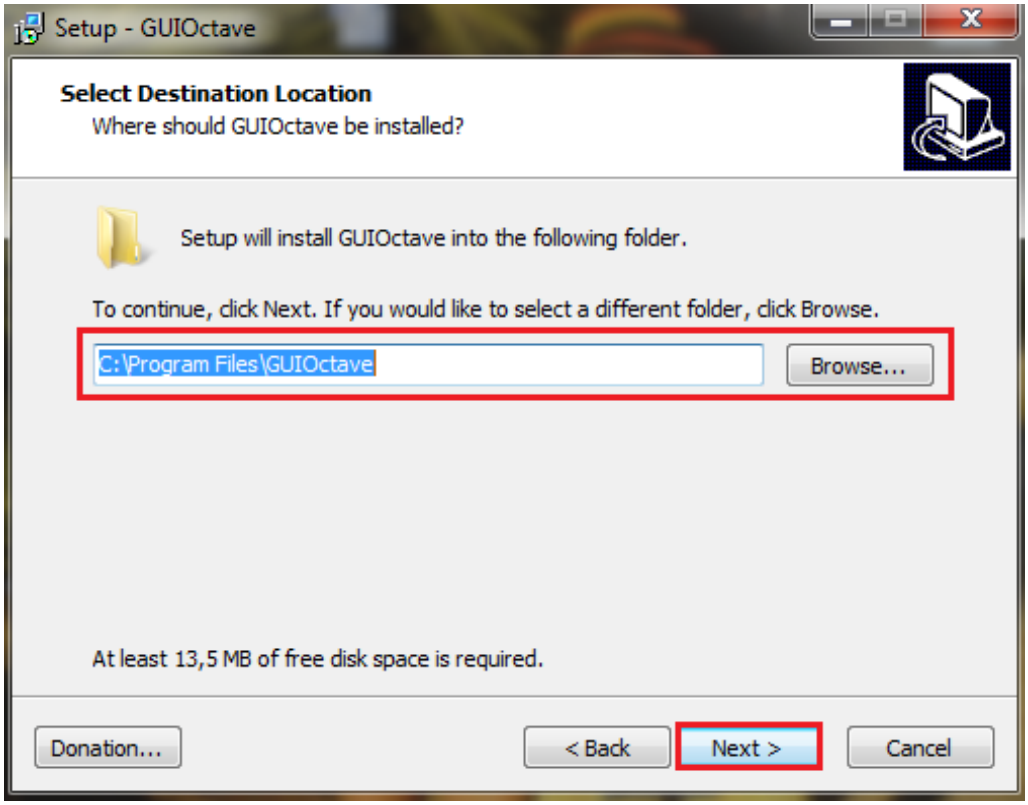
Aktuální verzi programu si stáhneme například z domovské stránky projektu [5]. V současné době není domovská stránka projektu dostupná, z tohoto důvodu je instalační soubor opět umístěn v příloze B na dvd.

b) Instalace programu

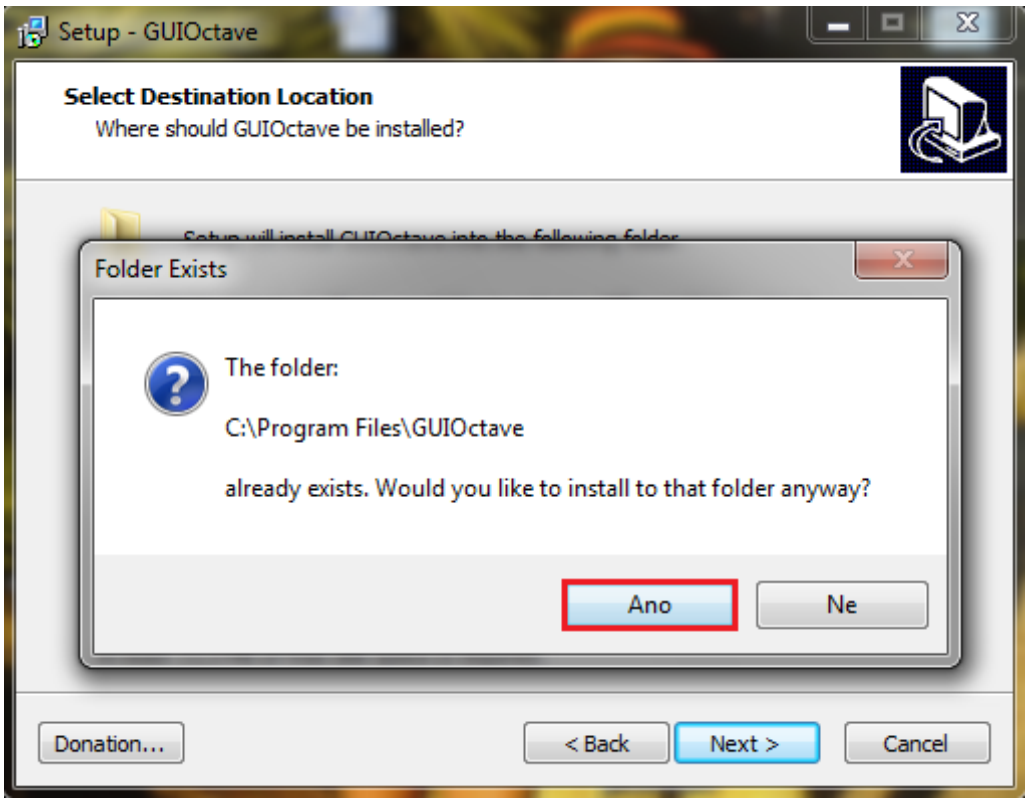
Instalaci programu zahájíme spuštěním stažené aplikace. Po spuštění aplikace se nám otevře úvodní instalační okno, které potvrdíme tlačítkem Next, viz obrázek 3.3. Dále je potřeba potvrdit licenční podmínky, viz obrázek 3.4. V dalším kroku vybereme složku, kam chceme program umístit, jak je vidět na obrázku 3.5. Pokud jsme si složku nevytvořili předem, je potřeba provést potvrzení podle obrázku 3.6. Následně si můžeme vybrat typ instalace, já jsem zvolil plnou instalaci, viz obrázek 3.7. Dalším krokem vybereme název složky, která bude umístěna ve start menu, tento postup je na obrázku 3.8. V kroku sedm volíme, jestli chceme vytvořit spouštěcí ikony, viz obrázek 3.9. Následně se nám zobrazí souhrn nastavení instalace, který potvrdíme jako na obrázku 3.10. Po dokončení instalace už jen potvrdíme úspěšné nainstalování tlačítkem Finish, viz obrázek 3.11.

Obrázek 3.3: *Instalace GUI Octave – krok 1*

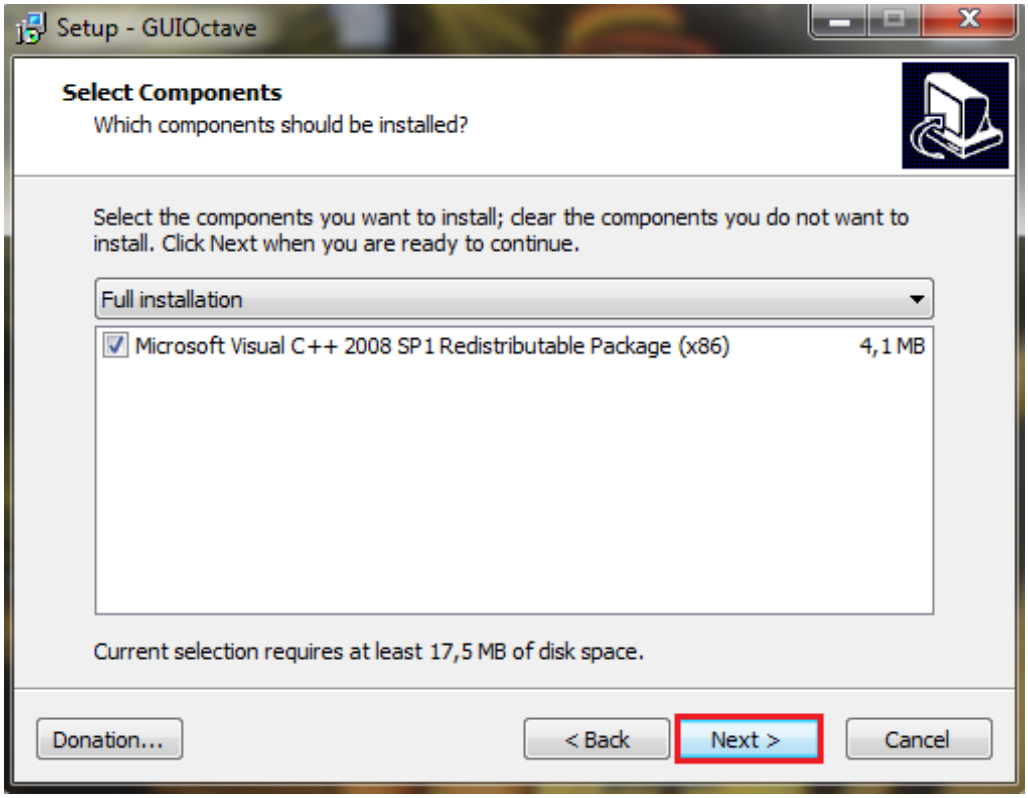
Obrázek 3.4: *Instalace GUI Octave – krok 2*



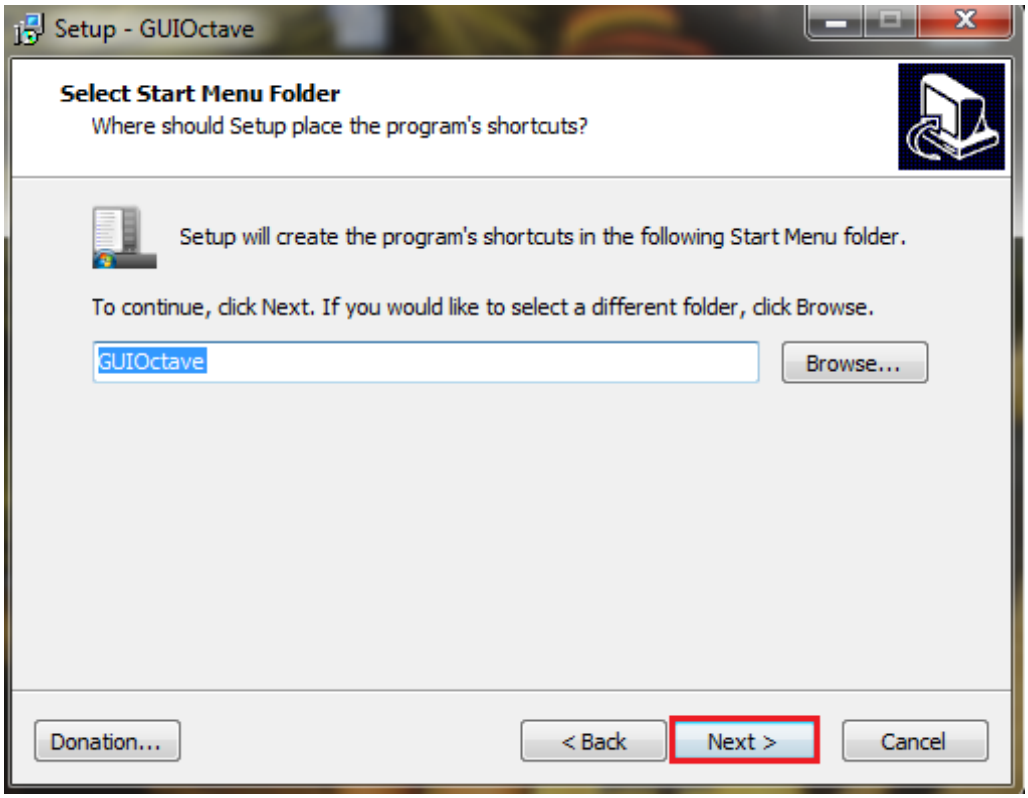
Obrázek 3.5: *Instalace GUI Octave – krok 3*



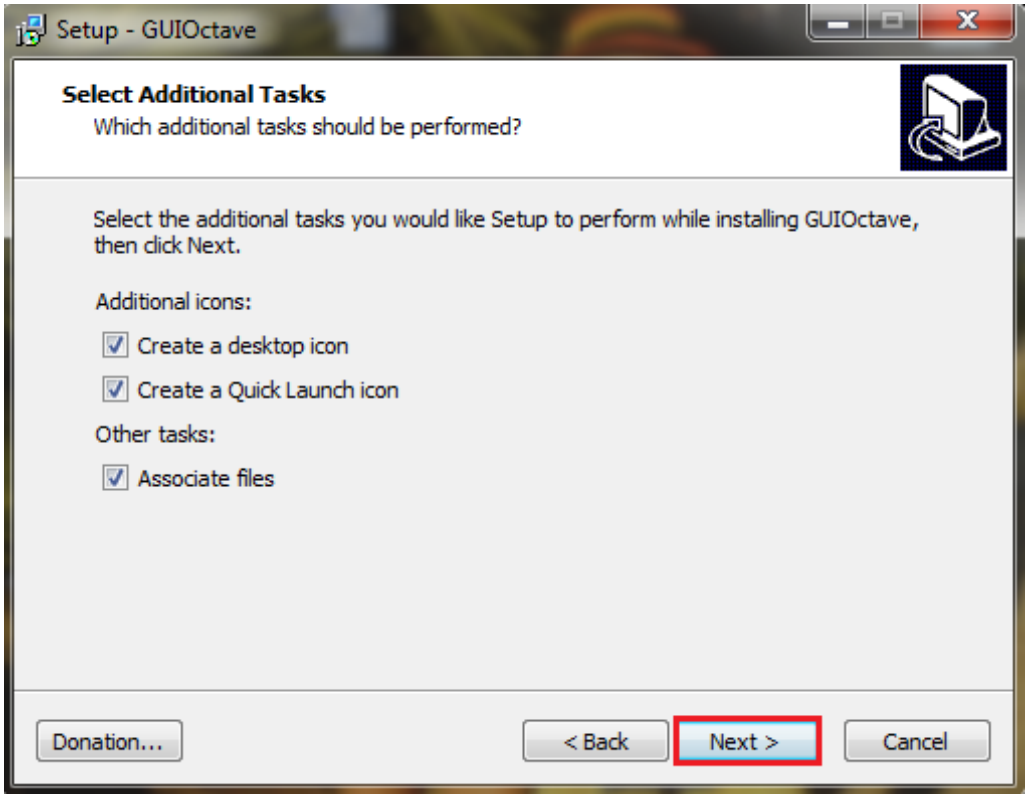
Obrázek 3.6: *Instalace GUI Octave – krok 4*



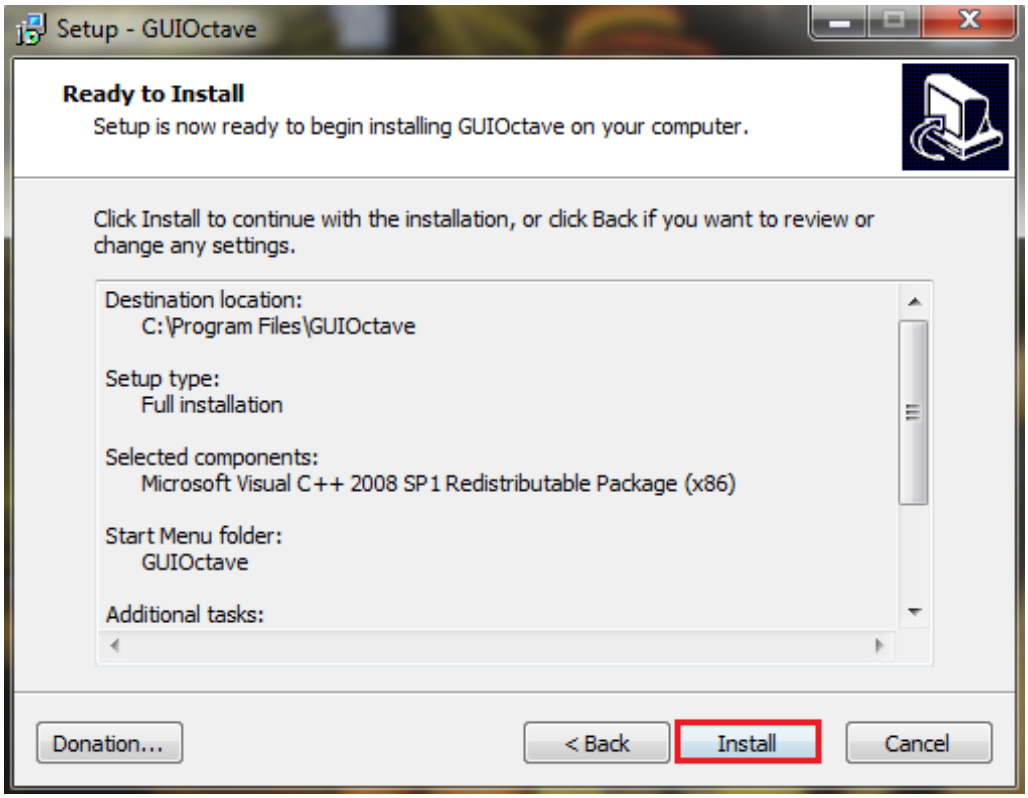
Obrázek 3.7: *Instalace GUI Octave – krok 5*



Obrázek 3.8: *Instalace GUI Octave – krok 6*



Obrázek 3.9: *Instalace GUI Octave – krok 7*

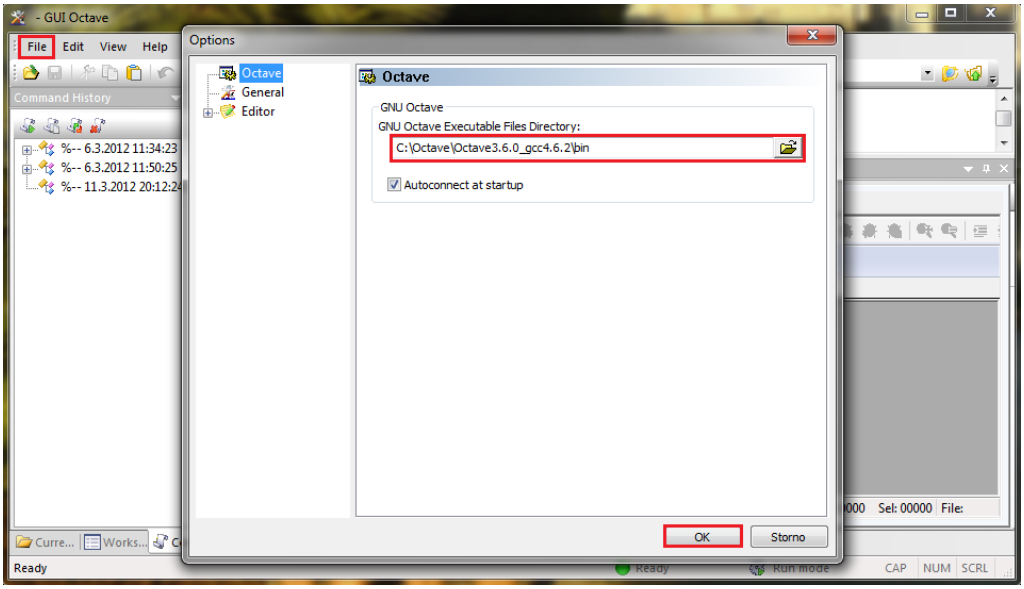


Obrázek 3.10: *Instalace GUI Octave – krok 8*

Obrázek 3.11: *Instalace GUI Octave – krok 9*

c) Nastavení GUI Octave

Po spuštění programu je opět potřeba nastavit cestu programu Octave, jako je tomu u QtOctave. Nastavení provedeme přes záložku File, kde zvolíme možnost Options. Následně nastavíme cestu k aplikaci octave.exe. V našem případě je to složka C:\Octave\Octave3.6.1_gcc4.6.2\bin. Způsob nastavení je zobrazen na obrázku 3.12. Na závěr program restartujeme a můžeme začít pracovat.



Obrázek 3.12: *Nastavení rozhraní GUI Octave*

Při zpracování této kapitoly jsem čerpal z literatury [4], [5].

4 Základy práce s maticemi v Octave

Program Octave je stejně jako Matlab orientován maticově. To znamená, že základním objektem je matice. Proto si ukážeme jak pracovat s maticemi a jaké funkce pro práci s maticemi Octave nabízí. Pro základní práci s maticemi není potřeba instalovat nějaký rozšiřující balíček.

4.1 Seznámení s maticemi

Jako demonstraci toho, že je Octave opravdu maticově orientován stačí do terminálu zapsat `A=2`. Poté použijeme příkaz `size(A)`, jehož výsledek nám udává velikost matice (v tomto případě se jedná o matici s jedním řádkem a jedním sloupcem). Matici můžeme libovolně rozšiřovat, pokud chceme na určitou pozici v matici vložit číslo, tak postupujeme následovně. Zadáme, o jakou matici se jedná, na jakou pozici chceme číslo vložit a jaké číslo to je, např. `A(4,2)=4`. V našem případě se jedná o matici A, kde na čtvrtý řádek v druhém sloupci vložíme číslo 4. Výsledek je vidět na obrázku 4.1. Pokud chceme nějaké číslo vložit na celý řádek, nebo sloupec použijeme místo hodnoty řádku (sloupce) dvojtečku, např. `A(4,:)=5`. V matici A se na celý čtvrtý řádek vloží pětky. Pro odstranění řádku (sloupce) zadáme `A(4,:)=[]`. V matici A odstraníme celý čtvrtý řádek. Pomocí příkazu `whos(A)` zjistíme základní informace o matici A (název, velikost, typ, …). Po vytvoření je matice automaticky typu double, pokud chceme tento typ změnit, zadáme např. `A=int8(A)`. Tím změníme typ double na 8 bitový integer. Nyní můžeme do matice ukládat pouze celá čísla, pokud bychom chtěli vložit číslo desetinné, tak se číslo zaokrouhlí a bude do matice vložena pouze celá část.

```
>> A=2
A =  2
>> size(A)
ans =

   1   1

>> A(4,2)=4
A =

   2   0
   0   0
   0   0
   0   4

>> whos A
Variables in the current scope:

   Attr Name        Size                     Bytes  Class
   ==== ====        ====                     =====  =====
        A           4x2                         64  double

Total is 8 elements using 64 bytes
```

Obrázek 4.1: *Ukázka základních příkazů pro práci s maticemi*

4.2 Vytvoření vektorů a matic

Nejjednodušší způsob vytvoření vektoru je, že zadáme `t=0:5`. Tím vytvoříme vektor s hodnotami 0 1 2 3 4 5. Pokud chceme vektor, který nebude složen pouze z celých čísel, můžeme zadat krok, s jakým se má zvětšovat `t=0:0.5:5`, vektor pak bude obsahovat hodnoty od 0 do 5 s krokem 0.5. Další možností vytvoření vektoru je `t=[1 6 7 8 0]`, vektor pak obsahuje přímo námi zadané hodnoty 1 6 7 8 0. Z tohoto zápisu také vychází vytvoření matice, protože vektor je vlastně matice s jedním řádkem a *n* sloupci. K oddělení jednotlivých řádků používáme středník. Matici A tedy můžeme vytvořit takto `A=[1,2,3;4 5 6]`. Matice A bude obsahovat dva řádky a tři sloupce. Abychom oddělili jednotlivé sloupce, můžeme buď použít čárku (1,2,3), nebo stačí mezera (4 5 6). Příklady jsou uvedeny na obrázku 4.2.

```
>> t=0:5
t =

   0   1   2   3   4   5

>> t=0:0.5:5
t =

 Columns 1 through 7:

    0.00000    0.50000    1.00000    1.50000    2.00000    2.50000    3.00000

 Columns 8 through 11:

    3.50000    4.00000    4.50000    5.00000

>> A=[1,2,3;4 5 6]
A =

   1   2   3
   4   5   6
```

Obrázek 4.2: *Vytvoření vektorů a matice*

4.3 Základní operace s maticemi a maticové funkce

V Octave můžeme využívat základní operace s maticemi, na které jsme zvyklí z lineární algebry (součet matic, rozdíl matic, násobení matic a podíl matic). Octave navíc obsahuje řadu funkcí, které jsou definovány nad jednou maticí. Tyto funkce slouží ke značnému ulehčení práce (výpočet determinantu).

4.3.1 Operace s maticemi

Sčítání matic – abychom mohli matice sčítat, tak musí být obě matice stejné dimenze (stejný počet řádků a sloupců). `A=[1 2;3 4]; B= [5 6;7 8]; C= A+B`, součet se provede mezi stejnolehlými prvky matic A a B. Výsledek tedy je `C=[6 8;10 12]`. Příklad je uveden na obrázku 4.3.

Rozdíl matic – opět platí, že matice musí být stejné dimenze. `D=A-B`, rozdíl se provede mezi stejnolehlými prvky matic A a B. Tedy `D=[-4 -4; -4 -4]`. Příklad je uveden na obrázku 4.3.

```
>> A=[1 2;3 4];

>> B=[5 6;7 8];

>> C=A+B
C =

    6    8
   10   12

>> D=A-B
D =

   -4   -4
   -4   -4
```

Obrázek 4.3: *Součet a rozdíl matic*

Součin matic – abychom mohli provést násobení `E=A*B` musí mít matice B stejný počet řádků jako má matice A sloupců. Příklad je uveden na obrázku 4.4.

Součin matic po prvcích – zde platí stejná podmínka jako u součtu a rozdílu, matice musí být stejné dimenze. `F=A.*B` součin se provede mezi stejnolehlými prvky matic A a B. Příklad je uveden na obrázku 4.4.

```
>> A=[1 2;3 4];

>> B=[5 6;7 8];

>> E=A*B
E =

   19   22
   43   50

>> F=A.*B
F =

    5   12
   21   32
```

Obrázek 4.4: *Součin matic a součin matic po prvcích*

Dělení matic zprava – `G=A/B` můžeme upravit na násobení `A*inv(B)`, funkce `inv` vytvoří inverzní matici, poté už platí totéž pro násobení. Příklad je uveden na obrázku 4.5.

Podíl stejnolehlých prvků matic A a B – `H=A./B` matice musí být stejné dimenze. Příklad je uveden na obrázku 4.5.

```
>> A=[1 2;3 4];

>> B=[5 6;7 8];

>> G=A/B
G =

   3.00000  -2.00000
   2.00000  -1.00000

>> H=A./B
H =

   0.20000   0.33333
   0.42857   0.50000
```

Obrázek 4.5: *Pravostranné dělení a pravostranné dělení po prvcích*

Dělení matic zleva – opět můžeme převést `I=A\B` na násobení `inv(A)*B`. Příklad je uveden na obrázku 4.6.

Podíl stejnolehlých prvků matic B a A – `J=A.\B` matice musí být stejné dimenze. Příklad je uveden na obrázku 4.6.

```
>> A=[1 2;3 4];

>> B=[5 6;7 8];

>> I=A\B
I =

  -3.0000  -4.0000
   4.0000   5.0000

>> J=A.\B
J =

   5.0000   3.0000
   2.3333   2.0000
```

Obrázek 4.6: *Levostranné dělení a levostranné dělení po prvcích*

4.3.2 Maticové funkce

Abychom mohli používat maticové funkce, je potřeba mít nějakou matici nadefinovanou. Poté už pouze stačí vybrat funkci, kterou chceme s maticí provést.

`inv(A)` – funkce provede inverzi čtvercové matice, inverzi můžeme také provést pomocí příkazu `A^-1`.

`A'` – transpozice matice A, dojde k výměně řádků a sloupců.

`det(A)` – výpočet determinantu matice A, matice musí být čtvercová.

`diag(A)` – vrací vektor prvků nacházejících se na hlavní diagonále.

`size(A)` – vrací velikost matice (počet řádků a sloupců).

`triu(A)` – vrací dolní trojúhelníkovou matici.

`tril(A)` – vrací horní trojúhelníkovou matici.

`rank(A)` – vrací hodnost matice A.

`rot90(A,-1)` – dojde k otočení matice o 90°, je-li druhý parametr -1 dojde k otočení ve směru hodinových ručiček, pokud je 1 otočí se proti směru hodinových ručiček.

`zeros(2,3)` – vytvoření nulové matice o velikosti 2 řádky a 3 sloupce.

`ones(2,2)` – vytvoření jednotkové matice o velikosti 2 řádky a 2 sloupce.

`eye(3,3)` – vytvoření diagonální matice.

`rand(3,4)` – vytvoření matice náhodných čísel z intervalu <0,1>.

`randn(3,4)` – vytvoření matice pseudonáhodných čísel s normálním rozdělením.

Příklady použitých funkcí a příkazů v této kapitole jsou uvedeny v příloze C na dvd v m-souboru *zaklady_prace_s_maticemi_v_Octave.m*.

Při zpracování této kapitoly jsem čerpal z literatury [1], [6].

5 2D grafy

Pomocí 2D grafů můžeme jednoduše provést vizualizaci závislosti jedné veličiny na druhé (funkce jedné proměnné). Pro práci s dvojrozměrnými grafy obsahuje Octave řadu implementovaných funkcí, které nám umožňují snadnější práci. Základní funkce pro tvorbu 2D grafu, jeho popisu a nastavení jsou podrobně probrány v následujících podkapitolách. Při používání grafického rozhraní QtOctave pod sytémem MS Windows je potřeba v grafickém rozhraní nastavit grafickou sadu pomocí příkazu `graphics_toolkit gnuplot` jinak nedojde ke korektnímu zobrazení grafu. V grafickém rozhraní GUI Octave je tato sada nastavena automaticky. Funkce používané v Octave pro tvorbu grafů jsou obsaženy v základním instalačním souboru, proto není nutné instalovat další rozšiřující balíček.

5.1 Tvorba 2D grafu

Než začneme se samotným vykreslováním 2D grafu, je nejprve nutné nadefinovat signál, který chceme zobrazit. Signál se skládá ze dvou vektorů. Jeden vektor tvoří nezávisle proměnnou (osa x) a druhý závisle proměnnou (osa y). V našem případě si zvolíme jako nezávisle proměnnou vektor t a jako závisle proměnnou signál x.

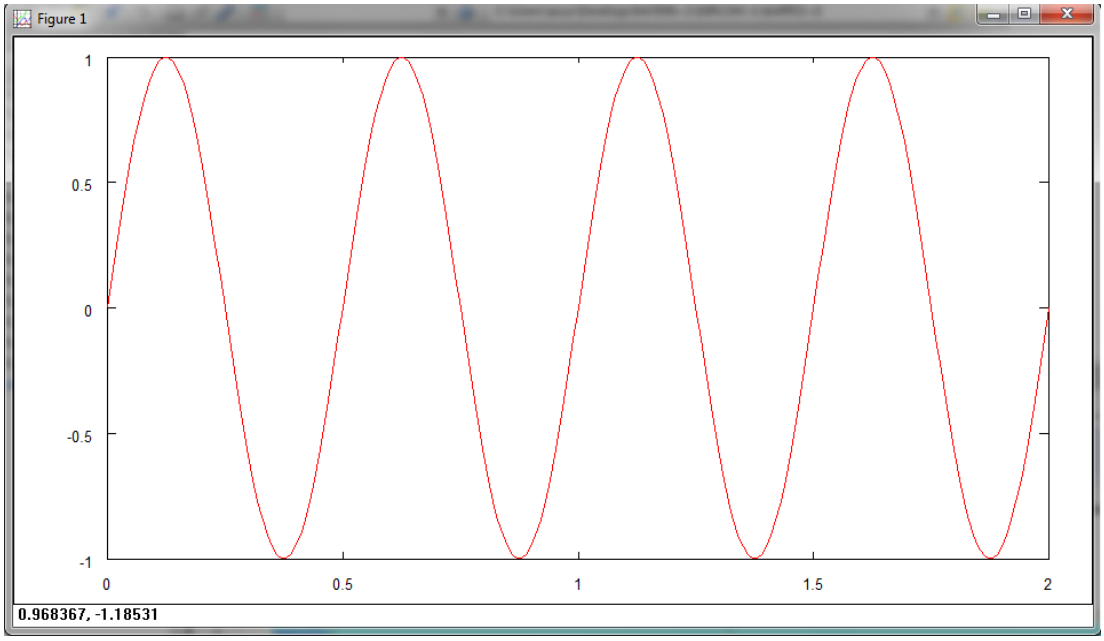
`t=0:0.01:2` – vektor času t od 0 do 2 s krokem 0.01.

`f=2` – frekvence f=2Hz.

`x=sin(2*pi*f*t)` – vytvoření signálu x.

K zobrazení signálu použijeme příkaz `plot(x,y)`.

`plot(t,x,'r')` – zobrazení signálu x v závislosti na vektoru t, zadáním řetězce 'r' nastavíme barvu grafu na červenou. Další možná nastavení typu a barvy čar si můžete nechat zobrazit v terminálovém okně pomocí příkazu `help plot`. Graf je zobrazen na obrázku 5.1.



Obrázek 5.1: *Zobrazení signálu x v závislosti na vektoru t*

5.2 Popis grafu

Součástí každého správného grafu je jeho popis. Pro popis grafu slouží následující příkazy:

`xlabel('t [s]')` – popis osy x. Pokud bychom chtěli u popisku dolní index, tak stačí zadat `t_{1}`. Složenou závorku používáme pouze v případě, že index obsahuje více znaků.

`ylabel('U [V]', 'FontSize',12)` – popis osy y. Nastavení velikosti písmen popisu na 12.

`title('Zavislost U na t')` – název grafu.

`text(2,0,'bod_1')` – vložení textu do grafu. Text bod_1 se vloží na pozici x=2 a y=0.

`gtext('bod-klik')` – vložení textu do grafu klikem myši. Text bod-klik se vloží do grafu na místo, kam „klikneme“ myší.

`legend('sin(2*pi*f*t)','cos(2*pi*t)')` – vložení legendy do grafu. Příklad je uveden na obrázku 5.3.

Pomocí tak zvaného interpreteru můžeme definovat, jak má být text vykreslen. K dispozici máme dva druhy interpretace. První způsob je, že text bude vykreslen standardně (bez speciálních znaků), pro tuto variantu je třeba nastavit interpreter na hodnotu `'none'`, zápis poté bude vypadat například takto `text(2,0,'bod_1','Interpreter','none')`. Popis grafů můžeme také provést

pomocí TeXové notace, která umožňuje popisek doplnit o speciální znaky (řecká písmena, horní a dolní index atd.), nebo například změnu fontu, interpreter je potom nastaven na hodnotu `'tex'`, tato hodnota je nastavena jako výchozí, a proto není nutné samotné nastavení provádět. Pokud chceme použít TeXový znak, tak máme dvě možnosti. První možnost je, že před znak, který chceme použít, umístíme zpětné lomítko např. `\Delta`. Druhá možnost je, že zápis znaku provedeme pomocí následující notace `{/Symbol d}`, tento způsob lze využít u všech možných popisků, na rozdíl od prvního způsobu, kde nemusí být zobrazení znaku vždy korektní (například při nastavení osy pomocí funkce `set` nedojde v příkazu `set(gca,'YTickLabel',['\Delta'])` k zobrazení Δ, při použití druhého způsobu je zobrazení korektní). Princip činnosti funkce `set` je popsán v kapitole 5.3. Značky jednotlivých řeckých písmen pro druhý způsob zápisu jsou uvedeny v tabulce 5.1. Při použití velkých řeckých písmen stačí zadat značku velkým písmenem. Standardní popis os (`xlabel`, `ylabel`, `title`) můžeme bez problémů provést oběma způsoby, výsledný efekt bude stejný. Ukázka popisku os pomocí TeXové notace je vidět na obrázku 5.2. Interpreter pro LaTeXový zápis zatím není v Octave implementován jako je tomu u programu Matlab.

Tabulka malých řeckých písmen		
Řecká písmena	Znak v Octave	Zápis písmena v Octave
α	a	{/Symbol a}
β	b	{/Symbol b}
γ	g	{/Symbol g}
δ	d	{/Symbol d}
ε	e	{/Symbol e}
ζ	z	{/Symbol z}
η	h	{/Symbol h}
θ	q	{/Symbol q}
λ	l	{/Symbol l}
μ	m	{/Symbol m}
ν	n	{/Symbol n}
ξ	x	{/Symbol x}
π	p	{/Symbol p}
ρ	r	{/Symbol r}
τ	t	{/Symbol t}
φ	f	{/Symbol f}
χ	c	{/Symbol c}
ψ	y	{/Symbol y}
ω	w	{/Symbol w}

Tabulka 5.1: *Tabulka malých řeckých písmen*

Popis os a názvu grafu pomocí prvního způsobu:

`xlabel('\tau [s]')` – popis osy x pomocí prvního způsobu TeXové notace.

`ylabel('\Delta U_{1} [V]')` – popis osy y pomocí prvního způsobu TeXové.

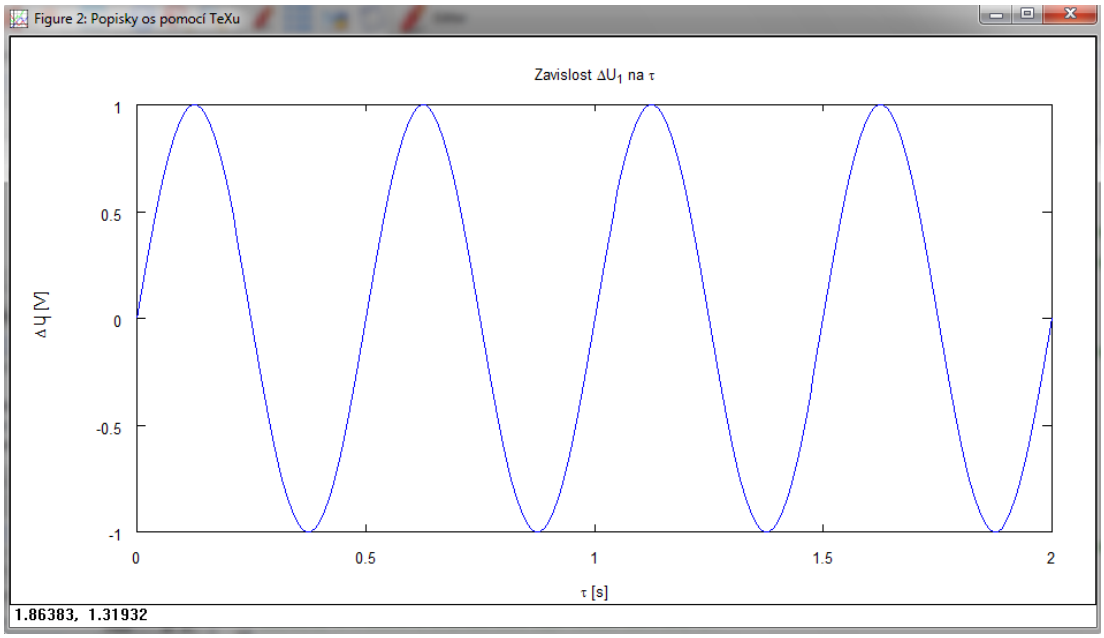
`title('Zavislost \DeltaU_{1} na \tau')` – název grafu.

Tentýž popis os a názvu grafu pomocí druhého způsobu:

`xlabel(' {/Symbol t} [s]')` – popis osy x pomocí druhého způsobu TeXové notace.

`ylabel(' {/Symbol D} U_{1} [V]')` – popis osy y pomocí druhého způsobu TeXové notace.

`title('Zavislost {/Symbol D} U_{1} na {/Symbol t}')` – název grafu.



Obrázek 5.2: *Popisky os pomocí TeXové notace*

Pro přehlednost grafu je někdy dobré zobrazit mřížku. Mřížku zobrazíme pomocí příkazu `grid on`. Pokud bychom chtěli mřížku vypnout, tak zadáme příkaz `grid off`. V Octave se nastavují škály os automaticky, chceme-li osy nastavit jinak, tak máme k dispozici příkaz `axis([`X_{MIN} X_{MAX} Y_{MIN} Y_{MAX}`])`. Octave také nabízí možnost formátování os podle uživatele (na osu můžeme uvést hodnoty nebo symboly pouze v určitých místech). Pro takové formátování použijeme následující příkazy:

`set(gca,'YLim',[-1 1])` – pomocí funkce `set` nastavíme limity osy y aktuálního objektu axes.

`set(gca,'YTick',[-1,0,1])` – nastavení pozic na ose y, kam chceme umístit hodnoty.

`set(gca,'YTickLabel',['min';'0';'max'])` – přiřazení hodnot na dané pozice.

Funkce `set` je podrobně popsána v kapitole 5.3. Ukázka takto nastavených os je na obrázku 5.3.

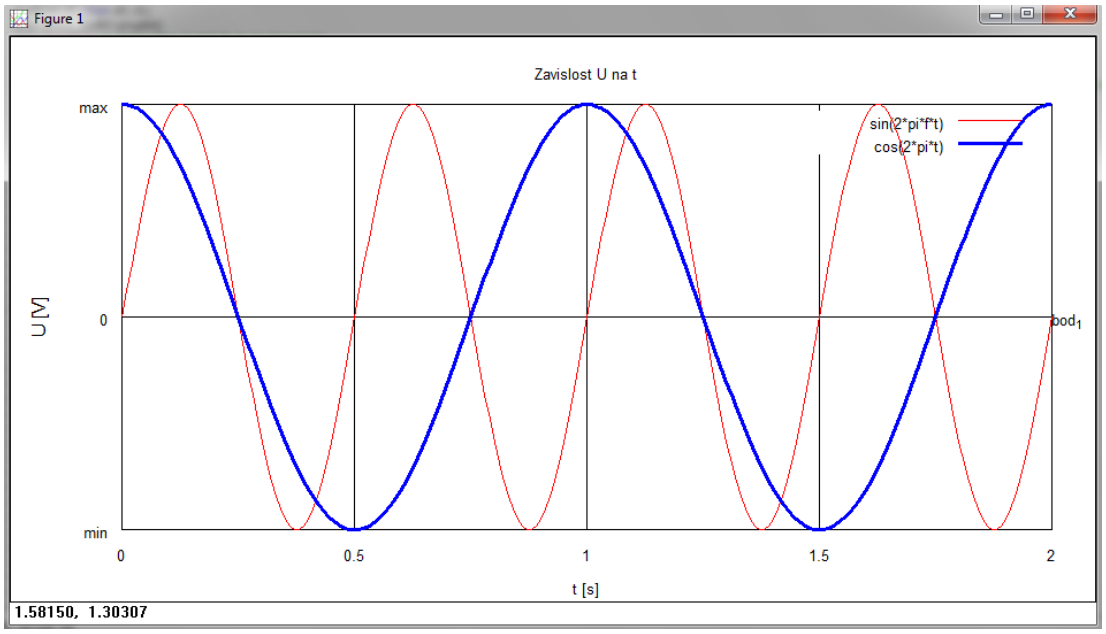
Chceme-li zobrazit více grafů v jednom okně, máme dvě možnosti. První možnost je pomocí příkazu `hold on`, tento příkaz způsobí, že aktuální graf nebude dalším překreslen. Po zadání příkazu `plot(x,y)` bude aktuální i předešlý graf v jednom okně. Příkaz `hold on` vypneme příkazem `hold off`.

`graf_1=plot(t,x,'r')` – zobrazení signálu x v závislosti na vektoru t.

`hold on` – podržení předchozího grafu.

`y=cos(2*pi*t)` – vytvoření signálu y.

`graf_2=plot(t,y, 'b', 'LineWidth',3)` – vykreslení grafu 2 do stejného okna jako graf 1. Barva grafu je nastavena na modrou a tloušťka čáry na 3.

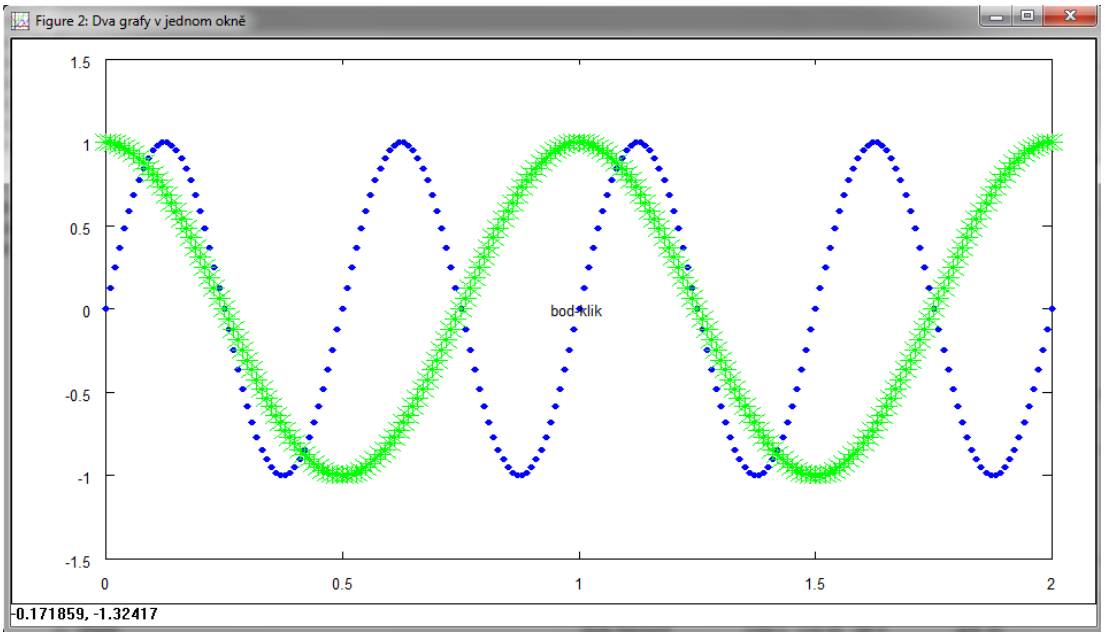


Obrázek 5.3: *Dva grafy v jednom okně s popisem a naformátovanou osou y*

Druhá možnost je přímo pomocí příkazu `plot(t, x, t, y)`. Pomocí příkazu `figure` si otevřeme nové okno.

`f2= figure('Name', 'Dva grafy v jednom okně')` – otevření nového okna s označením (handle) „f2“ a názvem „Dva grafy v jednom okně“.

`graf_3= plot(t, x, '.b', t, y, '.*g')` – vykreslení obou grafů naráz do jednoho okna. Signál x bude vykreslen tečkovaně (.) a modře (b), signál y bude hvězdičkový (.*) a zelený (g). Výsledný graf je zobrazen na obrázku 5.4.



Obrázek 5.4: *Dva grafy v jednom okně pomocí příkazu plot(t,x,t,y)*

5.3 Nastavení objektů figure a axes

Vlastnosti objektu lze definovat dvěma způsoby. První způsob je při vzniku objektu např. `figure('Color', 'y')` a druhý je později pomocí funkce `set` např. `set(gcf,'Color', 'y')`. Při práci s funkcí `set` se setkáme s funkcemi `gcf` a `gca`, které slouží k jednoduchému přístupu k identifikátorům objektů, které jsou právě aktivní. Konkrétně funkce `gcf` vrací identifikátor aktuálního objektu figure a funkce `gca` vrací identifikátor aktuálního objektu axes. Funkce `set` má následující parametry `set(id_objektu,,'vlastnost','konktrétní_hodnota_vlastnosti')`. Pomocí následujících příkazů si osvojíme jednoduchá nastavení těchto objektů. Výsledný graf je zobrazen na obrázku 5.5.

`f3=figure('Name', 'Nastavení objektů figure a axes')` – vytvoření nového okna figure f3. f3 je tzv. identifikátor objektu, v řeči Octave je to handle.

`graf_4=plot(t, x, '.g', t, y, 'r')` – vytvoření grafu graf_4.

`set(graf_4, 'MarkerSize', 20)` – nastavení velikosti teček grafu_4 na 20 pixelů.

`set(f3,'Color', 'y')` – nastavení barvy figure f3 na žlutou. Místo identifikátoru f3 můžeme použít příkaz `gcf`, který vrací identifikátor aktuálního objektu figure.

`set(gca,'Color', 'w')` – nastavení barvy oblasti os na bílou. Příkaz `gca` vrací identigikátor aktuálního objektu axes.

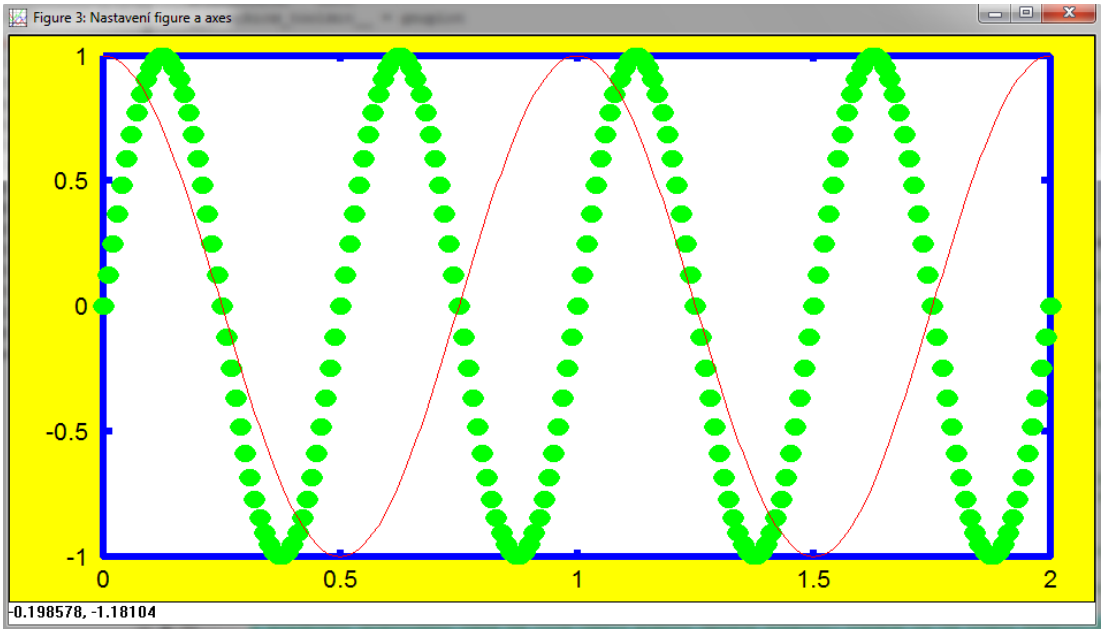
`set(gca,'FontSize',16)` – nastavení velikosti písma os.

`set(gca, 'LineWidth',6)` – nastavení šířky čar grafu_4.

S funkcí `set` také souvisí funkce `get`, pomocí které můžeme zjistit vlastnosti daného objektu.

`get(f3)` – funkce `get` vypíše do terminálu všechny vlastnosti objektu f3.

Pokud chceme nastavit defaultní nastavení všech objektů, je potřeba v příkazu `set` zadat jako indentifikátor objektu 0. Např. `set(0,'DefaultLineLineWidth',3)`, `set(0,'DefaultAxesFontSize',18)` atd.



Obrázek 5.5: *Nastavení objektů figure a axes*

5.4 Zobrazení více obrázků v jednom grafickém okně

Pro rozdělení grafického okna na více částí se v Octave používá příkaz `subplot(m,n,c)`. Parametr m znamená počet obrázků svisle, n je počet obrázků vodorovně a c je pozice příslušného obrázku. Ukázka více obrázků v jednom okně je na obrázku 5.6.

`f4=figure('Name','Více grafů v jednom okně')` – vytvoření nového okna figure f4.

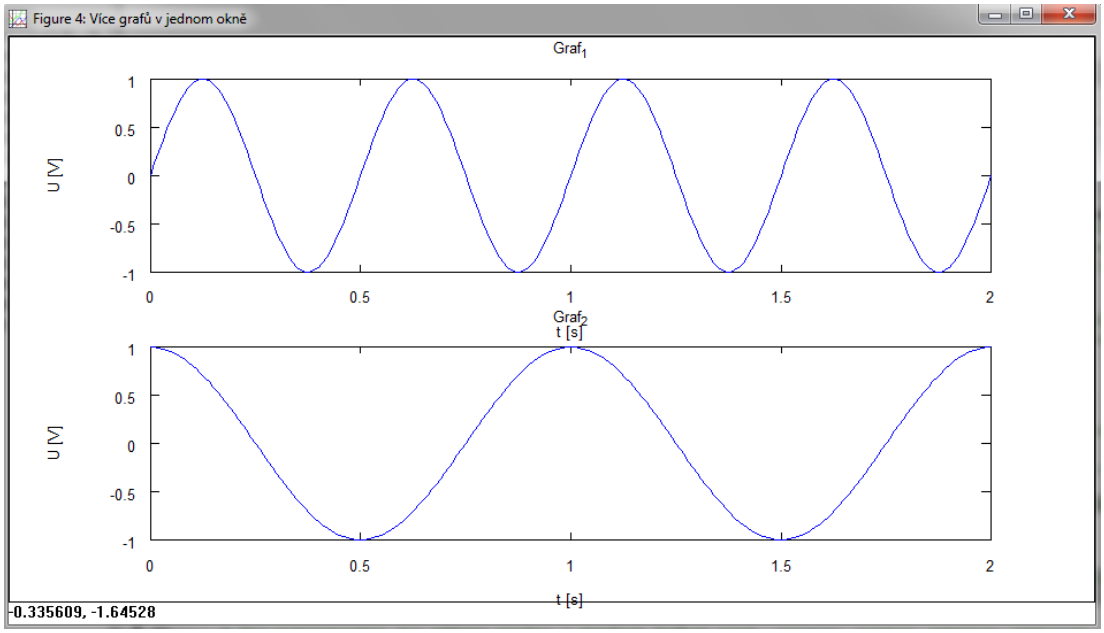
`subplot(2, 1, 1); plot(t, x)` – rozdělení figure f4 na 2 řádky a jeden sloupec. Vykreslení signálu x na pozici 1.

`title('Graf_1'); xlabel('t [s]'); ylabel('U [V]')` – popis grafu_1 a os.

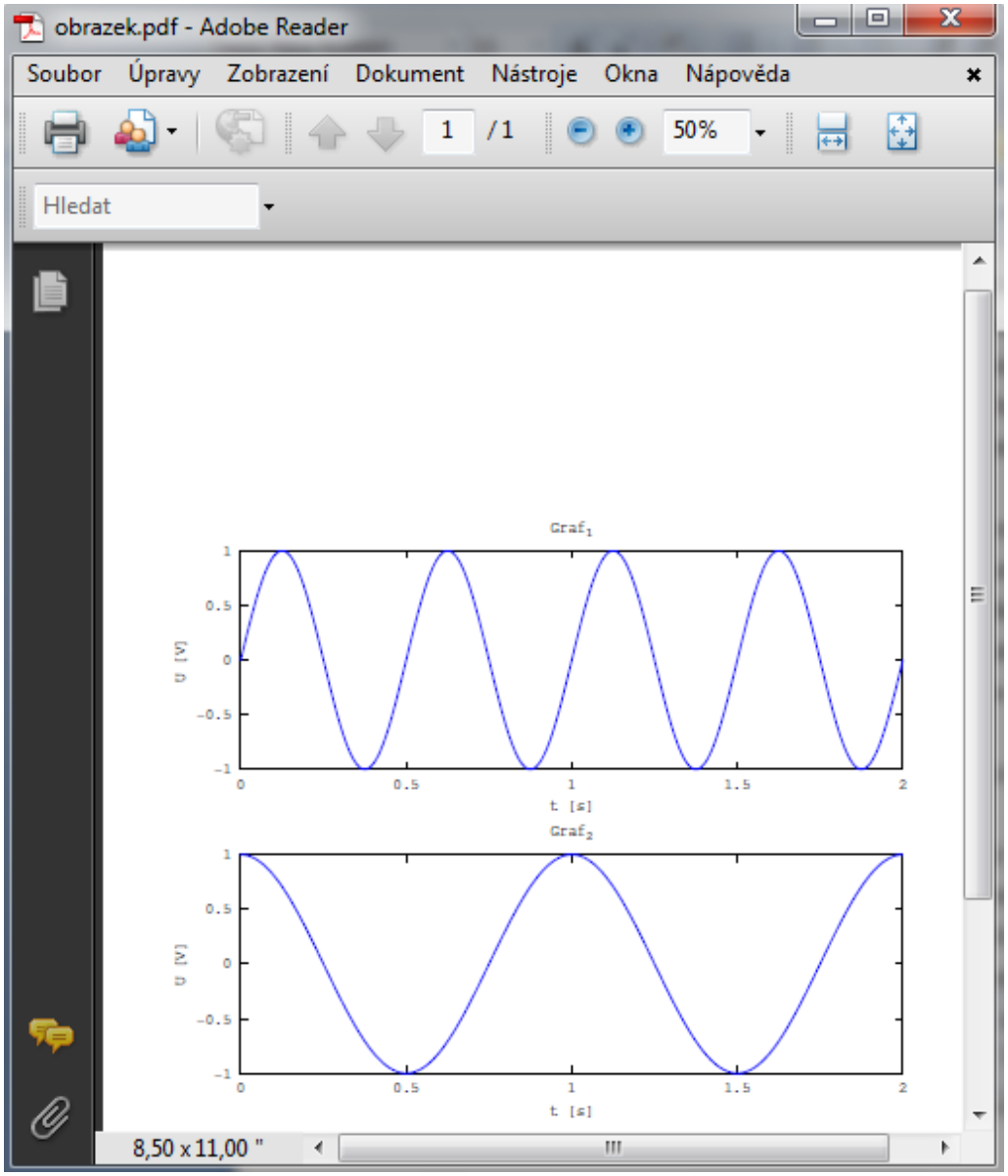
`subplot(2, 1, 2); plot(t, y)` – vykreslení signálu y na pozici 2.

`title('Graf_2'); xlabel('t [s]'); ylabel('U [V]')` – popis grafu_2 a os.

`print -dpdf obrazek.pdf` – pomocí příkazu `print` uložíme aktuální obrázek do souboru pdf. Uložení můžeme provést i do jiných souborů (png, jpg atd. všechny možnosti zjistíme pomocí příkazu `help print`). Soubor obrazek.pdf se uloží do adresáře, kde se právě nacházíme. (adresář vidíme (v QtOctave) v levém dolním rohu obrazovky). Ukázka pdf s grafem je na obrázku 5.7.



Obrázek 5.6: *Více obrázků v jednom grafickém okně*



Obrázek 5.7: *Graf uložený do pdf*

Pro vykreslování dat můžeme využít i dalších funkcí např. (`plotyy`, `loglog`, `semilogx`, `stairs`, `contour`, `splot`, `bar`, `barh`, `hist`, `pareto`, `errorbar`, `stem`, `area`, `pie`, `fill`, `contourf`, `image`, `pcolor`, `ezcontourf`, `feather`, `comet`, `polar`, `rose`, `compass`, `ezpolar`, `scatter`, `spy`, `plotmatrix`) práce s těmito funkcemi je obdobná jako práce s funkcí `plot`. Více detailů je rozebráno v helpu.

Příklady použitých funkcí a příkazů v této kapitole jsou uvedeny v příloze D na dvd v m-souboru *graf_2D.m*.

Při zpracování této kapitoly jsem čerpal z literatury [1], [6].

6 3D grafy

Práce s 3D grafy je obdobná jako u 2D grafů, proto si v následujícím textu uvedeme pouze základní tvorbu 3D grafů. Stejně jako u 2D grafů jsou funkce pro tvorbu 3D grafů implementovány přímo v samotném Octave, proto není potřeba mít nainstalovaný rozšiřující balíček. Formátování 3D grafů (tloušťka čar, barva atd.) se provádí pomocí stejných příkazů jako u 2D grafů. U popisu os nesmíme zapomenout na osu z. Popis osy z pak bude vypadat `zlabel('osa z')`. Ostatní příkazy pro popis grafu (legenda, název grafu, text v grafu atd.) jsou stejné jako u 2D grafu. Pokud chceme měnit rozsah os, je třeba myslet na to, že máme tři osy. Příkaz `axis` tedy bude vypadat následovně `axis([`X_{MIN} X_{MAX} Y_{MIN} Y_{MAX} Z_{MIN} Z_{MAX}`])`.

6.1 Vytvoření spojitého 3D grafu

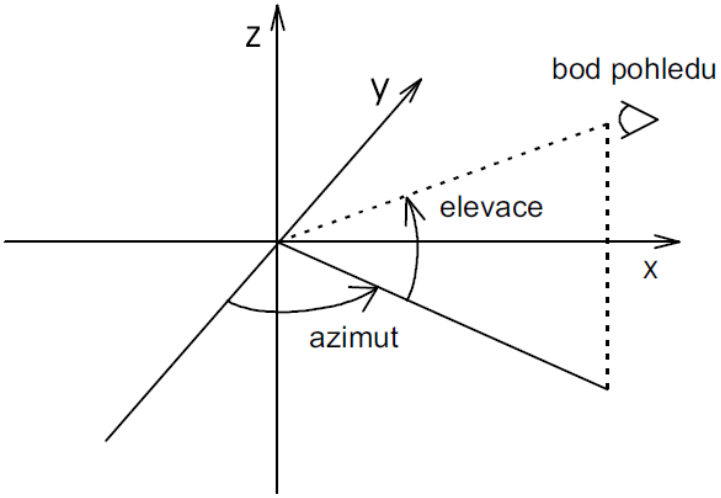
Tvorba 3D grafu principiálně vychází z tvorby 2D grafů. Na rozdíl od 2D grafů zde potřebujeme mít tři vektory popisující osy x, y, z. Příkaz pro vykreslení 3D grafu je `plot3(x,y,z)`.

`t=0:0.01:6*pi` – vytvoření vektoru t (6*pi znamená, že budeme mít tři otočky šroubovice).

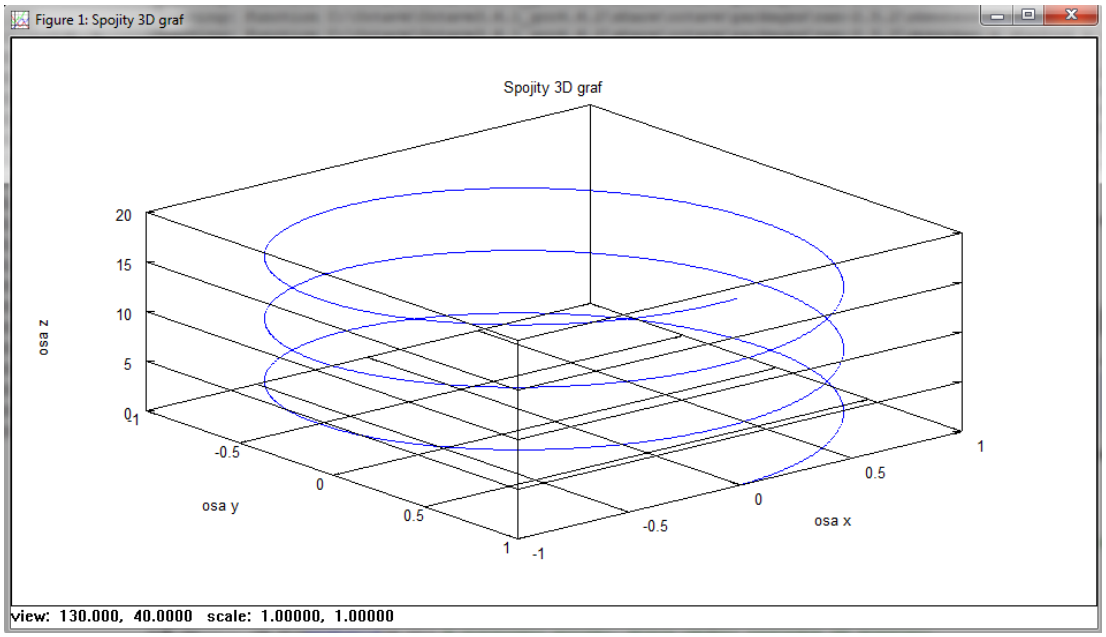
`f1 = figure('Name', 'Spojitý 3D graf')` – otevření nového okna figure s názvem Spojitý 3D graf.

`graf1=plot3(sin(t),cos(t),t)` – vytvoření spojitého 3D grafu (šroubovice), funkce sin(t) a cos(t) představují vektory, popisující osy x a y. Jedna otočka šroubovice je 2*pi.

Opět je možné pro zvýraznění grafu zapnout mřížku pomocí příkazu `grid on` nebo vytvořit rámeček (krychli) kolem objetu axes pomocí příkazu `box on`. Popis os a titulek grafu je možné vytvořit stejným způsobem jako u 2D grafu. Důležitým příkazem u 3D grafů je příkaz `view(az,el)`, pomocí kterého můžeme měnit úhel pohledu na graf. Hodnota az představuje azimut neboli horizontální rotaci, hodnota el představuje vertikální elevaci. Azimut se otáčí kolem osy z s kladnými hodnotami ve směru proti pohybu hodinových ručiček. Kladné hodnoty elevace odpovídají pohledu shora, záporné hodnoty pohledu zdola. Hodnoty obou úhlů jsou ve stupních. Výchozí nastavení pro azimut je -37,5° a pro elevaci 30°. Zobrazení azimutu a elevace je na obrázku 6.1. Výsledný spojitý 3D graf s `view(40,-40)` je zobrazen na obrázku 6.2.



Obrázek 6.1: *Zobrazení azimutu a elevace*



Obrázek 6.2: *Spojitý 3D graf*

6.2 Tvorba plošných 3D grafů

Základním objektem pro tvorbu plošných grafů je matice. Tuto matici (mřížku) vytvoříme pomocí speciálního příkazu `meshgrid(x,y)`. Po vytvoření základní mřížky máme řadu možností jak graf vykreslit. Můžeme například vykreslit základní typ 3D síťovaného grafu (obrázek 6.3), síťovaný graf doplněný o vrstevnice (obrázek 6.4) nebo základní plošný graf (obrázek 6.5).

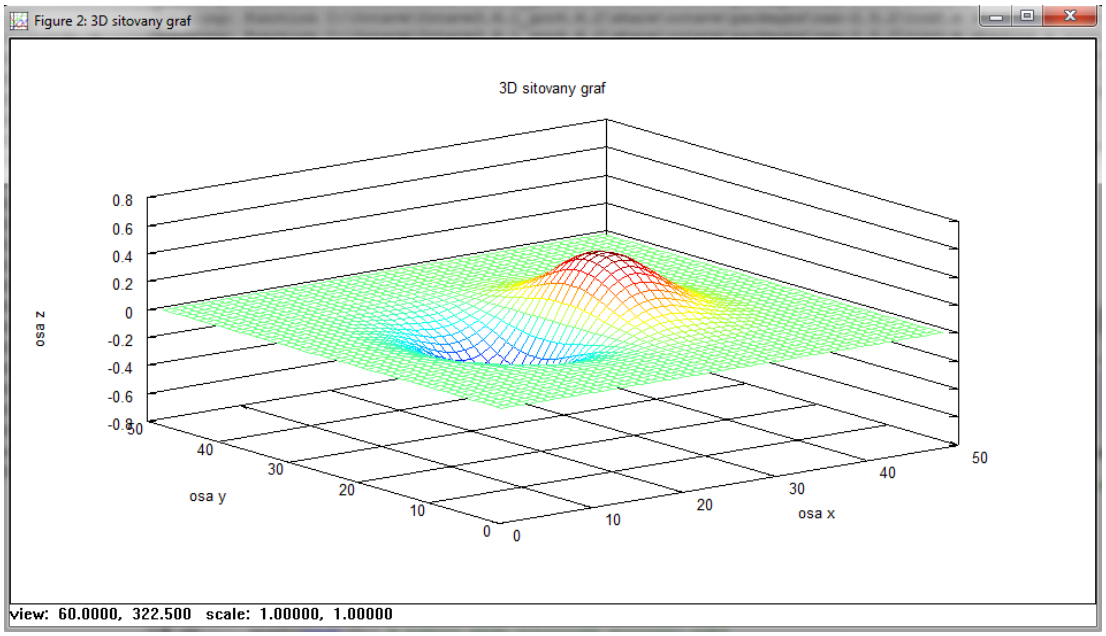
`x= -3:0.125:3` – definice první nezávisle proměnné.

`y=x` – definice druhé nezávisle proměnné.

`[X,Y]=meshgrid(x,y)` – vytvoření matice, která umožní následné 3D kreslení.

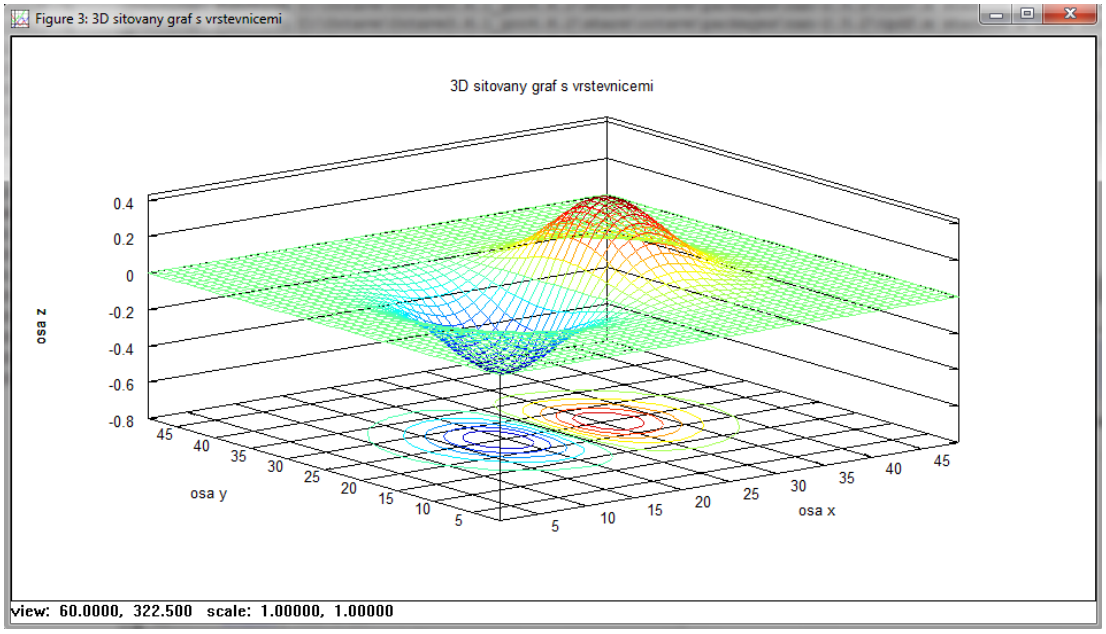
`Z=X.*exp(-X.^2-Y.^2)` – definice závisle proměnné.

`graf2=mesh(Z)` – vytvoření 3D síťovaného grafu.

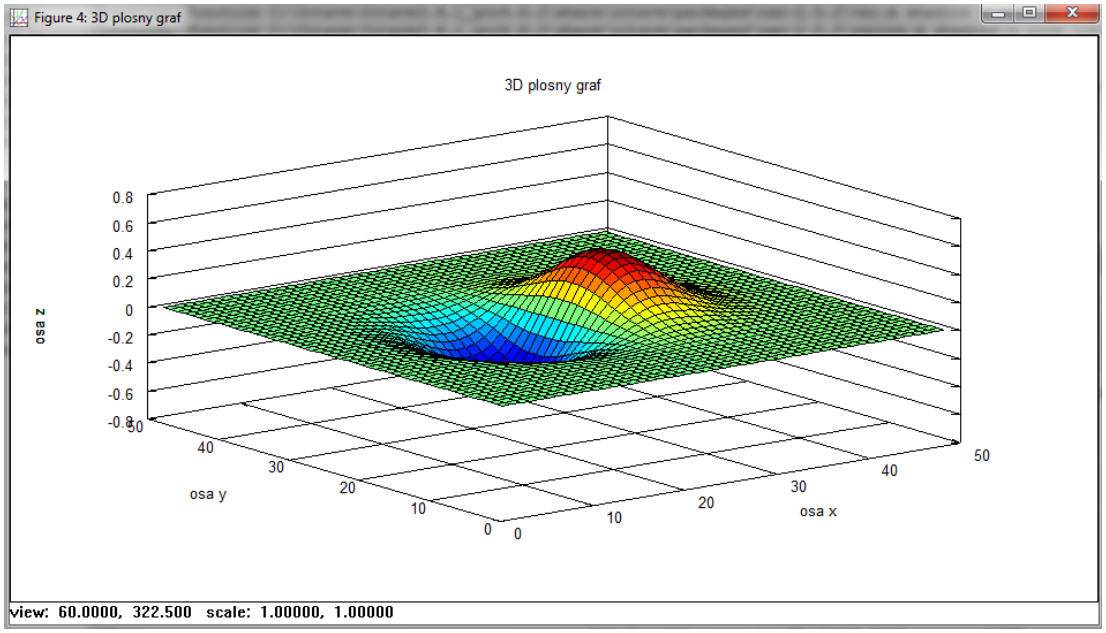


Obrázek 6.3: *3D síťovaný graf*

Pokud použijeme místo příkazu `mesh(Z)` příkaz `meshc(Z)`, tak dojde k vykreslení síťovaného grafu s vrstevnicemi (obrázek 6.4). Příkaz `meshc(Z)` tvoří kombinaci příkazů `mesh(Z)` a `contour(Z)`. Pro vykreslení plošného 3D grafu použijeme místo příkazu `mesh(Z)` příkaz `surf(Z)` (obrázek 6.5). Pomocí příkazu `colormap()` můžeme měnit barevnou škálu plošného grafu. Stejně jako u 2D grafů, tak i zde máme možnost kreslit více obrázků s 3D grafy do jednoho okna. Opět je k tomu použit příkaz `subplot(m, n, c)`.



Obrázek 6.4: *3D síťovaný graf s vrstevnicemi*



Obrázek 6.5: *3D plošný graf*

Pro vykreslování dat můžeme využít i dalších funkcí např. (`contour3`, `ezplot3`, `meshz`, `ezmesh`, `stem3`, `pie3`, `patch`, `cylinder`, `ellipsoid`, `surfl`, `surfc`, `ezsurf`, `ezsurfc`, `quiver3`, `comet3`, `scatter3`). Práce s těmito funkcemi je obdobná jako práce s funkcí `plot3`. Více detailů je rozebráno v helpu.

Příklady použitých funkcí a příkazů v této kapitole jsou uvedeny v příloze E na dvd v m-souboru *graf_3D.m*.

Při zpracování této kapitoly jsem čerpal z literatury [1], [6].

7 Ladění zdrojového kódu (debugging)

Při psaní zdrojového kódu se nám může stát, že někde uděláme chybu. U krátkých programů to nemusí být až takový problém, ale u delšího kódu už může být nalezení chyby obtížnější, proto i Octave (jako ostatní programovací jazyky) obsahuje tzv. debugger (ladící program), který umožňuje snadnější nalezení chyby. V Octave máme dvě základní možnosti jak program ladit. První možnost je pomocí příkazů, které zadáváme do terminálového okna. Druhá možnost je, že ladění provedeme v editoru/debugeru grafického rozhraní (GUI Octave, QtOctave atd.) pomocí příslušných tlačítek. Funkce pro ladění zdrojového kódu jsou součástí samotného Octave.

7.1 Ladění zdrojového kódu pomocí příkazů

Než začneme se samotným laděním, je nejprve nutné přepnout aktuální složku, ve které se nacházíme na složku, kde je umístěn náš m-soubor. Octave pracuje se stejnými příkazy pro práci s adresáři a soubory jaké známe z příkazové řádky. Pro nás jsou v tuto chvíli důležité dva příkazy a to příkaz `pwd`, pomocí kterého se nám zobrazí složka, kde se aktuálně nacházíme. Druhý příkaz je příkaz `cd`, který umožňuje změnu adresáře.

Existují dva způsoby přerušení probíhajícího skriptu (vstupu do ladícího režimu). První způsob je nastavením tzv. breakpointu (místo přerušení) nebo můžeme přerušení docílit na základě splnění určité podmínky. Tyto podmínky jsou tři. Přerušení na základě chyby, přerušení na základě varování nebo přerušení na základě přerušení (např. vstup do funkce). Po splnění námi nastavené podmínky se program přepne do ladícího režimu.

7.1.1 Vstup ladícího režimu

`debug_on_error(1)` – nastavení přerušení programu po detekci chyby v kódu, hodnotu funkce musíme nastavit na 1, aby byla funkce aktivní. Automaticky je nastavena na 0, tedy vypnuta.

`debug_on_warning(1)` – nastavení přerušení programu po detekci varování.

`debug_on_interrupt(1)` – nastavení přerušení programu po detekci přerušení.

Po zadání příkazu už stačí jen program spustit. Pokud je program bez chyb provede se stejně jako při klasickém spuštění. Je-li v programu nějaká chyba, tak se program na tomto místě zastaví, vypíše se chybové hlášení a program se přepne do ladícího režimu, viz obrázek 7.1.

```
>> debug_on_error(1)

>> zksym
error: `sy' undefined near line 2 column 5
error: called from
    zksym at line 2 column 3
stopped in C:\Users\asus\Desktop\zksym.m at line 2
2: z = sy("z");
debug>
```

Obrázek 7.1: *Vstup do ladícího režimu po detekci chyby v programu zksym*

7.1.2 Opuštění ladícího režimu

`dbcont` – opuštění ladícího režimu a zbytek programu se vykoná normálně.

`dbquit` – opuštění ladícího režimu a program se korektně ukončí.

7.1.3 Breakpointy (místa přerušení)

`dbstop('název_m-souboru',číslo řádku)` – nastavení breakpointu v m-souboru (ve funkci) na příslušném řádku.

`dbclear('název_m-souboru',číslo řádku)` – odstranění breakpointu v příslušného řádku.

`dbstatus` – vypíše informace o nastavených breakpointech (čísla řádků).

7.1.4 Ladící režim

`dbwhere` – v ladícím režimu vypíše zprávu o aktuálním souboru a číslo řádku, na kterém došlo k přerušení programu.

`dbtype` – v ladícím režimu vypíše do terminálového okna skript s čísly řádků.

`isdebugmode` – slouží ke kontrole, zda se nacházíme v ladícím režimu, pokud funkce vrátí 1, nacházíme se v ladícím režimu, pokud vrátí 0, nenacházíme se v ladícím režimu.

`dbstep` – slouží ke krokování programu, při zadání této funkce se vykoná jeden řádek programu, pokud se v programu narazí na volání dalšího m-souboru nebo funkce nedojde k zanoření, ale funkce se provede v jednom kroku.

`dbstep n` – pokud zadáme parametr n (číslo řádku), tak se vykonají všechny řádky až po n včetně.

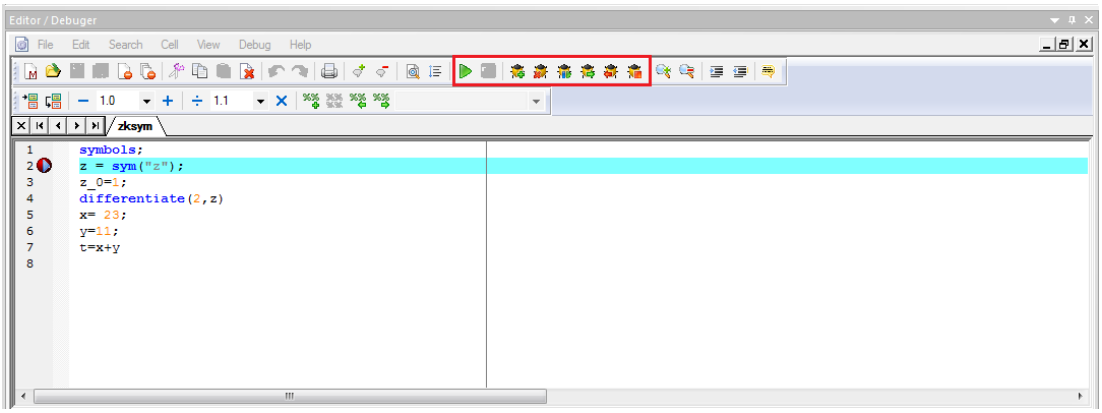
`dbstep in` – slouží ke krokování programu, kdy případné volání jiného m-souboru nebo funkce způsobí zanoření ladícího systému dovnitř této funkce.

`dbstep out` – slouží k okamžitému návratu z volané funkce.

`dbnext` – obdobná funkce jako `dbstep`.

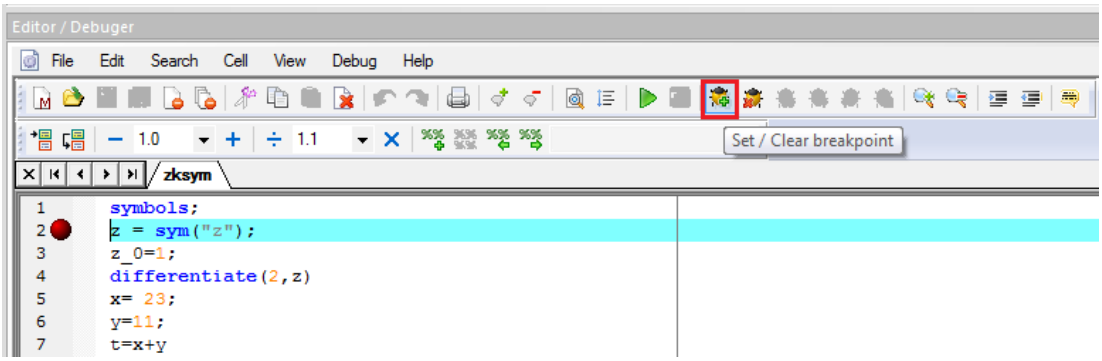
7.2 Ladění zdrojového kódu v GUI Octave

Díky grafickému rozhraní GUI Octave máme možnost provádět ladění programu efektivněji a přehledněji než tomu bylo v předchozím případě. Ladění v GUI Octave je obdobou ladění na které jsme zvyklí z Matlabu. V podstatě využíváme stejných příkazů jako v předchozím případě, ale tentokrát zadáváme příkazy pomocí speciálních tlačítek. Tento fakt může být pro řadu uživatelů velká výhoda a může znamenat značné ulehčení práce. Tlačítka se nachází v editoru na horní liště. Umístění tlačítek je zobrazeno na obrázku 7.2.



Obrázek 7.2: *Umístění tlačítek určených k ladění programu v GUI Octave*

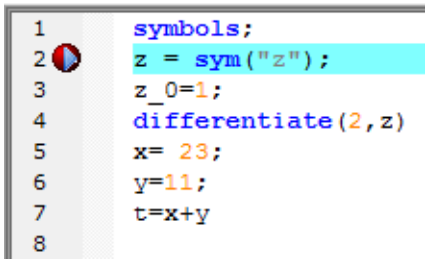
Abychom mohli začít se samotným laděním, je potřeba na některý řádek zdrojového kódu umístit tzv. místo přerušení (breakpoint), nejčastěji ho vkládáme na začátek kódu. Umístění breakpointu můžeme provést dvěma způsoby. První způsob je, že na příslušný řádek umístíme kurzor a poté stiskneme tlačítko set/clear breakpoint viz. obrázek 7.3. Stejným tlačítkem breakpoint i odstraníme. Vedle čísla řádku se zobrazí červený puntík, který reprezentuje breakpoint.



Obrázek 7.3: *Umístění breakpointu*

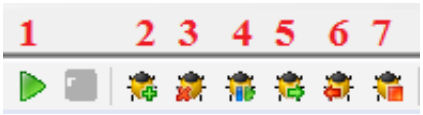
Druhá možnost je, že v editoru na pravé straně od řádku m-souboru provedeme dvojklik. Tímto způsobem docílíme stejného výsledku jako vidíme na obrázku 7.3. Ať už breakpoint umístíme jakýmkoliv způsobem, tak se vždy v terminálovém okně objeví příkazy, které jsme si vysvětlily v kapitole 7.1.

V dalším kroku už můžeme daný program spustit pomocí tlačítka run/debug/continue viz obrázek 7.5. Vykonávání programu se přeruší na řádku, kde jsme umístili breakpoint. Tento řádek je označen modrou šipkou (obrázek 7.4). Systém čeká na další zásah.



Obrázek 7.4: *Přerušení programu na řádku s breakpointem*

Nyní můžeme začít s krokováním programu. Ke krokování slouží dvě tlačítka. Tlačítko step a step in. Stiskem tlačítka step dojde k vykonání jednoho řádku programu, v terminálovém okně se nám navíc zobrazí, na jakém řádku kódu se nacházíme a co daný řádek obsahuje. Pokud se na daném řádku nachází chyba, tak po jeho vykonání dojde k výpisu chybového hlášení v terminálovém okně. Z výpisu chybového hlášení je patrné, na jakém řádku se chyba nachází a jakého je charakteru. Tlačítko step in použijeme tehdy, je-li v m-souboru volána další funkce nebo m-soubor a chceme krokovat i obsah této funkce. Pokud chceme zanořenou funkci opustit, stiskneme tlačítko step out. Jak jednotlivá tlačítka vypadají, je zobrazeno na obrázku 7.5.

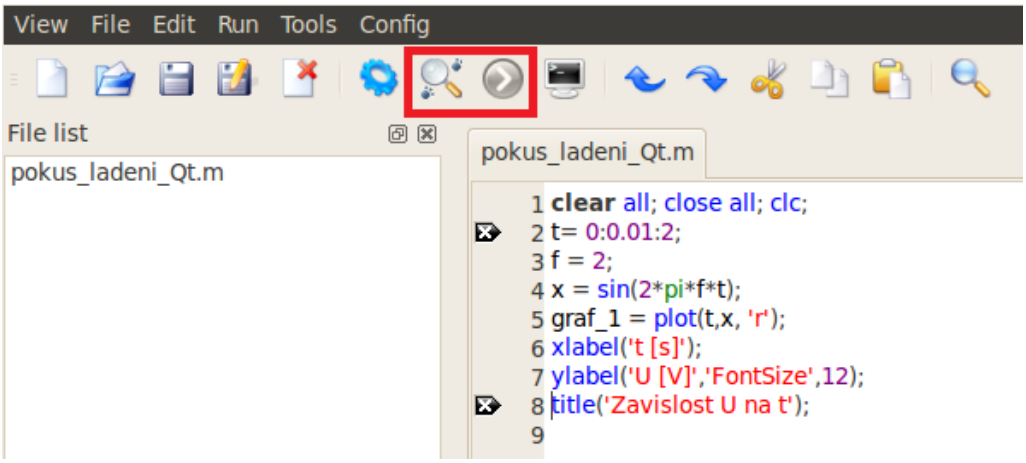


Obrázek 7.5: *Zobrazení všech tlačítek určených k ladění programu*

1- Tlačítko run/debug/continue
2- Tlačítko set/clear breakpoint
3- Tlačítko clear breakpoints in all files
4- Tlačítko step
5- Tlačítko step in
6- Tlačítko step out
7- Tlačítko stop

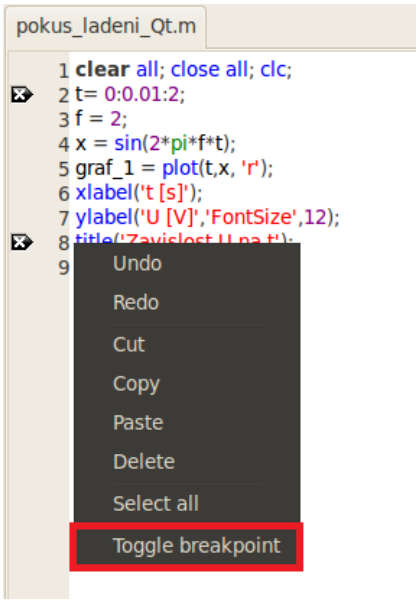
7.3 Ladění zdrojového kódu v QtOctave

Laděním zdrojového kódu v programu QtOctave je založeno na stejném principu jako v GUI Octave, opět se využívá standardních příkazů z kapitoly 7.1, které zadáváme pomocí příslušných tlačítek, viz obrázek 7.6.



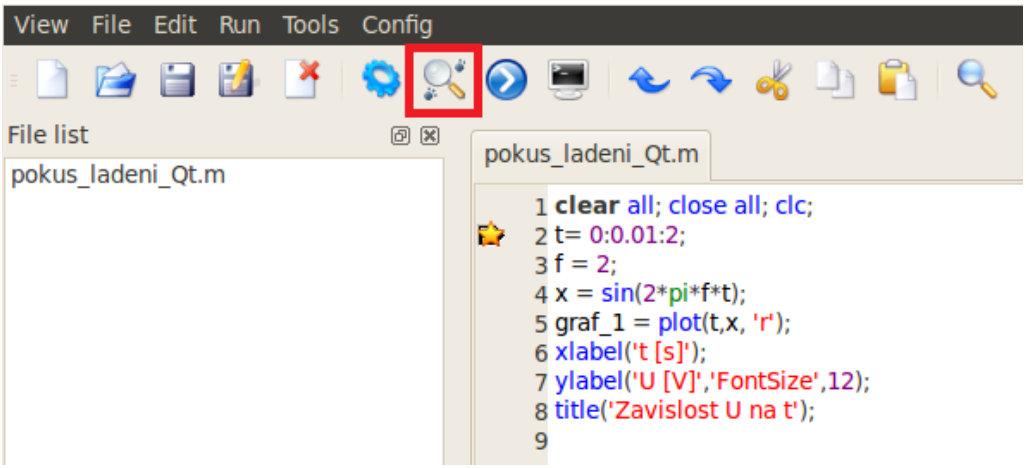
Obrázek 7.6: *Tlačítka určená k ladění zdrojového kódu v QtOctave*

Na počátku ladění je opět potřeba do kódu umístit breakpoint (start ladění). Pokud pracujeme v systému Windows, tak breakpoint nastavíme (nebo odstraníme) tak, že na řádek, kam chceme breakpoint umístit klepneme myší a zmáčkneme klávesu F7. V systému Linux to provedeme tak, že na řádek, kam chceme breakpoint umístit, klepneme levým tlačítkem myši a poté přes pravé tlačítko myši vybereme možnost toggle breakpoint. Stejným způsobem lze i breakpoint odstranit. Tato operace je zobrazena na obrázku 7.7.



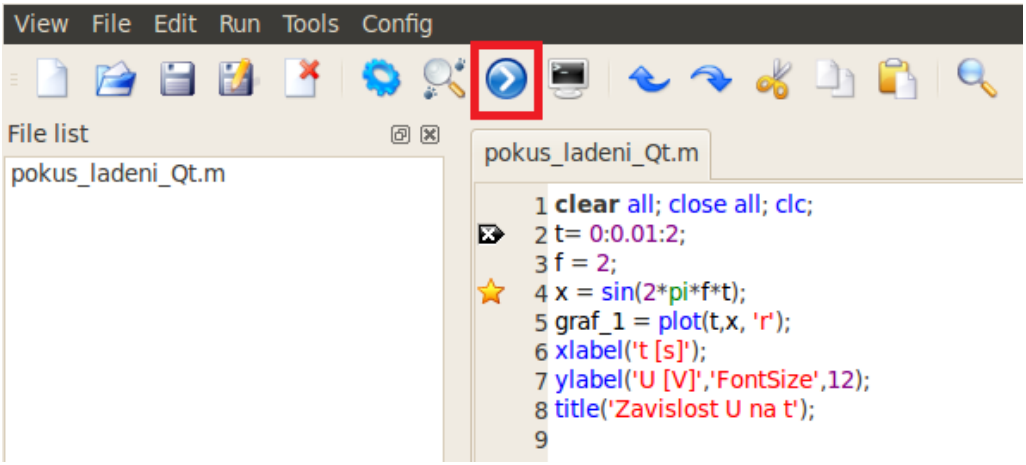
Obrázek 7.7: *Nastavení/Odstranění breakpointu v systému Linux*

Po nastavení breakpointu je program připraven k ladění. Pro vstup do ladícího režimu je potřeba program spustit pomocí tlačítka debut. Pokud program spustíme pomocí tlačítka run, tak se program vykoná celý bez ohledu na breakpointy. Po spuštění pomocí tlačítka debug se program zastaví na prvním nastaveném breakpointu, místo se označí hvězdičkou (pouze v systému Linux, ve Windows nedojde k označení), viz obrázek 7.8. K ukončení ladícího režimu se používá totéž tlačítko, v tomto případě zastává funkci příkazu `dbcont`.



Obrázek 7.8: *Spuštění ladícího režimu*

Nyní je program připraven k samotnému ladění. Krokování provádíme pomocí tlačítka detailed debugging, které vykonává funkci příkazu `step`. Ukázka krokování je vidět na obrázku 7.9.



Obrázek 7.9: *Krokování programu*

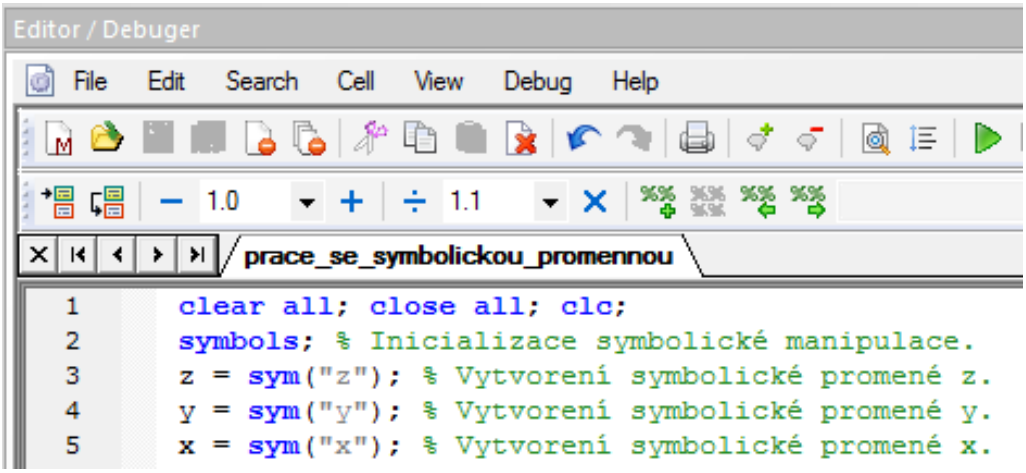
Při zpracování této kapitoly jsem čerpal z literatury [1], [7].

8 Práce se symbolickými proměnnými

V některých případech je pro nás výhodnější pracovat se symbolickými proměnnými než s čísly. Pro práci se symbolickými proměnnými v Octave slouží balíček symbolic. Pokud tento balíček nemáme nainstalovaný, tak stačí postupovat podle kapitoly 2 (Instalace GNU Octave), kde je podrobně popsána instalace právě tohoto balíčku.

8.1 Inicializace balíčku symbolic

Abychom mohli balíček symbolic využívat, je potřeba na začátku m-souboru, nebo než začneme se symbolickými proměnnými pracovat, provést jeho inicializaci. Inicializace se provádí zadáním příkazu `symbols`, jak je vidět na obrázku 8.1. Po zadání tohoto příkazu je balíček aktivní a můžeme s ním začít pracovat.



Obrázek 8.1: *Inicializace symbolické manipulace*

8.2 Možnosti balíčku symbolic

Možnosti balíčku symbolic nejsou tak velké a rozsáhlé jako třeba u Symbolic math toolbox v programu Matlab, proto si uvedeme jen ty nejdůležitější a nejpoužívanější funkce, se kterými se můžeme setkat. Největší nevýhodou tohoto balíčku je absence možnosti symbolického integrování. Jelikož dochází k pravidelným aktualizacím balíčků, tak se dá očekávat, že nedokonalosti budou odstraněny a balíček bude doplněn o další funkce. Použití níže uvedených příkazů a funkcí je popsáno v příloze F na dvd v m-souboru *práce_se_symbolickou_promennou*.

`sym('x')` – vytvoření symbolické proměnné x.

`is_ex(r)` – funkce vrátí 1, pokud je objekt r symbolickým vyjádřením.

`is_sym(x)` – funkce vrátí 1, pokud je objekt x typu sym, jinak vrátí 0.

`syminfo(x)` – vypíše informace o symbolické proměnné x.

`subs(r,x,n)` – nahrádí ve výrazu r symbolickou proměnou x číslem n.

`collect(r,x)` – slouží ke zjednodušení výrazu, ve výrazu r dojde k vytknutí x.

`expand(r)` – slouží ke zjednodušení výrazu, dojde k roznásobení výrazu r.

`Sin(x)` – symbolický sinus (totéž platí pro Cos i Tan).

`Sinh(x)` – symbolický hyperbolický sinus (totéž platí pro Cosh i Tanh).

`aSin(x)` – symbolický inverzní sinus (totéž platí pro aCosh i aTanh).

`aSinh(x)` – symbolický inverzní hyperbolický sinus (totéž platí pro aCosh i aTanh).

`Log(x)` – symbolický logaritmus.

`Exp(x)` – symbolická exponenciála.

`Sqrt(x)` – symbolická odmocnina.

`splot(f,x,t)` – vykreslení symbolické funkce v rozmezí t.

`differentiate(f,x,n)` – symbolický výpočet n-té derivace f podle x.

Při zpracování této kapitoly jsem čerpal z literatury [2].

9 Tvorba animací

Pro tvorbu animací v GNU Octave jsou k dispozici dva základní přístupy. První přístup spočívá v tom, že v určitém cyklu dochází k překreslování stávajícího grafu pomocí funkce `drawnow`. Při použití grafické sady `graphics_toolkit gnuplot` je překreslování značně pomalé a animace se jeví jako nespojitá. Pokud je použita grafická sada `graphics_toolkit ftlk` dojde ke zrychlení překreslování, ale na druhou stranu po dokončení animace nedojde ke korektnímu ukončení scriptu a program přestane běžet. Pod systémem Linux je jako výchozí nastavena sada `graphics_toolkit gnuplot`, ale i přesto je překreslování rychlejší než pod systémem Windows. Jako nejvýhodnější přístup se tedy jeví druhý přístup, který je založen na postupném ukládání stávajícího grafu jako obrázku a následném sloučení obrázků do videa pomocí speciálního programu, k dispozici ke stažení jsou například (MonkeyJam, VideoSpin atd.). Výsledkem tohoto přístupu jsou spojité animace na vysoké úrovni. Určitou nevýhodou je použití speciálního programu, bez kterého se v tomto případě neobejdeme. Z tohoto důvodu je výhodnější pro jednodušší aplikace použít první přístup. Funkce pro tvorbu animací jsou obsaženy v základní verzi Octave.

9.1 Tvorba animací pomocí funkce drawnow

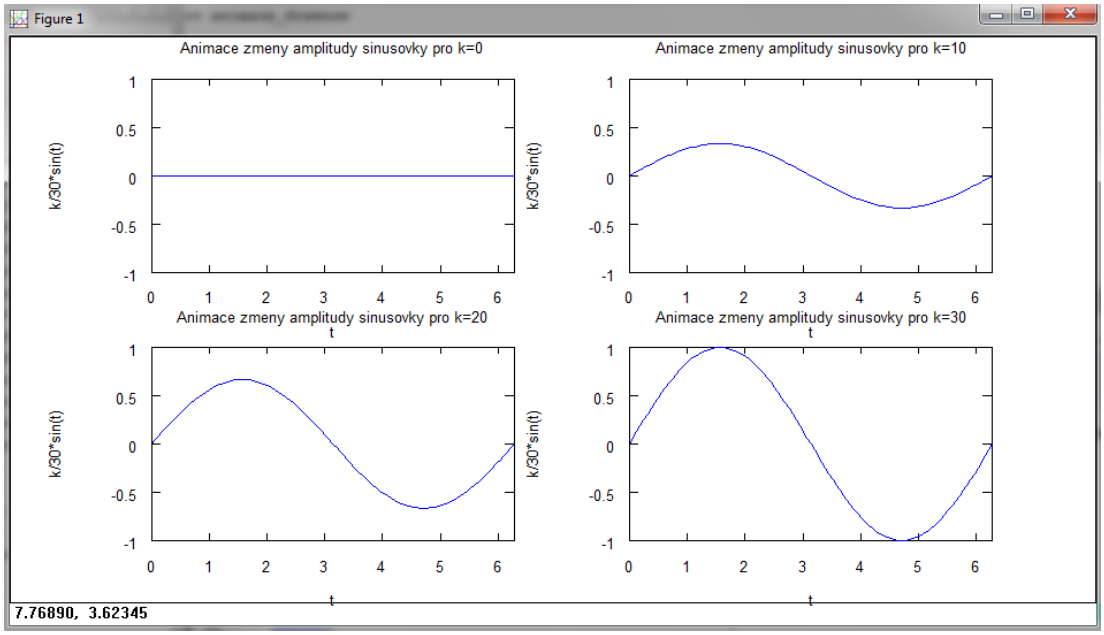
Jak bylo zmíněno v úvodní části, tak při použití funkce `drawnow` dochází k překreslování grafů ve stávajícím okně. Abychom docílili pravidelného překreslování, je potřeba v kódu vytvořit cyklus, ve kterém po každé změně proměnných dojde k překreslení.

- Animace změny amplitudy sinusovky

Jako základní ukázka poslouží změna amplitudy sinusovky od 0 do 1. Na počátku (hodnota amplitudy 0) máme rovnou přímku. Každým průchodem cyklu se amplituda zvyšuje až po maximální hodnotu (hodnota amplitudy 1), zároveň dojde v každém průchodu cyklem k vykreslení aktuální hodnoty do grafu. Obsah m-souboru je vidět na obrázku 9.1. Na obrázku 9.2 jsou vidět aktuální grafy pro příslušné hodnoty k.

```
1  %% Animace zmeny amplitudy sinusovky pomocí funkce drawnow
2  clear all; close all; clc;
3  t=0:2*pi/100:2*pi; % Vytvoreni nezavisle prommene
4  figure; % Otevreni okna figure
5  for k=0:30 % Cyklus pro k od 0:30
6      plot(t,k/30*sin(t)) % Vykresleni grafu
7      axis([0,2*pi,-1,1]) % Nastaveni os
8      xlabel('t'); ylabel('k/30*sin(t)'); title('Animace zmeny amplitudy sinusovky'); % Popis os a nazev grafu
9      drawnow % Funkce drawnow vykreslní aktuální graf, pokud by zustala v cyklu pouze ...
10     % ...funkce plot doslo by k vykresleni az vysledneho grafu
11 end % Konec cyklu
```

Obrázek 9.1: *Zdrojový kód animace změny amplitudy sinusovky pomocí funkce drawnow*



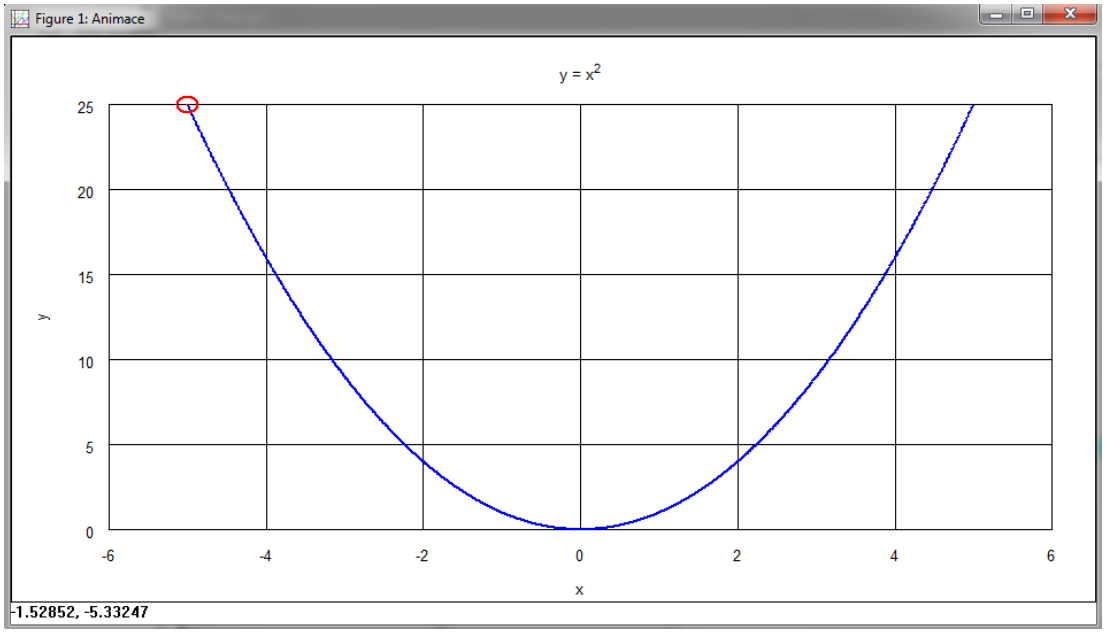
Obrázek 9.2: *Animace změny amplitudy sinusovky pro k=0; 10; 20 a 30*

- Animace pohybu bodu po křivce

Druhá ukázka představuje možnost animace pohybu bodu po křivce pomocí funkce `drawnow` a funkce `set`. Způsob realizace je takový, že nejprve vytvoříme křivku, po které se má bod pohybovat. Poté vytvoříme bod s výchozími parametry (souřadnice, barva, tvar, velikost). Na závěr vytvoříme cyklus, ve kterém se budou pomocí funkce `set` aktualizovat souřadnice bodu a pomocí funkce `drawnow` překreslovat do grafu. Výhodou této koncepce je, že dochází pouze k překreslování bodu a není tedy nutné znovu překreslovat celý graf (průběh animace je rychlejší). Ukázka zdrojového kódu je na obrázku 9.3 a na obrázku 9.4 je, jak vypadá výchozí situace této animace.

```
1   %% Animace pohybu bodu po krivce pomoci funkce drawnow a funkce set
2   clear all; close all; clc;
3   x = -5:0.01:5; %Vytvoreni nezavisle promenne
4   y = x.^2; %Vytvoreni zavisle promenne
5   % Vykresleni grafu
6   figure('Name', 'Animace');
7   plot(x,y, 'LineWidth',2);
8   grid on;
9   xlabel('x'); ylabel('y'); title('y = x^2');
10  % Vytvoreni bodu
11  bod = line('XData',x(1), 'YData',y(1), 'Color','r', ...
12      'Marker','o', 'MarkerSize',6, 'LineWidth',2); % Bod bude cerveny, tvar kolecka,...
13    %...velikost bude 6 a sirka cary bude 2
14  % Cyklus pro pohyb bodu po grafu
15  for x=-5:0.1:5
16      y=x.^2;
17      set(bod, 'XData',x, 'YData',y)    % Nastavni aktualnich souradnic
18      drawnow    % Vykresleni bodu pro aktualni souradnice
19  end
```

Obrázek 9.3: *Zdrojový kód pro animaci pohybu bodu po křivce*



Obrázek 9.4: *Výchozí situace animace pohybu bodu po křivce*

Zdrojové kódy animací jsou uvedeny v příloze G na dvd.

9.2 Tvorba animací pomocí ukládání aktuálních obrázků

Postup při tvorbě animací pomocí ukládání obrázku je obdobný jako v předchozím případě pouze s tím rozdílem, že nevyužíváme v cyklu funkci `drawnow`, ale v daném cyklu dochází k ukládání aktuálních grafů do složky. Ukládání je realizováno pomocí funkce `print`.

- Animace změny amplitudy sinusovky pomocí ukládání aktuálních obrázků

Pro tento případ použijeme stejnou ukázku, kterou jsme si ukázali v případě animací pomocí funkce `drawnow`. Zdrojový kód je obdobný jako v předchozím případě pouze s tím rozdílem, že si na začátku vytvoříme složku, kam chceme obrázky ukládat. Tuto složku vytvoříme pomocí funkce `mkdir` a v cyklu nepoužijeme funkci `drawnow`, ale použijeme funkce `sprintf` a `print`. Funkci `sprintf` používáme z důvodu snadnějšího a přehlednějšího ukládání. Tato funkce zapisuje formátovaná data do textového řetězce. Funkce `print` uloží aktuální graf do námi vytvořené složky. Po proběhnutí celého cyklu jsou ve složce uloženy všechny potřebné grafy k animaci. Ke spojení obrázků do animace použijeme některý z volně dostupných programů pro tvorbu animací z obrázků například (MonkeyJam, VideoSpin atd.). Na obrázku 9.5 je vidět zdrojový kód této „animace“. Výsledná animace vytvořená pomocí programu VideoSpin je společně se zdrojovým kódem uvedena v příloze G na dvd.

```
%% Animace zmeny amplitudy sinusovky pomocí ukládání aktuálních obrázku
clear all; close all; clc;
t=0:2*pi/100:2*pi;
mkdir('vystup'); %Vytvoreni slozky s nazvem vystup
for k=0:30
    plot(t,k/30*sin(t));
    axis([0,2*pi,-1,1]);
    nazev=sprintf('vystup/%02d.png',k); % Vytvoreni textoveho retezce pod kterym se obrázek ulozí...
    % %02d.png znamená ulozeni na dve mista (01.png atd.)
    print(nazev); %Ulozeni aktualniho grafu do slozky vystup
end
```

Obrázek 9.5: *Zdrojový kód pro vytvoření animace pomocí ukládání obrázků*

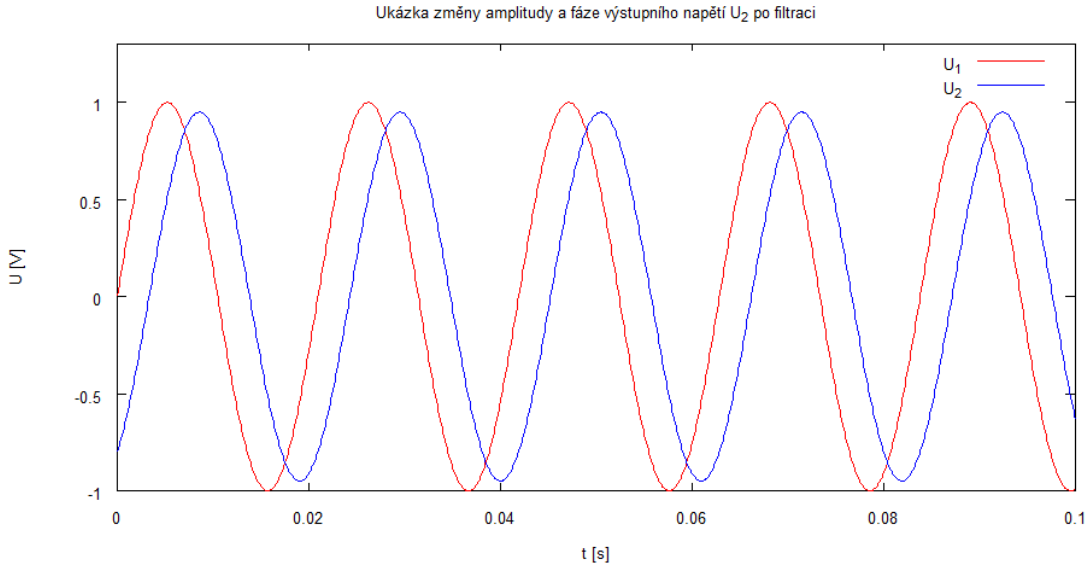
9.3 Tvorba 3D animací

Pro tvorbu 3D animací můžeme využít obou předchozích variant. Pouze u varianty s překreslováním grafů pomocí funkce `drawnow` je překreslování pomalejší (3D graf je složitější), tudíž je výhodnější využít druhou variantu.

Při zpracování této kapitoly jsem čerpal z literatury [6], [8], [9].

10 Číslicové filtry

Filtr je prvek, který některé frekvence přivedené na vstup propustí na svůj výstup a jiné potlačí podle předem definovaných pravidel. Reálný číslicový filtr je soustava, která mění frekvenčně amplitudovou a frekvenčně fázovou charakteristiku signálu. Průběh vstupního a výstupního signálu na dané frekvenci je zobrazen na obrázku 10.1. Abychom mohli v Octave filtry navrhovat je nutné mít nainstalovaný rozšiřující balíček signal a mít k dispozici funkce obsažené v příloze H na dvd.



Obrázek 10.1: *Ukázka změny amplitudy a fáze výstupního napětí U_2 po filtraci*

Číslicové filtry lze popsat pomocí:

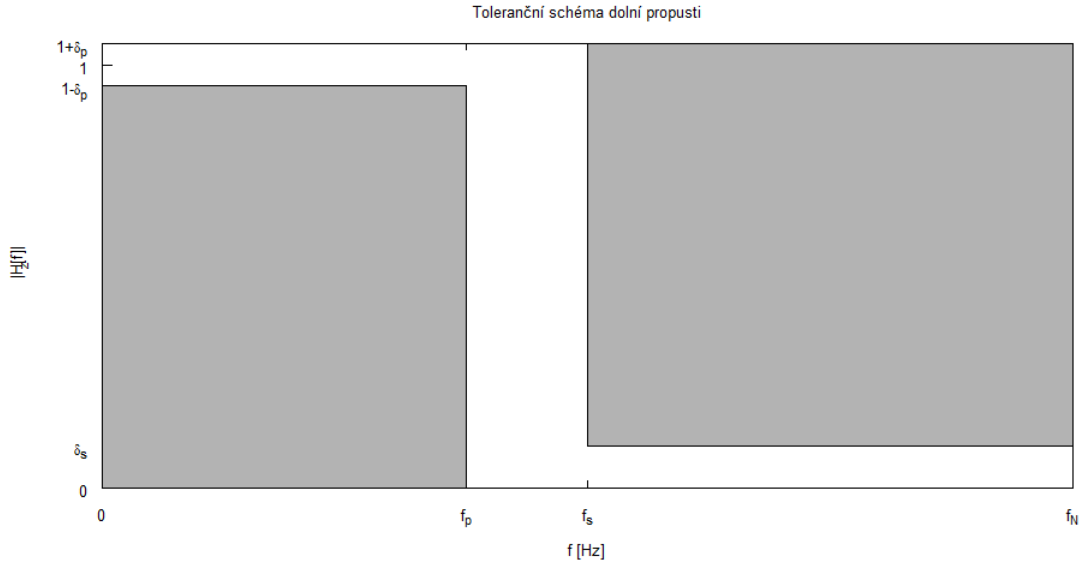
- Diferenční rovnice
- Impulzní odezvy (odezva na jednotkový impulz)
- Frekvenční odezvy
- Přenosové funkce

Tyto filtry lze realizovat dvěma způsoby a to filtry FIR (Finite Impuls Response) a IIR (Infinite Impulse Response) podle požadavků. Požadavky na vlastnosti filtrů jsou většinou definovány pomocí tzv. tolerančního schématu.

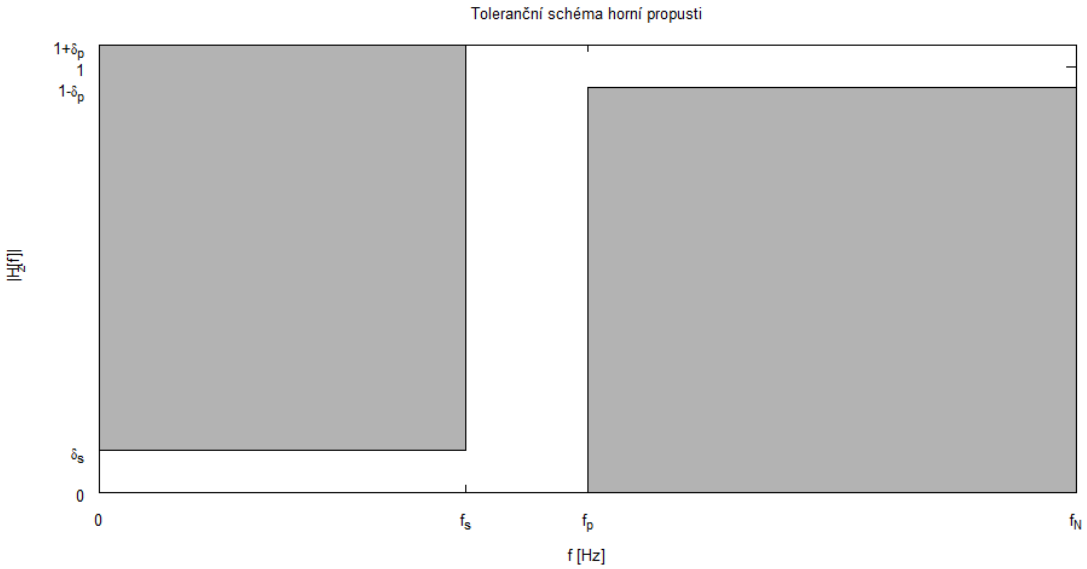
10.1 FIR filtry

FIR filtry jsou filtry s konečnou impulzní odezvou. Největší výhodou těchto filtrů je, že u nich můžeme dosáhnout lineární fáze v celém kmitočtovém pásmu. Abychom dosáhli lineární fáze je potřeba splnit některou z podmínek symetrie. Pokud je skupinové i fázové zpoždění konstantní, jedná se o pozitivní symetrii, pokud je konstantní pouze skupinové zpoždění, jedná se o negativní symetrii (antisymetrii). Další výhodou FIR filtrů je, že jsou vždy stabilní (nedojde k rozkmitání), to je způsobeno tím, že všechny póly přenosové funkce *H[z]*

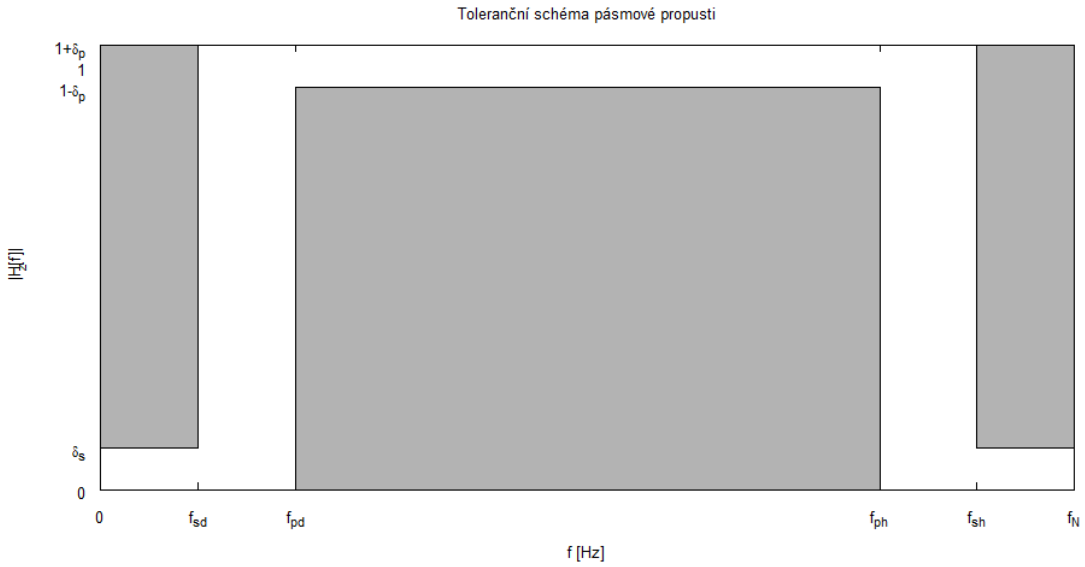
v z-rovině leží v počátku. Mezi nevýhody patří možnost nesplnění zadaných požadavků přesně, vyšší řád filtru než u IIR a při návrhu nelze využít analogie s analogovými filtry. Toleranční schémata FIR filtrů jsou uvedena na obrázcích 10.2, 10.3, 10.4 a 10.5.



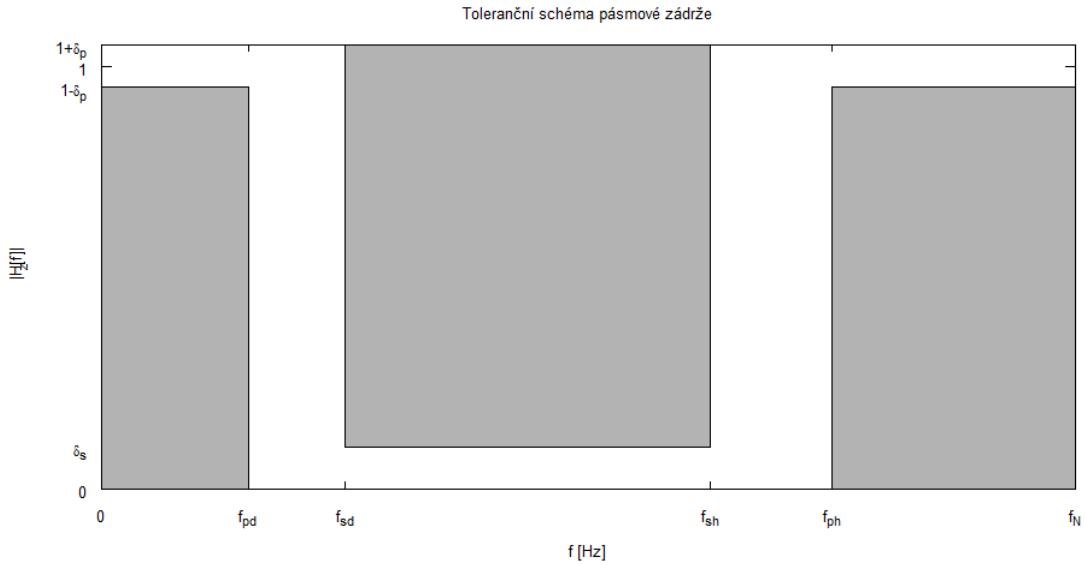
Obrázek 10.2: *Toleranční schéma dolní propusti FIR*



Obrázek 10.3: *Toleranční schéma horní propusti FIR*



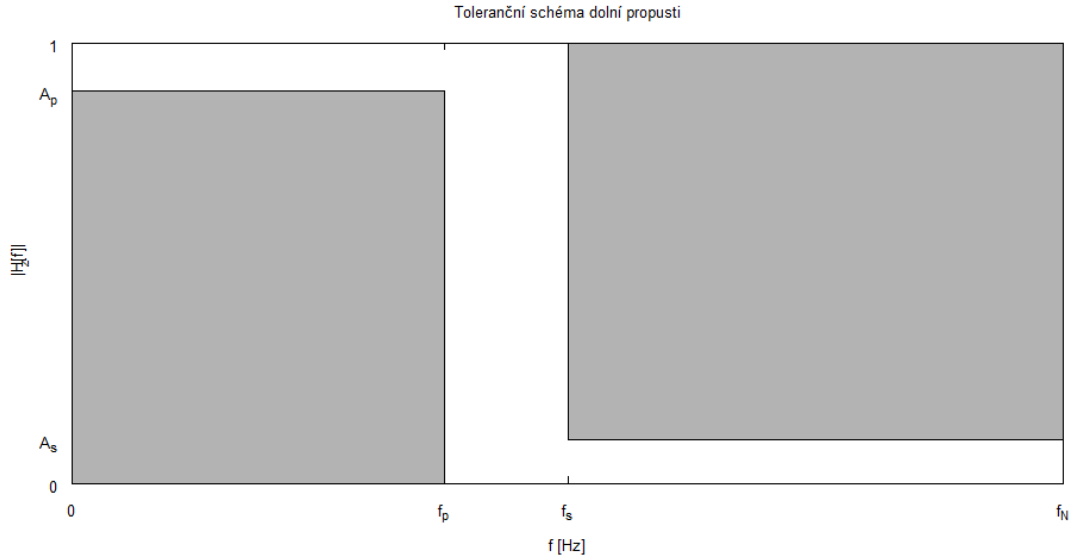
Obrázek 10.4: *Toleranční schéma pásmové propusti FIR*



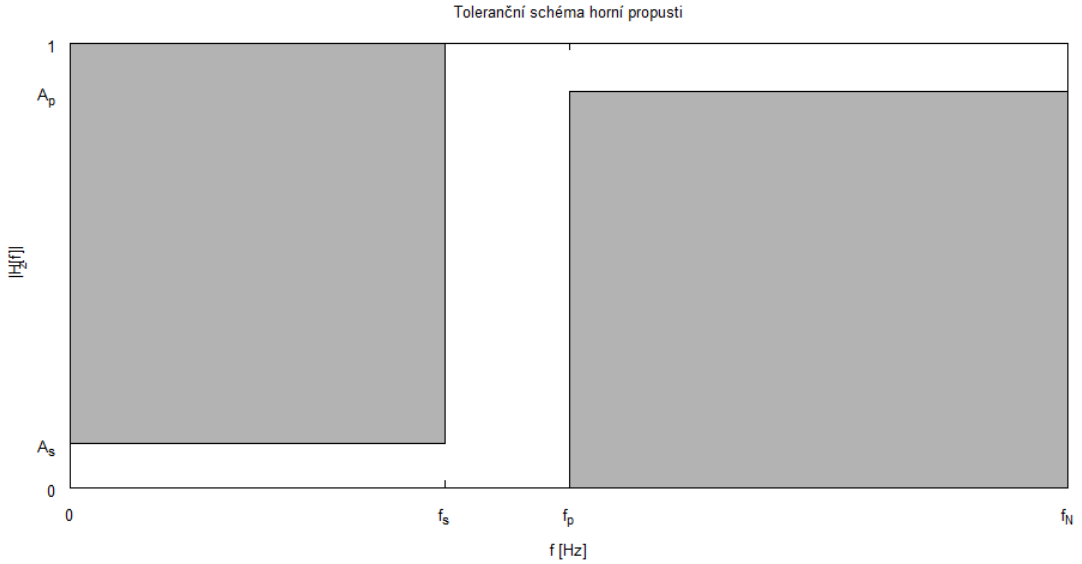
Obrázek 10.5: *Toleranční schéma pásmové zádrže FIR*

10.2 IIR filtry

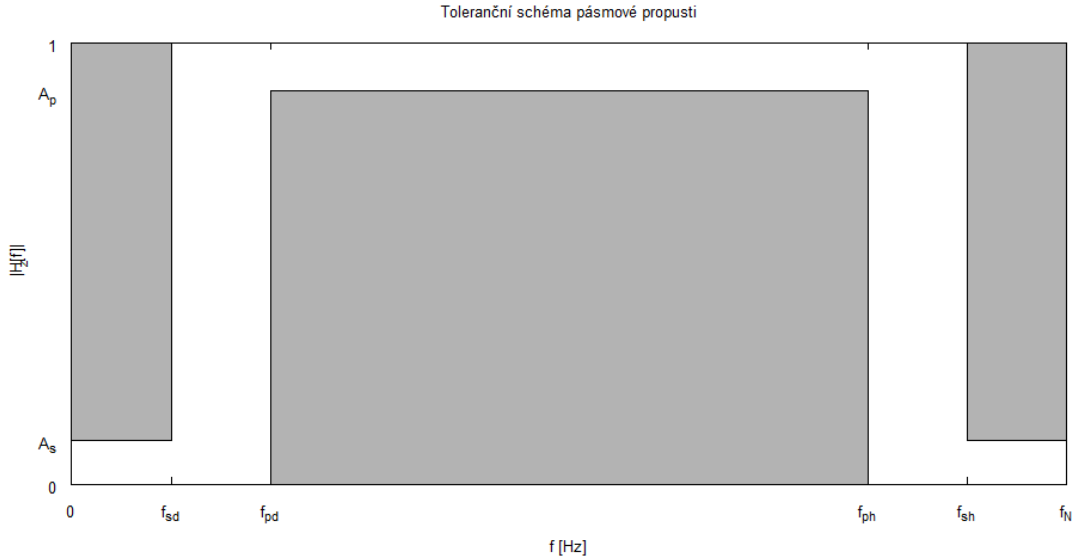
IIR filtry jsou filtry s nekonečnou impulzní odezvou. Nekonečná impulzní odezva je dána zpětnovazební smyčkou. Mezi největší výhody IIR filtrů patří snadný návrh, při kterém lze využít analogie s analogovými filtry a nižší rád filtru (menší nároky na paměť). Naopak mezi nevýhody patří nelineární fázová charakteristika (nelineární zpoždění) a možnost nestability filtru. Abychom docílili stability filtru, je potřeba umístit póly přenosové funkce *H[z]* dovnitř jednotkové kružnice. Toleranční schémata IIR filtrů jsou uvedena na obrázcích 10.6, 10.7, 10.8 a 10.9.



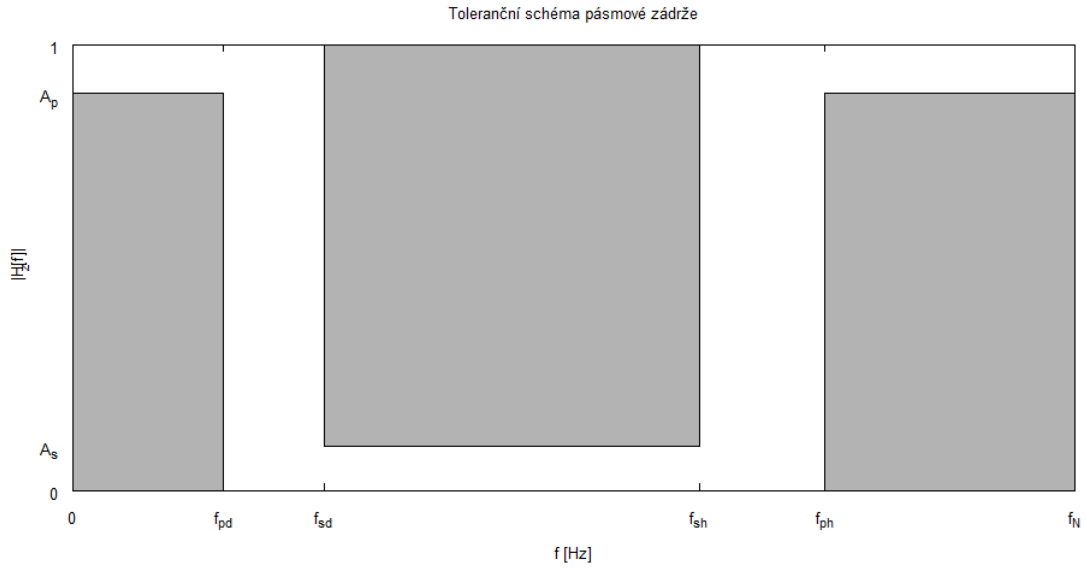
Obrázek 10.6: *Toleranční schéma dolní propusti IIR*



Obrázek 10.7: *Toleranční schéma horní propusti IIR*



Obrázek 10.8: *Toleranční schéma pásmové propusti IIR*



Obrázek 10.9: *Toleranční schéma pásmové zádrže IIR*

10.3 Návrh filtrů

Před návrhem samotného filtru je nutné analyzovat požadavky na filtr a podle těchto požadavků se rozhodnout, který způsob realizace bude pro daný případ vhodnější. Při návrhu FIR filtrů máme tři možnosti jak postupovat (metoda váhových funkcí, metoda vzorkování frekvenční charakteristiky a optimalizační metoda Parks-McClellan). Návrh IIR filtrů je podstatně jednodušší, protože můžeme využít analogie s analogovými filtry. Pro návrh IIR filtrů jsou v Octave implementovány speciální funkce, pomocí kterých hned získáme výsledné parametry filtrů. Jako druhou možnost návrhu můžeme využít postupný přechod z digitální oblasti do analogové a zpět.

10.3.1 Návrh FIR filtrů

Jak bylo zmíněno výše, pro návrh FIR filtrů můžeme využít tři metody, my si probereme pouze první dvě. Obě tyto metody využívají k návrhu filtru v Octave vhodná okna. Mezi nejzákladnější typy oken patří pravoúhlé, Hammingovo, Blackmanovo a Hannovo. Každé z oken má dvě základní vlastnosti. První vlastnost je šířka hlavního laloku, která ovlivňuje strmost přenosové charakteristiky v přechodovém pásmu (čím je šířka laloku menší, tím je strmost větší), druhá vlastnost je amplituda postranních laloků (čím je amplituda menší, tím je vyšší útlum v závěrném pásmu). Nelze současně dosáhnout malé šířky hlavního laloku a malé amplitudy postranních laloků, proto je vždy nutné volit určitý kompromis. Parametry základních oken jsou uvedeny v tabulce 10.1.

Okno	Vrchol postranního laloku	Šířka hlavního laloku
Obdélníkové	-13 dB	4π/L
Bartlett	-25 dB	8π/L
Hanning	-31 dB	8π/L
Hamming	-41 dB	8π/L
Blackman	-57 dB	12π/L

Tabulka 10.1: *Vlastnosti základních oken*

- **Metoda váhových funkcí (oken)**

Metoda vychází ze vzájemného vztahu mezi impulzní charakteristikou a přenosovou funkcí. Jelikož má přenosová funkce FIR filtru ve jmenovateli pouze N-1 násobný nulový kořen, stačí pouze zjistit koeficienty čitatele přenosové funkce. Tyto koeficienty jsou zároveň vzorky impulzní charakteristiky. Výpočet těchto koeficientů spočívá ve zkrácení nekonečné (ideální) impulzní odezvy na konečný počet vzorků a v omezení vhodným oknem. Jako příklad si ukážeme návrh dolní propusti pomocí Hammingova okna, které je přímo ve funkci `fir1` přednastaveno. Příklady návrhů ostatních typů filtrů s jinými okny jsou uvedeny v příloze H na dvd formou m-souborů.

a) Výpočet mezní frekvence f_c

$$f_c = \frac{\left(f_p + f_s\right)}{2}\left[Hz; Hz, Hz\right] \tag{10.1}$$

b) Normování mezní frekvence pomocí Nyquistovy frekvence f_N do intervalu <0,1>

$$\omega_n = \frac{f_c}{f_N}\left[-; Hz, Hz\right] \tag{10.2}$$

c) Zvolení řádu filtru N, pro horní propust a pásmovou zádrž musí být N sudé

d) Návrh filtru pomocí funkce `fir1`. Výsledkem této funkce jsou koeficienty b_z přenosové funkce *H[z]*. Výchozí nastavení této funkce je určené pro návrh dolní propusti pomocí Hammingova okna, proto stačí zadat pouze parametry N a ω_n, pokud je ω_n vektor dojde k návrhu pásmové propusti. Pro návrh horní propusti, nebo pásmové zádrže je nutné zadat typ filtru `'high'`, `'stop'`.

```
b_z = fir1(N,w_n)
b_z = fir1(N,w_n,'typ_filtru',okno)
```

e) Po vykreslení přenosové charakteristiky vidíme, zda filtr splňuje zadané toleranční schéma, pokud tomu tak není, provedeme návrh pro vyšší řád filtru nebo můžeme použít jiný typ okna.

- **Metoda vzorkování frekvenční charakteristiky**

Tato metoda je založena na vzorkování kmitočtové charakteristiky filtru. Vzorkování provádíme v L bodech intervalu $<0,f_{vz}>$. Výsledné koeficienty přenosové funkce získáme zpětnou Fourierovou transformací vzorkovaného průběhu. Pro návrh využijeme funkce `fir2`, která určuje výsledné koeficienty b_z přenosové funkce *H[z]*. Příklad návrhu provedeme pro horní propust. Návrhy ostatních typů filtrů jsou uvedeny v příloze H na dvd.

a) Výpočet mezní frekvence f_c, pro výpočet použijeme stejný vzorec jako v předchozím případě (10.1)

b) Normování mezní frekvence pomocí Nyquistovy frekvence f_N do intervalu $<0,1>$, normování provedeme podle vzorce (10.2)

c) Vytvoření vektoru frekvenčních bodů *meze_f*, jednotlivé frekvence vektoru jsou normovány Nyquistovou frekvencí

```
meze_f = [0 w_n w_n 1];
```

d) Vytvoření vektoru *meze_m*, který obsahuje požadované velikosti přenosu na daných frekvenčních bodech specifikovaných vektorem *meze_f*

```
meze_M = [0 0 1 1];
```

e) Zvolení řádu filtru N a určení vektoru koeficientů impulzní odezvy L, pro horní propust a pásmovou zádrž musí být N sudé

$$L = N + 1\left[-;-,-\right] \tag{10.3}$$

f) Vytvoření váhové posloupnosti w_p pro zadané okno

```
w_p = hamming(L);
```

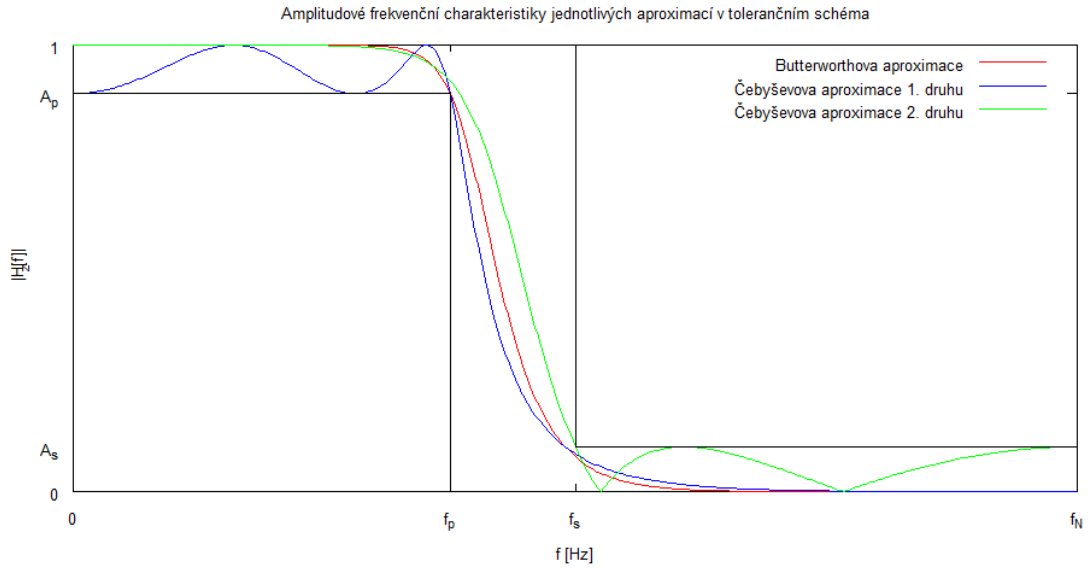
g) Návrh filtru pomocí funkce `fir2`, výsledkem této funkce jsou koeficienty b_z přenosové funkce *H[z]*.

```
b_z = fir2(N, meze_f, meze_M, w_p);
```

h) Kontrolu návrhu opět provedeme vykreslením příslušných charakteristik, pokud některá z charakteristik neodpovídá zadaným požadavkům, provedeme návrh znovu s vyšším řádem filtru nebo použijeme jiný typ okna.

10.3.2 Návrh IIR filtrů

Pro návrh IIR filtrů si ukážeme dvě metody. První metoda je pomocí funkcí v Octave, která poskytuje pouze výsledné parametry filtrů a druhá metoda, která poskytuje průběžné parametry filtrů podle toho, ve které oblasti se nacházíme. U druhé metody můžeme pomocí průběžných výsledků kontrolovat správnost návrhu a také můžeme sledovat, jak se jednotlivé parametry mění a jestli tyto změny odpovídají teorii. Obě metody využívají přechodu k normované dolní propusti (analogový filtr), ze které se následně navrhuje zadaný číslicový filtr. Při návrhu můžeme využít tří základních aproximací (Butterworthova, Čebyševova 1. druhu a Čebyševova 2. druhu). V případě Butterworthovy aproximace je amplitudová charakteristika v propustném pásmu maximálně plochá a celý průběh je monotónní. U Čebyševovy aproximace 1. druhu dochází ke zvlnění amplitudové charakteristiky v propustném pásmu, v přechodovém a závěrném pásmu monotónně klesá. Čebyševova aproximace 2. druhu (inverzní Čebyšev) způsobuje zvlnění amplitudové charakteristiky v závěrném pásmu. Ukázka průběhů amplitudových frekvenčních charakteristik pro jednotlivé aproximace je zobrazena na obrázku 10.10.



Obrázek 10.10: *Amplitudové frekvenční charakteristiky jednotlivých aproximací v tolerančním schéma dolní propusti*

- **Přímá metoda pomocí funkcí v Octave**

Při návrhu vycházíme z požadované modulové frekvenční charakteristiky, která je zadaná tolerančním schématem. Pro všechny typy filtrů (DP, HP a PP) je postup obdobný, proto si v tomto textu ukážeme pouze návrh dolní propusti. Návrhy ostatních typů filtrů jsou uvedeny v příloze H na dvd ve formě m-souborů. Pásmová zádrž je implementována pouze pro Butterwothovu aproximaci a i u této aproximace není návrh korektní, proto si návrh PZ ukážeme pomocí druhé metody.

a) Pro jednotlivé mezní frekvence f_p a f_s vypočteme normované frekvence ω_p a ω_s.

$$\omega_p = \frac{f_p}{f_N}\left[-; Hz, Hz\right] \tag{10.4}$$

$$\omega_s = \frac{f_s}{f_N}\left[-; Hz, Hz\right] \tag{10.5}$$

b) Přechod k normované dolní propusti, zjištění jejího řádu N_{NDP} a normované charakteristické frekvence ω_n pro zadanou aproximaci (Butterworthova, Čebyševova 1. druhu a Čebyševova 2. druhu).

```
[N_NDP, w_n] = buttord(w_p, w_s, a_p, a_s)
[N_NDP, w_n] = cheb1ord(w_p, w_s, a_p, a_s)
[N_NDP, w_n] = cheb2ord(w_p, w_s, a_p, a_s)
```

c) Zjištění koeficientů b_z a a_z přenosové funkce *H[z]* číslicového filtru.

```
[b_z, a_z] = butter(N_NDP, w_n)
[b_z, a_z] = cheby1(N_NDP, a_p, w_n)
[b_z, a_z] = cheby2(N_NDP, a_s, w_n)
```

d) Zjištění nulových bodů z_0 a pólů z_{infty} přenosové funkce *H[z]*

```
z_0 = roots(b_z)
z_infty = roots(a_z)
```

e) Nyní máme filtr navržený a po vykreslení příslušných charakteristik vidíme, jestli filtr splňuje zadané požadavky. Pokud filtr nesplňuje některý z požadavků je potřeba provést návrh znovu, například s vyšším řádem filtru. Postup vykreslování charakteristik včetně tolerančního schéma je uveden v m-souborech v příloze H na dvd.

- **Postupná metoda pomocí přechodu z digitální oblasti do analogové a zpět**

Při návrhu opět vycházíme, jako v předchozím případě, z modulové frekvenční charakteristiky. Postup je takový, že nejprve provedeme transformaci frekvenčních požadavků do analogové oblasti a poté už probíhá návrh jako u analogového filtru. Abychom dosáhli stabilního filtru, je nutné, aby póly přenosové funkce *H(s)* ležely vlevo od imaginární osy. Posledním krokem je transformace přenosové funkce do číslicové oblasti. Jako příklad si ukážeme návrh pásmové zádrže pomocí Butterworthovy aproximace. Pro ostatní typy filtrů a aproximací jsou uvedeny vzorce v příloze A. Ukázkové příklady jsou obsaženy v příloze H na dvd.

a) Normování mezních frekvencí pomocí Nyquistovy frekvence f_N do intervalu <0,π>

$$\varpi_{pd} = \pi \times \frac{f_{pd}}{f_N}\left[-;-, Hz, Hz\right] \tag{10.6}$$

$$\varpi_{sd} = \pi \times \frac{f_{sd}}{f_N}\left[-;-, Hz, Hz\right] \tag{10.7}$$

$$\varpi_{ph} = \pi \times \frac{f_{ph}}{f_N}\left[-;-,Hz,Hz\right] \tag{10.8}$$

$$\varpi_{sh} = \pi \times \frac{f_{sh}}{f_N}\left[-;-,Hz,Hz\right] \tag{10.9}$$

b) Dopředná frekvenční transformace

- Výpočet mezních úhlových frekvencí analogové předlohy filtru (předzkreslení pro bilineární transformaci)

$$\omega_{pd} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{pd}}{2}\left[s^{-1};s,-\right] \tag{10.10}$$

$$\omega_{sd} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{sd}}{2}\left[s^{-1};s,-\right] \tag{10.11}$$

$$\omega_{ph} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{ph}}{2}\left[s^{-1};s,-\right] \tag{10.12}$$

$$\omega_{sh} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{sh}}{2}\left[s^{-1};s,-\right] \tag{10.13}$$

- Určení mezních úhlových frekvencí normované dolní propusti

$$\Omega_{pd} = \frac{\omega_{pd} \times \left(\omega_{ph} - \omega_{pd}\right)}{\omega_{pd}^2 - \omega_{pd} \times \omega_{ph}} = -1\left[-;s^{-1},s^{-1}\right] \tag{10.14}$$

$$\Omega_{ph} = \frac{\omega_{ph} \times \left(\omega_{ph} - \omega_{pd}\right)}{\omega_{ph}^2 - \omega_{pd} \times \omega_{ph}} = 1\left[-;s^{-1},s^{-1}\right] \tag{10.15}$$

$$\Omega_{sd} = \frac{\omega_{sd} \times \left(\omega_{ph} - \omega_{pd}\right)}{\omega_{sd}^2 - \omega_{pd} \times \omega_{ph}}\left[-;s^{-1},s^{-1},s^{-1}\right] \tag{10.16}$$

$$\Omega_{sh} = \frac{\omega_{sh} \times \left(\omega_{ph} - \omega_{pd}\right)}{\omega_{sh}^2 - \omega_{pd} \times \omega_{ph}}\left[-;s^{-1},s^{-1},s^{-1}\right] \tag{10.17}$$

V případě normované dolní propusti je vždy $\Omega_p=1$.

$$\Omega_s = \min\left(\left|\Omega_{sd}\right|,\left|\Omega_{sh}\right|\right) \tag{10.18}$$

- Výpočet řádu normované dolní propusti N_{NDP} a výpočet charakteristické frekvence Ω_n pro danou aproximaci, v našem případě se jedná o Butterworthovu aproximaci

$$N_{NDP} \geq \frac{\log \sqrt{\frac{\frac{1}{A_s^2}-1}{\frac{1}{A_p^2}-1}}}{\log\left(\frac{\Omega_s}{1}\right)}\left[-;-,-,-,\right] \tag{10.19}$$

Jelikož řád filtru nemusí vyjít jako celé číslo, provádíme vždy zaokruhlení směrem nahoru, v Octave k tomu slouží funkce `ceil`.

$$\Omega_{np} = \frac{\Omega_p}{\left(\frac{1}{A_p^2}-1\right)^{\frac{1}{2\times N_{NDP}}}}\left[-;-,-,-\right] \tag{10.20}$$

$$\Omega_{ns} = \frac{\Omega_s}{\left(\frac{1}{A_s^2}-1\right)^{\frac{1}{2\times N_{NDP}}}}\left[-;-,-,-\right] \tag{10.21}$$

Jako Ω_n vybereme jednu z vypočtených hodnot podle toho, na které frekvenci chceme přesně splnit specifikaci. V našem případě $\Omega_n = \Omega_{ns}$.

- Určení nul p_0, pólů p_∞ a násobného koeficientu k_p přenosové funkce *H(p)* pro normovanou dolní propust pomocí funkce `buttap_octave`

```
[p_0,p_infty,k_p]=buttap_octave(N_NDP)
```

- Získání koeficientů b_p a a_p přenosové funkce *H(p)*

```
b_p = k_p*poly(p_0)
a_p = poly(p_infty)
```

c) Zpětná frekvenční transformace *H(p)* →*H(s)*

- Určení charakteristických frekvencí analogové předlohy filtru

$$\omega_{nd} = \frac{-\frac{\left(\omega_{ph}-\omega_{pd}\right)}{\Omega_n}+\sqrt{\frac{\left(\omega_{ph}-\omega_{pd}\right)^2}{\Omega_n^2}+4\times\omega_{pd}\times\omega_{ph}}}{2}\left[s^{-1};s^{-1},s^{-1},-\right] \tag{10.22}$$

$$\omega_{nh} = \frac{\frac{\left(\omega_{ph}-\omega_{pd}\right)}{\Omega_n}+\sqrt{\frac{\left(\omega_{ph}-\omega_{pd}\right)^2}{\Omega_n^2}+4\times\omega_{pd}\times\omega_{ph}}}{2}\left[s^{-1};s^{-1},s^{-1},-\right] \tag{10.23}$$

- Určení nul s_0, pólů s_∞ a násobného koeficientu k_s přenosové funkce *H(s)* pomocí funkce `sftransmoje`

```
[s_0,s_infty,k_s]=sftransmoje(p_0, p_infty, k_p,
[omega_nd omega_nh], 0)
```

- Získání koeficientů b_s a a_s přenosové funkce *H(s)*

```
b_s = poly(s_0)*k_s
a_s = poly(s_infty)
```

d) Bilineární transformace *H(s)* →*H[z]*
   - Určení nul z_0, pólů z_∞ a násobného koeficientu k_z přenosové funkce *H[z]* pomocí funkce `bilinear`

```
[z_0,z_infty,k_z]=bilinear(s_0,s_infty,k_s,T_vz)
```

   - Získání koeficientů b_z a a_z přenosové funkce *H[z]*

```
b_z = k_z*poly(z_0)
a_z = poly(z_infty)
```

e) Jako u přímé metody máme nyní filtr navržený a po vykreslení příslušných charakteristik vidíme, jestli filtr splňuje zadané požadavky. Pokud filtr nesplňuje některý z požadavků, je potřeba provést návrh znovu například s vyšším řádem filtru. Postup vykreslování charakteristik včetně tolerančního schéma je uveden v m-souborech v příloze H na dvd.

Při zpracování této kapitoly jsem čerpal z literatury [10], [11], [12], [13], [14].

11 Práce s definovanou délkou slova

Reálné číslicové systémy pracují s omezenou přesností (s pevnou délkou slova), tato přesnost je dána velikostí registrů příslušného číslicového systému. Octave naopak pracuje v tzv. plné (dvojité) přesnosti. Z tohoto důvodu je vhodné omezit přesnost i ve výpočetním softwaru. Tímto omezením docílíme toho, že návrh v Octave přímo odpovídá reálné situaci. Funkce použité v této kapitole jsou součástí základní verze Octave a proto není nutné doinstalovávat rozšiřující balíčky.

11.1 Práce s bity

Pro práci s bity v Octave slouží následující funkce:

`bitand(A, B)` – funkce provede AND mezi A a B v jednotlivých bitech, A a B musí být nezáporná celá čísla

`bitor(A, B)` – funkce provede OR mezi A a B v jednotlivých bitech, A a B musí být nezáporná celá čísla

`bitxor(A, B)` – funkce provede XOR mezi A a B v jednotlivých bitech, A a B musí být nezáporná celá čísla

`bitset(A, N, VAL)` – funkce provede nastavení bitu N na 0 nebo 1 (podle hodnoty VAL) v čísle A, A musí být nezáporné celé číslo

`bitget(A, N)` – funkce vrátí hodnotu bitu N v čísle A, A musí být nezáporné celé číslo

`bitcmp(A, K)` – funkce vrátí K bitový doplněk čísla A, A musí být nezáporné celé číslo

`bitshift(A, K, N)` – funkce provede K bitový posun čísla A (pokud je K kladné jedná se o posun doleva, pokud je K záporné jedná se o posun doprava), parametr N udává, kolik bitů chceme zachovat, pokud parametr N nezadáme, dojde pouze k posunu a výsledné číslo je kompletní

Příklady použití jednotlivých funkcí jsou uvedeny v příloze I na dvd.

11.2 Zobrazení čísel

Pro zobrazení číselných hodnot můžeme využít libovolnou číselnou soustavu. V Octave se jako výchozí číselná soustava používá desítková, pro převod do ostatních číselných soustav můžeme využít funkci `dec2base(A, B)`, kde hodnota A je číslo v desítkové soustavě a B je základ soustavy, do které chceme převod provést. Číslo v dané soustavě můžeme vyjádřit pomocí dvou formátů (vyjádření s pevnou a pohyblivou řádovou čárkou).

11.2.1 Čísla s pevnou řádovou čárkou

Každé reálné číslo A lze vyjádřit jako součet mocnin základu Z vynásobených příslušnou číslicí a.

$$A = a_m \times Z^m + \ldots + a_1 \times Z^1 + a_0 \times Z^0 + a_{-1} \times Z^{-1} + a_{-n} \times Z^{-n} \tag{11.1}$$

$$A = \sum_{i=-\infty}^{m} a_i \times Z^i \tag{11.2}$$

Zápis podle vzorce 11.1 je pro reálné číslicové systémy nepoužitelný, protože registry nemají nekonečnou délku. Z toho důvodu můžeme tvar přepsat pro zadanou délku registru b a základ binární soustavy B následovně.

$$A = a_{b-n-1} \times B^{b-n-1} + \ldots + a_1 \times B^1 + a_0 \times B^0 + a_{-1} \times B^{-1} + a_{-n} \times B^{-n} \tag{11.3}$$

$$A = \sum_{i=-n}^{b-n-1} a_i \times B^i \tag{11.4}$$

b-n-1 – určuje největší zobrazitelnou váhu celočíselné části

a_{-n} – koeficient s nejmenší váhou

Číslo je tedy určeno s přesností 2^{-n} , $(|A-a_m \ldots a_1 a_0, a_{-1} \ldots a_{-n}| < 2^{-n})$. Podle umístění řádové čárky lze číslo rozdělit na celočíselnou část a zlomkovou část. Nejčastější umístění řádové čárky je za prvním bitem zleva (vyjádření mantisou) nebo za posledním bitem zprava (celočíselné vyjádření). V číslicových systémech se při práci s pevnou řádovou čárkou obvykle používá vyjádření mantisou. Po aritmetické operaci je výsledek zaokrouhlen nebo omezen na definovaném nejméně platném bitu určeném délkou registru. Hlavní výhodou použití čísel v pevné řádové čárce v číslicových systémech je jednodušší realizace aritmetických operací. Nevýhodou je naopak malý rozsah zobrazovaných čísel.

V Octave je jako výchozí zobrazení čísel nastaveno zobrazení v pevné řádové čárce na platných 5 číslic. Pro nastavení zobrazovacího formátu slouží příkaz `format`. Následující tabulka 11.1 ukazuje základní formáty zobrazení čísel v Octave. Příklady jsou uvedeny v příloze I na dvd.

Formát	Popis formátu
format	výchozí nastavení v Octave, stejné jako short
format short	zobrazení čísla v pevné řádové čárce na 5 platných číslic
format short e	zobrazení čísla v pohyblivé řádové čárce na 5 platných číslic
format long	zobrazení čísla v pevné řádové čárce na 15 platných číslic
format long e	zobrazení čísla v pohyblivé řádové čárce na 15 platných číslic

Tabulka 11.1: *Základní formáty zobrazení čísel v Octave*

Pozor v Matlabu platí pro stejné formáty jiná pravidla. Formát `short` zobrazuje číslo v pevné řádové čárce na 4 desetinná místa. Formát `long` zobrazuje čísla v pevné řádové čárce na 15 desetinných míst. Totéž platí pro formáty `short e` a `long e` pouze jsou čísla zobrazena v pohyblivé řádové čárce.

Ukázka zobrazení čísla π pro předchozí formáty je uvedena na obrázku 11.1.

```
>>> format short
>>> pi
ans =  3.1416
>>> format short e
>>> pi
ans = 3.1416e+000
>>> format long
>>> pi
ans =  3.14159265358979
>>> format long e
>>> pi
ans = 3.14159265358979e+000
```

Obrázek 11.1: *Zobrazení čísla π v různých formátech*

Zobrazení celých kladných a záporných čísel v pevné řádové čárce

Pokud chceme zobrazovat pouze čísla kladná, tak máme k dispozici celou délku registru. Rozsah zobrazovaných čísel tedy závisí na velikosti registru (velikost registru 8 bitů nám umožní zobrazovat čísla v rozsahu $<0,2^b-1>$, kde b je počet bitů registru). Pokud ovšem chceme zároveň zobrazovat i čísla záporná, tak musíme využít jednoho z následujících zobrazení záporných čísel.

- Přímý kód se znaménkem

Vyjádření čísla přímým kódem se znaménkem nám omezí rozsah zobrazovaných čísel, protože nejvyšší bit je určen pro specifikaci znaménka (0 odpovídá kladnému znaménku, 1 zápornému). Rozsah zobrazovaných čísel je poté následující $<-2^{b-1}+1,2^{b-1}-1>$. Ve vyjádření přímým kódem existují dvě nuly (kladná a záporná).

Příklad zobrazení čísla 5 a -5 přímým kódem pro 8-bitový registr:

$(5)_{10}= (0000\ 0101)_{PK}$

$(-5)_{10}= (1000\ 0101)_{PK}$

- Jednotkový doplněk

Při vyjádření čísel jednotkovým doplňkem jsou kladná čísla vyjádřena stejně jako pomocí přímého kódu a záporná čísla jsou vyjádřena jednotkovým doplňkem (^1A). Výpočet jednotkového doplňku provedeme tak, že od maximální možné hodnoty (číslo obsahuje samé jedničky) odečteme číslo A. Výsledný jednotkový doplněk ^1A má oproti číslu A všechny bity negované. Nejvyšší bit opět specifikuje znaménko jako u přímého kódu. Rozsah zobrazovaných čísel je stejný jako u přímého kódu $<-2^{b-1}+1, 2^{b-1}-1>$. Ve vyjádření opět existují dvě nuly.

Vyjádření čísla A= -41 jednotkovým doplňkem (8-bitový registr):

$^1A= 2^b-1+A = 2^8-1-41= 256 -1-41= 214$

$A = (41)_{10}= (0010\ 1001)_{PK}$

$^1A = (214)_{10}= (1101\ 0110)_{JD}$

- Dvojkový doplněk

Kladná čísla jsou opět vyjádřena pomocí přímého kódu, záporná čísla jsou vyjádřena pomocí dvojkového doplňku (^2A). Dvojkový doplněk získáme tak, že všechny bity čísla A znegujeme (vypočítáme jednotkový doplněk) a poté k nim přičteme jedničku. Nejvyšší bit je opět znaménkový. Rozsah je při vyjádření dvojkovým doplňkem $<-2^{b-1}, 2^{b-1}-1>$. Ve vyjádření je už pouze jedna nula.

Vyjádření čísla A= -41 dvojkovým doplňkem (8-bitový registr):

$^2A= 2^b-1+A+1= {}^1A+ 1= 214+1=215$

$^2A = (215)_{10}= (1101\ 0111)_{DD}$

- Kód s posunutou nulou

Princip je založen na lineárním posunu nuly do středu rozsahu. Pro 8-bitový registr je střed rozsahu v $2^{b-1}-1$, (127). Binární hodnota nuly tedy je $(0111\ 1111)_2$, maximální kladné číslo je $(1111\ 1111)_2$ a maximální záporné číslo je $(0000\ 0000)_2$. Nevýhodou tohoto zobrazení je odlišné zobrazení kladných čísel od binární reprezentace. Hlavní výhodou je aplikace u pohyblivé řádové čárky pro vyjádření exponentu.

Vyjádření čísla A= 41 a B= -41 pomocí posunutí (8-bitový registr):

$^pA= 2^{b-1}-1+ A= 127+41=168$

$^pA= (168)_{10}= (1010\ 1000)_{PN}$

$^pB= 2^{b-1}-1+ B= 127-41=86$

$^pB= (86)_{10}= (0101\ 0110)_{PN}$

Ukázky vyjádření čísel v pevné řádové čárce jsou uvedeny v příloze I na dvd.

11.2.2 Čísla s pohyblivou řádovou čárkou

Pro práci s čísly, která se pohybují ve velkém rozsahu, je výhodnější použít vyjádření v pohyblivé řádové čárce (floating point). Zápis čísla v pohyblivé řádové čárce se skládá ze tří částí (mantisa, základ číselné soustavy a exponent).

$$A = m \times Z^e \tag{11.5}$$

m – mantisa

Z – základ číselné soustavy

e – exponent

Vyjádření čísel v pohyblivé řádové čárce můžeme provést pomocí dvou formátů. Vyjádření v jednoduché přesnosti (single) nebo vyjádření v dvojité přesnosti (double). V prvním případě je číslo uloženo pomocí 32 bitů, kde 1 bit tvoří znaménko, 8 bitů je určeno pro exponent a zbývajících 23 bitů tvoří mantisu. Vyjádření čísel v jednoduché přesnosti je zobrazeno na obrázku 11.2. Při vyjádření v dvojité přesnosti je číslo uloženo pomocí 64 bitů, kde 1 bit je opět znaménkový, 11 bitů je určeno pro exponent a zbývajících 52 bitů tvoří mantisu. Vyjádření čísel v dvojité přesnosti je zobrazeno na obrázku 11.3. Mantisa bývá vyjadřována v přímém kódu a v normovaném tvaru (normovaný tvar – první nenulová číslice je před řádovou čárkou a platí $1 \leq m \leq Z$). První bit mantisy se neukládá (reprezentuje jedničku před desetinnou čárkou). Přesnost zobrazení je dána počtem bitů mantisy. Rozsahy zobrazení pro jednotlivé přesnosti jsou uvedeny v tabulce 11.2. Největší nevýhodou pohyblivé řádové čárky je složitější a časově náročnější provádění aritmetických operací. Výhodou je naopak možnost zobrazení čísel ve velkém rozsahu.

znaménko	exponent	mantisa
31	30 - 23	22 - 0

Obrázek 11.2: *Vyjádření čísla v jednoduché přesnosti*

znaménko	exponent	mantisa
63	62 - 52	51 - 0

Obrázek 11.3: *Vyjádření čísla v dvojité přesnosti*

přesnost	posun	rozsah exponentu	rozsah čísla
jednoduchá (single)	127	-126 až 127	2^{-126} až 2^{127}
dvojitá (double)	1023	-1022 až 1023	2^{-1022} až 2^{1023}

Tabulka 11.2: *Rozsahy zobrazení pro jednotlivé přesnosti*

Příklad zápisu čísla -258,125 pomocí pohyblivé řádové čárky v jednoduché přesnosti:

- Převod čísla do binární soustavy (zvlášť převedeme celou a desetinou část)
  $(258)_{10} = (1\ 0000\ 0010)_2$

$(0,125)_{10}= (0,001)_2$

- Normalizovaný tvar

  $(258,125)_{10}= 1,00000010001*2^8$

- Vyjádření exponentu posunutím

  $exp= 2^{b-1}-1+ 8= 127+8=135$

  $exp= (135)_{10}= (1000\ 0111)_{PN}$

- Vyjádření čísla -258,125 v jednoduché přesnosti

znaménko	exponent	mantisa
1	1000 0111	00000010001000000000000

Příklad vyjádření čísla v jednoduché přesnosti je uveden v příloze I na dvd.

11.3 Práce se soubory

Funkce Octave pro souborový vstup a výstup umožňuje načíst data uložená v jiném formátu přímo do Octave nebo zapsat data vytvořená v Octave ve formátu, který je vhodný pro jiný program. Pro práci se soubory můžeme využít v Octave dvou režimů (textový a binární). V případě binárního režimu jsou data z paměti počítače ukládána do souboru v nezměněné podobě, zatímco u textového režimu dochází k převodu číselných hodnot na řetězce (převod do textové podoby), jedná se tedy o formátovaný zápis. U načítání dat ze souboru platí totéž (v binárním režimu jsou opět data načtena v nezměněné podobě, kdežto v textovém režimu je nutná konverze). Z těchto poznatků vyplývají výhody a nevýhody jednotlivých režimů. Hlavní výhodou textového režimu je, že data jsou v textové podobě. Nevýhodou naopak je, že operace s daty v textovém režimu jsou časově náročnější (převod z a do textové podoby), další nevýhoda je velikost souboru. V textovém režimu je soubor o poznání větší než v případě binárního režimu. Z tohoto důvodu se pro práci s velkými soubory používá binární režim. Pro práci v jednotlivých režimech slouží následující funkce (textový režim: `fprintf` a `fscanf`, binární režim: `fread` a `fwrite`), abychom mohli tyto funkce využívat, je nejprve nutné daný soubor otevřít.

11.3.1 Otevírání a zavírání souborů

Pro otevření příslušného souboru slouží funkce `fopen`. Její syntaxe vypadá následovně `fid=fopen(name,mode,arch)`, ve funkci je nutné definovat název souboru, režim přístupu a číselný formát souboru, který umožňuje sdílet soubory na počítačích různých architektur (udává, v jakém pořadí budou jednotlivé bajty ukládány/načítany do/z paměti). Funkce vrací identifikátor souboru fid přiřazený operačním systémem.

Parametr name udává v textovém řetězci název souboru s příslušným formátem (např. `'pokus.bin'`)

Parametr mode udává režim přístupu v textovém řetězci, může nabývat následujících hodnot:

‘r’ – otevře soubor pouze pro čtení

‘w’ – smaže obsah existujícího souboru, nebo vytvoří nový soubor a otevře ho pro zápis

‘a’ – vytvoří a otevře nový soubor pro zápis, nebo otevře existující soubor pro zápis na konec souboru

‘r+’ – otevře soubor pro čtení a zápis

‘w+’ – smaže obsah existujícího souboru, nebo vytvoří nový soubor a otevře ho pro čtení a zápis

‘a+’ – vytvoří a otevře nový soubor pro čtení a zápis, nebo otevře existující soubor pro čtení a zápis na konec souboru

Pokud parametr mode neuvedeme, tak se automaticky nastaví na hodnotu ‘r’, pro odlišení binárního režimu od textového se přidává k danému režimu přístupu ‘b’ nebo ‘t’. Například otevření binárního souboru pro čtení vypadá následovně ‘rb’.

Parametr arch je volitelný a představuje číselný formát souboru v textovém režimu, můžeme ho nastavit následovně:

‘native’ – číselný formát aktuálního počítače (nastaven jako výchozí)

‘ieee-be’ – IEEE Big Endien formát

‘ieee-le’ – IEEE Little Endien formát

‘vaxg’ – VAX G-float formát

‘vaxd’ – VAX D-float formát

‘cray’ – CRAY formát

Příklad otevření souboru pro čtení:

`fid = fopen('pokus.bin', 'w')` – funkce vrátí identifikátor souboru pokus.bin v režimu zápisu, který se uloží do proměnné fid

Po provedení příslušných operací (čtení/zápis) uzavřeme soubor pomocí funkce `fclose(fid)`. Parametr fid udává identifikátor aktuálně otevřeného souboru. Funkce `fclose(fid)` vrací hodnotu 0, pokud došlo k uzavření souboru.

Ukázka otevření a uzavření souboru je uvedena v příloze I na dvd.

11.3.2 Čtení binárních datových souborů s definovanou přesností

Pro čtení binárních datových souborů slouží v Octave funkce `fread`, syntaxe je následující `[A,count]=fread(fid,size,precision,skip,arch)`. Ve funkci je nutné definovat pouze identifikátor aktuálně otevřeného souboru. Ostatní parametry jsou

volitelné. Načtená data jsou uložena do matice A, pokud je použit výstupní parametr count, tak do něj funkce `fread` uloží počet úspěšně načtených prvků.

Parametr fid udává identifikátor aktuálně otevřeného souboru.

Parametr size udává, kolik prvků chceme načíst, je to volitelný parametr, pokud tento parametr nezadáme, automaticky dojde k načtení všech dat ze souboru. Jinak může mít následující tvar:

n – načte n prvků do sloupcového vektoru

inf – načte všechny prvky souboru, odpovídá výchozímu nastavení

[m,n] – načte tolik prvků, aby jimi (po sloupcích) vyplnil matici o rozměrech [m,n], pokud je prvků v souboru málo, tak dojde k doplnění matice nulami

Parametr precision udává požadovanou přesnost načítaných dat, jako výchozí je nastavena přesnost ‘uchar’. Po načtení jsou data automaticky uložena do matice A v dvojité přesnosti. Parametr precision můžeme také využít k načítání bloků prvků (např. ‘16*single’ načte blok 16-ti prvků s jednoduchou přesností). Této možnosti se využívá hlavně v kombinaci s parametrem skip. Další možností parametru precision je převod z načítané přesnosti do jiné (např. ‘double=>single’, první přesnost double udává přesnost načtených dat a druhá přesnost single udává přesnost uložených dat v matici A). Obě předchozí možnosti parametru precision lze kombinovat (např. ‘16*double=>single’, dojde k načtení bloku 16-ti prvků s dvojitou přesností a následně jsou prvky uloženy do matice A s jednoduchou přesností). Podporované přesnosti v Octave jsou:

‘uchar’ – neznaménkový znak (8 bitů), výchozí přesnost

‘schar’ – znaménkový znak (8 bitů)

‘char’ – znak (8 bitů)

‘int8’ – celé číslo (8 bitů)

‘int16’ – celé číslo (16 bitů)

‘int32’ – celé číslo (32 bitů)

‘int64’ – celé číslo (64 bitů)

‘uint8’ – neznaménkové celé číslo (8 bitů)

‘uint16’ – neznaménkové celé číslo (16 bitů)

‘uint32’ – neznaménkové celé číslo (32 bitů)

‘uint64’ – neznaménkové celé číslo (64 bitů)

‘single’ – reálné číslo (32 bitů)

‘double’ – reálné číslo (64 bitů)

‘short’ – celé číslo (velikost je závislá na hardwaru, zpravidla 16 bitů)

‘int’ – celé číslo (velikost je závislá na hardwaru, zpravidla 16 nebo 32 bitů)

‘long’ – celé číslo (velikost je závislá na hardwaru, zpravidla 32 bitů)

‘ushort’ – neznaménkové celé číslo (velikost je závislá na hardwaru, zpravidla 16 bitů)

‘uint’ – neznaménkové celé číslo (velikost je závislá na hardwaru, zpravidla 16 nebo 32 bitů)

‘ulong’ – neznaménkové celé číslo (velikost je závislá na hardwaru, zpravidla 32 bitů)

‘float’ – reálné číslo (velikost je závislá na hardwaru, zpravidla 32 bitů)

Parametr skip udává, kolik bajtů se má přeskočit po každém čtení. Pokud není tento parametr zadán, tak nedojde k žádnému přeskočení. Parametr skip použijeme, pokud potřebujeme načíst jen určitá data ze souboru. Abychom mohli bajty přeskakovat, je nutné v kódu použít funkci `fseek`, která nastaví identifikátor pozice v souboru.

Parametr arch má stejný význam jako u funkce `fopen`, může také nabývat stejných hodnot.

Ukázka načítání binárních datových souborů pro různé parametry funkce `fread` je uvedena v příloze I na dvd.

11.3.3 Zápis binárních datových souborů s definovanou přesností

Zápis binárních dat s definovanou přesností do souboru můžeme provést pomocí funkce `fwrite`. Syntaxe této funkce vypadá takto `count=fwrite(fid,A,precision,skip,arch)`, ve funkci je nutné definovat identifikátor aktuálně otevřeného souboru, data, která chceme do souboru zapsat a přesnost s jakou mají být data ukládána. Ostatní parametry jsou volitelné. Data jsou zapisována do souboru po sloupcích. Funkce vrací do proměnné count počet správně zapsaných prvků.

Parametr fid udává stejně jako u funkce `fread` identifikátor aktuálně otevřeného souboru.

Parametr A představuje binární data, která chceme do souboru uložit.

Parametr precision udává přesnost, s jakou mají být data do souboru ukládána. Podporované přesnosti jsou stejné jako u funkce `fread`.

Parametr skip udává počet bajtů, který se má přeskočit před každým zápisem. V tomto případě není nutné pro přeskočení části souboru použít funkci `fseek`, jako je tomu u čtení.

Parametr arch má opět stejný význak jako u funkce `fopen` a `fread`.

Ukázka zápisu binárních dat s definovanou přesností do souboru je uvedena v příloze I na dvd.

11.3.4 Nastavení a zjištění indikátoru pozice v souboru

Nastavení indikátoru pozice v souboru můžeme provést pomocí funkce `fseek`, naopak zjištění indikátoru pozice provedeme pomocí funkce `ftell`. Funkce `fseek` má následující syntaxi `status=fseek(fid,offset,origin)`, ve funkci je nutné nastavit identifikátor aktuálně otevřeného souboru, počet bajtů o který chceme indikátor pozice posunout a odkud chceme posun provést. Hodnota status se nastaví na 0, pokud proběhla operace úspěšně.

Parametr fid udává identifikátor aktuálně otevřeného souboru.

Parametrem offset nastavujeme počet bajtů, o který chceme indikátor pozice posunout. Pokud nastavíme offset vetší než 0, tak dojde k posunu směrem ke konci souboru. Nastavením hodnoty offset na hodnotu menší než 0 dojde k posunu k začátku souboru.

Parametr origin udává, odkud se má posun provádět. K dispozici máme tři hodnoty:

‘cof’ – posun se provede od aktuální pozice v souboru

‘bof’ – posun se provede od začátku souboru, tato hodnota je nastavena jako výchozí

‘eof’ – posun se provede od konce souboru

Pro zjištění pozice indikátoru pozice tedy použijeme funkci `ftell`, která má jednoduchou syntaxi `position=ftell(fid)`, ve funkci definujeme pouze identifikátor aktuálně otevřeného souboru. Funkce vrací pozici indikátoru pozice daného souboru. Do proměnné position se uloží počet bajtů od začátku souboru.

11.3.5 Zápis formátovaných dat do textových souborů

Pro zápis formátovaných dat do textových souborů můžeme využít funkce `fprintf`. Tato funkce má následující syntaxi `count=fprintf(fid, template, A, …, An)`, ve funkci je nutné definovat identifikátor souboru, do kterého chceme zápis provést, formát ukládaných dat a jaká data se mají do souboru ukládat. Po provedení funkce, se do proměnné count uloží počet úspěšně zapsaných bajtů.

Parametr fid udává identifikátor aktuálně otevřeného souboru, do kterého chceme zapisovat.

Parametr template je řetězec, podle kterého dochází k formátování dat. Tento parametr definuje výstupní formát dat. Formátovací řetězec se zpravidla skládá z konverzních specifikátorů a textu. Text může být složen z takzvaných standardních alfanumerických znaků (písmena, čísla) a z escape znaků.

Základní escape znaky vypadají následovně:

\n – nový řádek

\t – horizontální tabelátor

\v – vertikální tabelátor

\b – znak zpět

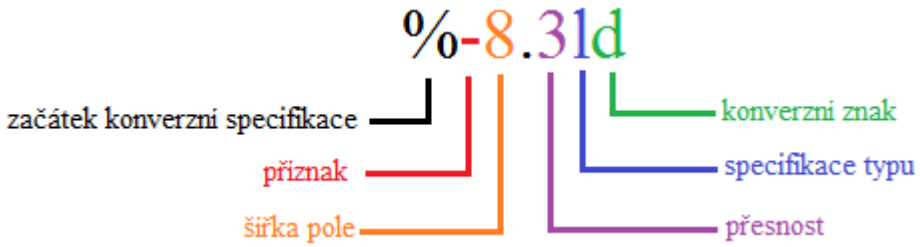
\\ – zpětné lomítko

\% – znak procenta

\f – odstránkování

\r – začátek řádky

Konverzní specifikace se skládá z následujících parametrů:



Obrázek 11.4: *Parametry konverzní specifikace pro funkci fprintf*

- Znak začátku konverzní specifikace – %

- Příznak (volitelný parametr) – umožňuje úpravu formátované hodnoty:
  - Znaménko mínus (-) způsobí zarovnání výsledku doleva
  - Znaménko plus (+) vloží před formátovanou hodnotu znaménko plus
  - Znak mezera ( ) doplní hodnotu mezerami na požadovanou šířku pole, mezery se vloží před hodnotu
  - Znak nula (0) doplní hodnotu nulami na požadovanou šířku pole

- Šířka pole (volitelný parametr) – označuje minimální počet znaků k tisku, pokud je námi zadaná šířka pole větší, než je počet znaků zapisované hodnoty, tak jsou před hodnotu vloženy mezery na doplnění šířky.

- Přesnost (volitelný parametr) – podle daného konverzního znaku udává buď počet platných číslic (pro převod hodnoty π platí `%4.3g` výsledek bude 3.14), nebo počet číslic za desetinnou tečkou (pro převod hodnoty π platí `%4.3f` výsledek bude 3.142).

- Specifikace typu (volitelný parametr) – tento parametr, který se v Matlabu používá k vyjádření čísla v dvojité nebo jednoduché přesnosti Octave ignoruje, proto je lepší tento parametr vynechat.

- Konverzní znak – udává, do jaké podoby chceme data převést:
  - d – desítkové celé číslo znaménkové
  - o – osmičkové celé číslo neznaménkové
  - u – desítkové celé číslo neznaménkové

- x – šestnáctkové celé číslo neznaménkové
- e – reálné číslo s pohyblivou řádovou čárkou v exponenciální notaci
- f – reálné číslo s pohyblivou řádovou čárkou v notaci s pevnou řádovou čárkou
- g – reálné číslo s pohyblivou řádovou čárkou ve výhodnější notaci (f nebo e)
- s – řetězec znaků
- c – jediný znak

Parametr A přestavuje matici dat, která chceme zapsat do příslušného textového souboru. Matice je procházena po sloupcích.

Jako další parametry můžeme zadat další matice, které chceme do souboru uložit.

Příklad zápisu formátovaných dat do souboru je uveden v příloze I na dvd.

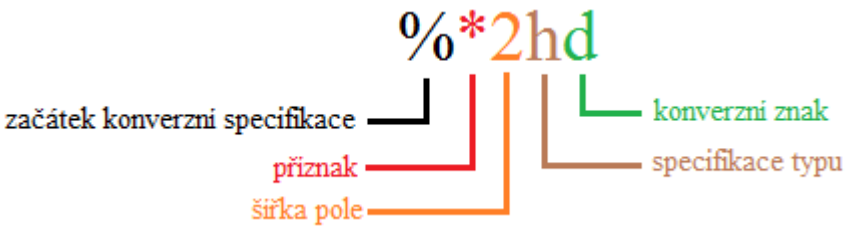
11.3.6 Čtení formátovaných dat z textových souborů

Pro načtení formátovaných dat z textového souboru do matice slouží funkce `fscanf`. Syntaxe této funkce je následující `[A, count] = fscanf(fid, template, size)`, ve funkci je nutné zadat identifikátor souboru a formát načítaných dat. Funkce načítá data do matice A a do proměnné count vrací počet úspěšně načtených prvků.

Parametr fid udává identifikátor aktuálně otevřeného souboru.

Parametr template je řetězec, který určuje formát načítaných dat. Tento řetězec obsahuje stejně jako u funkce `fprintf` jak text, tak konverzní specifikátory. Při čtení ze souboru dochází ke snaze přizpůsobit data formátovacímu řetězci, pokud dojde k přizpůsobení, tak jsou data zapsána do matice A ve sloupcovém pořadí. Pokud k přizpůsobení nedojde, tak matice zůstane prázdná a proměnná count bude nastavena na nulu.

Konverzní specifikace se skládá z následujících parametrů:



Obrázek 11.5: *Parametry konverzní specifikace pro funkci fscanf*

- Znak začátku konverzní specifikace – %

- Příznak (volitelný parametr) – umožňuje přeskočit určité formáty:
  - Znak hvězdička (*) způsobí přeskočení daného formátu, například konverzní specifikace *%*c* způsobí, že se do matice nebudou ukládat znaky (respektive jejich dekadické hodnoty z ASCII tabulky)

- Šířka pole (volitelný parametr) – udává maximální počet znaků, který chceme do matice načíst.

- Specifikace typu (volitelný parametr) – tento parametr stejně jako u funkce `fprintf` Octave ignoruje.
- Konverzní znak – udává, v jaké podobě chceme data načíst:
  - d – desítkové celé číslo znaménkové
  - o – osmičkové celé číslo neznaménkové
  - u – desítkové celé číslo neznaménkové
  - x – šestnáctkové celé číslo neznaménkové
  - e – reálné číslo s pohyblivou řádovou čárkou v exponenciální notaci
  - f – reálné číslo s pohyblivou řádovou čárkou v notaci s pevnou řádovou čárkou
  - g – reálné číslo s pohyblivou řádovou čárkou ve výhodnější notaci (f nebo e)
  - s – řetězec znaků
  - c – jediný znak

Parametr size udává, kolik prvků chceme načíst, je to volitelný parametr. Pokud tento parametr nezadáme, automaticky dojde k načtení všech dat ze souboru. Jinak může mít následující tvar:

n – načte n prvků do sloupcového vektoru

inf – načte všechny prvky souboru, odpovídá výchozímu nastavení

[m,n] – načte tolik prvků, aby jimi (po sloupcích) vyplnil matici o rozměrech [m,n], pokud je prvků v souboru málo, tak dojde k doplnění matice nulami. Počet sloupců n může mít hodnotu inf, ale počet řádků m ne.

Příklad čtení formátovaných dat ze souboru je uveden v příloze I na dvd.

Při zpracování této kapitoly jsem čerpal z literatury [1], [11], [15], [16], [17], [18], [19].

12 Závěr

Cílem této diplomové práce bylo poskytnout studentům předmětů zabývajících se zpracováním signálů alternativu k výpočetnímu systému Matlab. K tomuto účelu jsem využil výpočetní software Octave, který se ukázal jako alespoň částečně konkurence schopný licencovanému Matlabu, který se využívá v mnoha předmětech zabývajících se zpracováním signálů a obrazu. Přes veškerá úskalí, která nese používání programu Octave se mi podařilo zpracovat jednotlivé kapitoly práce tak, že mohou být využity při výuce předmětu Zpracování číslicových signálů. Ke každé kapitole týkající se zpracování signálů jsem vytvořil vzorový skript, ve kterém je prakticky ukázáno řešení dané problematiky přímo v Octave. Každý skript je opatřen podrobným komentářem. Pro zájemce z řad studentů je navíc ve vzdělávacím prostředí Moodle vytvořen předmět Octave, ve kterém jsou uloženy jednotlivé kapitoly mé práce.

Při práci v programu Octave je nutné dávat pozor na používání jednotlivých funkcí, protože ne zdaleka u všech je syntaxe stejná jako v programu Matlab. Některé funkce chybí, některé jsou nahrazeny jinými.

Co se týče jednotlivých kapitol, tak největší problém se vyskytnul u kapitoly Práce se symbolickými proměnnými, kde balíček symbolic ani zdaleka nedosahuje kvalit balíčku Symbolic Math Toolbox programu Matlab, největší slabina je asi v absenci možnosti symbolického integrování. Balíček symbolic by však měl být v létě kompletně předělán, takže se dá očekávat výrazný nárůst jeho funkčnosti a poté může být tato kapitola doplněna o nové funkce. Další omezení programu Octave se projevilo při tvorbě animací. Pomalé překreslování aktuálního grafu prakticky znemožňuje tvorbu animace přímo v Octave. Abychom mohli animaci vytvořit, je nutné mít k dispozici program pro tvorbu animací, který následně spojí vytvořené obrázky v Octave v animaci. Problém se také vyskytnul při návrhu IIR filtrů pomocí frekvenční transformace s digitální oblasti do analogové a zpět, kde bylo nutné upravit funkci sftrans balíčku signal. Na poslední větší problém jsem narazil při pokusu o nastavení definované přesnosti pro výpočet. Octave bohužel uživateli neumožňuje nastavení vlastní přesnosti jako je tomu v programu Matlab. Funkce k tomu určené jsou obsaženy v balíčku symbolic, ale nepracují správně, proto se dá opět očekávat, že při úpravě balíčku symbolic se i tato funkce bude dát využít.

Množství problematiky, které je ještě nutno zpracovat, aby se dal Octave plnohodnotně využít je obrovské, proto pokud se osvědčí program Octave alespoň v nějakém předmětu, tak budu tuto práci dále rozšiřovat a doplňovat o nové možnosti zpracování signálů v Octave.

Použitá literatura

[1] *GNU Octave* [online]. 1998 [cit. 2012-03-12]. Dostupné z: http://www.gnu.org/software/octave/

[2] *Octave-Forge* [online]. 2009 [cit. 2012-03-12]. Dostupné z: http://octave.sourceforge.net/index.html

[3] *Octave - český průvodce programem* [online]. 2006 [cit. 2012-03-12]. Dostupné z: http://www.octave.cz/index.html

[4] *QtOctave* [online]. 2011 [cit. 2012-03-12]. Dostupné z: http://qtoctave.wordpress.com/

[5] *GUI Octave: Home* [online]. 2012 [cit. 2012-03-12]. Dostupné z: http://guioctave.com/

[6] ZAPLATÍLEK, Karel a Bohuslav DOŇAR. *MATLAB pro začátečníky*. 2. vyd. Praha: BEN - technická literatura, 2005, 151 s. ISBN 80-730-0175-6.

[7] ZAPLATÍLEK, Karel a Bohuslav DOŇAR. *MATLAB tvorba uživatelských aplikací*. 1. vyd. Praha: BEN – technická literatura, 2004, 215 s. ISBN 80-730-0133-0.

[8] *Creating Animations With Octave | Karol Krizka* [online]. 2009 [cit. 2013-01-31]. Dostupné z: http://www.krizka.net/2009/11/06/creating-animations-with-octave/

[9] Creating a point moving along a graph in MATLAB. In: *Stack Overflow* [online]. 2011 [cit. 2013-04-16]. Dostupné z: http://stackoverflow.com/questions/4959528/creating-a-point-moving-along-a-graph-in-matlab

[10] SKAPA, Jan. Zpracování číslicových signálů: cvičení. Ostrava, 2011.

[11] SKALICKÝ, Petr. *Digitální filtrace a signálové procesory: Návrh číslicových filtrů s nekonečnou impulzní odezvou*. Praha, 1995. Dostupné z: http://radio.feld.cvut.cz/courses/CS/web/CS/Literatura/Navrh_CF.pdf

[12] NOVOTNÝ, Jan. *Úvod do číslicového zpracování signálů - UCZ: cv4*. Praha, 2001. Dostupné z: http://noel.feld.cvut.cz/vyu/ucz/cv4/index.htm

[13] Index of /POLITECHNIKA/Dydaktyka/AUTOMATYKA/AutoLab/Matlab/TOOLBOX/SIGNAL/. *Http://chmielowski.eu/* [online]. 2013 [cit. 2013-02-28]. Dostupné z: http://chmielowski.eu/POLITECHNIKA/Dydaktyka/AUTOMATYKA/AutoLab/Matlab/TOOLBOX/SIGNAL/

[14] SCHIMMEL, Jiří a Petr SYSEL. *MCSI -- Číslicové zpracování signálů*. 2010.

[15] *Aritmetika počítače*. 2006. Dostupné z: http://atrey.karlin.mff.cuni.cz/~morf/vyuka/sem/materialy/skripta_aritmetika.pdf

[16] PERŮTKA, Karel. *MATLAB - Základy pro studenty informačních technologií a automatizace*. 1. doplněné vydání. Zlín, 2004. Dostupné z: www.grman.tym.sk/php/download.php?file=matlab_zaklady

[17] *IEEE 754 - reprezentace čísel s pohyblivou řádovou čárkou*. 1999. Dostupné z: http://amber.feld.cvut.cz/psp/IEEE754.htm

[18] HERINGOVÁ a Petr HORA. *MATLAB: Díl I. - Práce s programem*. 1995. Dostupné z: http://www.cdm.cas.cz/czech/hora/vyuka/mvs/tutorial.pdf

[19] HERINGOVÁ, Blanka a Petr HORA. *MATLAB: Díl II. - Popis funkcí*. 1995. Dostupné z: http://www.cdm.cas.cz/czech/hora/vyuka/mvs/reference.pdf

Seznam příloh

Součástí DP je DVD.

Adresářová struktura přiloženého DVD:

Příloha.A: Vzorce pro návrh IIR filtrů

a) Normování mezních frekvencí pomocí Nyquistovy frekvence f_N do intervalu <0,π>

- Dolní propust

$$\varpi_p = \pi \times \frac{f_p}{f_N}\left[-;-, Hz, Hz\right]$$

$$\varpi_s = \pi \times \frac{f_s}{f_N}\left[-;-, Hz, Hz\right]$$

- Horní propust

$$\varpi_p = \pi \times \frac{f_p}{f_N}\left[-;-, Hz, Hz\right]$$

$$\varpi_s = \pi \times \frac{f_s}{f_N}\left[-;-, Hz, Hz\right]$$

- Pásmová propust

$$\varpi_{pd} = \pi \times \frac{f_{pd}}{f_N}\left[-;-, Hz, Hz\right]$$

$$\varpi_{sd} = \pi \times \frac{f_{sd}}{f_N}\left[-;-, Hz, Hz\right]$$

$$\varpi_{ph} = \pi \times \frac{f_{ph}}{f_N}\left[-;-, Hz, Hz\right]$$

$$\varpi_{sh} = \pi \times \frac{f_{sh}}{f_N}\left[-;-, Hz, Hz\right]$$

- Pásmová zádrž

$$\varpi_{pd} = \pi \times \frac{f_{pd}}{f_N}\left[-;-, Hz, Hz\right]$$

$$\varpi_{sd} = \pi \times \frac{f_{sd}}{f_N}\left[-;-, Hz, Hz\right]$$

$$\varpi_{ph} = \pi \times \frac{f_{ph}}{f_N}\left[-;-, Hz, Hz\right]$$

$$\varpi_{sh} = \pi \times \frac{f_{sh}}{f_N}\left[-;-, Hz, Hz\right]$$

b) Dopředná frekvenční transformace

- Výpočet mezních úhlových frekvencí analogové předlohy filtru (předzkreslení pro bilineární transformaci)
  - Dolní propust

$$\omega_p = \frac{2}{T_{vz}} \times \tan\frac{\varpi_p}{2}\left[s^{-1}; s, -\right]$$

$$\omega_s = \frac{2}{T_{vz}} \times \tan\frac{\varpi_s}{2}\left[s^{-1}; s, -\right]$$

  - Horní propust

$$\omega_p = \frac{2}{T_{vz}} \times \tan\frac{\varpi_p}{2}\left[s^{-1}; s, -\right]$$

$$\omega_s = \frac{2}{T_{vz}} \times \tan\frac{\varpi_s}{2}\left[s^{-1}; s, -\right]$$

  - Pásmová propust

$$\omega_{pd} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{pd}}{2}\left[s^{-1}; s, -\right]$$

$$\omega_{sd} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{sd}}{2}\left[s^{-1}; s, -\right]$$

$$\omega_{ph} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{ph}}{2}\left[s^{-1}; s, -\right]$$

$$\omega_{sh} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{sh}}{2}\left[s^{-1}; s, -\right]$$

  - Pásmová zádrž

$$\omega_{pd} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{pd}}{2}\left[s^{-1}; s, -\right]$$

$$\omega_{sd} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{sd}}{2}\left[s^{-1}; s, -\right]$$

$$\omega_{ph} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{ph}}{2}\left[s^{-1}; s, -\right]$$

$$\omega_{sh} = \frac{2}{T_{vz}} \times \tan\frac{\varpi_{sh}}{2}\left[s^{-1}; s, -\right]$$

- Určení mezních úhlových frekvencí normované dolní propusti
  - Dolní propust → NDP

$$\Omega_p = \frac{\omega_p}{\omega_p} = 1\left[-; s^{-1}, s^{-1}\right]$$

$$\Omega_s = \frac{\omega_s}{\omega_p}\left[-; s^{-1}, s^{-1}\right]$$

- Horní propust → NDP

$$\Omega_p = \frac{\omega_p}{\omega_p} = 1\left[-; s^{-1}, s^{-1}\right]$$

$$\Omega_s = \frac{\omega_p}{\omega_s}\left[-; s^{-1}, s^{-1}\right]$$

- Pásmová propust → NDP

$$\Omega_{pd} = \frac{\omega_{pd}^2 - \omega_{pd} \times \omega_{ph}}{\omega_{pd} \times \left(\omega_{ph} - \omega_{pd}\right)} = -1\left[-; s^{-1}, s^{-1}\right]$$

$$\Omega_{ph} = \frac{\omega_{ph}^2 - \omega_{pd} \times \omega_{ph}}{\omega_{ph} \times \left(\omega_{ph} - \omega_{pd}\right)} = 1\left[-; s^{-1}, s^{-1}\right]$$

$$\Omega_{sd} = \frac{\omega_{sd}^2 - \omega_{pd} \times \omega_{ph}}{\omega_{sd} \times \left(\omega_{ph} - \omega_{pd}\right)}\left[-; s^{-1}, s^{-1}, s^{-1}\right]$$

$$\Omega_{sh} = \frac{\omega_{sh}^2 - \omega_{pd} \times \omega_{ph}}{\omega_{sh} \times \left(\omega_{ph} - \omega_{pd}\right)}\left[-; s^{-1}, s^{-1}, s^{-1}\right]$$

V případě normované dolní propusti je vždy $\Omega_p = 1$.

$$\Omega_s = \min\left(\left|\Omega_{sd}\right|, \left|\Omega_{sh}\right|\right)$$

- Pásmová zádrž → NDP

$$\Omega_{pd} = \frac{\omega_{pd} \times \left(\omega_{ph} - \omega_{pd}\right)}{\omega_{pd}^2 - \omega_{pd} \times \omega_{ph}} = -1\left[-; s^{-1}, s^{-1}\right]$$

$$\Omega_{ph} = \frac{\omega_{ph} \times \left(\omega_{ph} - \omega_{pd}\right)}{\omega_{ph}^2 - \omega_{pd} \times \omega_{ph}} = 1\left[-; s^{-1}, s^{-1}\right]$$

$$\Omega_{sd} = \frac{\omega_{sd} \times \left(\omega_{ph} - \omega_{pd}\right)}{\omega_{sd}^2 - \omega_{pd} \times \omega_{ph}}\left[-; s^{-1}, s^{-1}, s^{-1}\right]$$

$$\Omega_{sh} = \frac{\omega_{sh} \times \left(\omega_{ph} - \omega_{pd}\right)}{\omega_{sh}^2 - \omega_{pd} \times \omega_{ph}}\left[-; s^{-1}, s^{-1}, s^{-1}\right]$$

V případě normované dolní propusti je vždy $\Omega_p = 1$.

$$\Omega_s = \min\left(\left|\Omega_{sd}\right|, \left|\Omega_{sh}\right|\right)$$

- Výpočet řádu normované dolní propusti N_{NDP} a výpočet charakteristické frekvence Ω_n pro danou aproximaci

- Butterworthova aproximace

$$N_{NDP} \geq \frac{\log\sqrt{\frac{\frac{1}{A_s^2-1}}{\frac{1}{A_p^2-1}}}}{\log\left(\frac{\Omega_s}{1}\right)}\left[-;-,-,-\right]$$

$$\Omega_{np} = \frac{\Omega_p}{\left(\frac{1}{A_p^2}-1\right)^{\frac{1}{2\times N_{NDP}}}}\left[-;-,-,-\right]$$

$$\Omega_{ns} = \frac{\Omega_s}{\left(\frac{1}{A_s^2}-1\right)^{\frac{1}{2\times N_{NDP}}}}\left[-;-,-,-\right]$$

Jako Ω_n vybereme jednu z vypočtených hodnot podle toho, na které frekvenci chceme přesně splnit specifikaci. $\Omega_n = \Omega_{ns}$ nebo $\Omega_n = \Omega_{np}$.

- Čebyševova aproximace 1. druhu

$$N_{NDP} \geq \frac{\cosh^{-1}\sqrt{\frac{\frac{1}{A_s^2-1}}{\frac{1}{A_p^2-1}}}}{\cosh^{-1}\left(\frac{\Omega_s}{\Omega_p}\right)}\left[-;-,-,-\right]$$

$$\Omega_n = \Omega_p$$

- Čebyševova aproximace 2. druhu

$$N_{NDP} \geq \frac{\cosh^{-1}\sqrt{\frac{\frac{1}{A_s^2-1}}{\frac{1}{A_p^2-1}}}}{\cosh^{-1}\left(\frac{\Omega_s}{\Omega_p}\right)}\left[-;-,-,-\right]$$

$$\Omega_n = \Omega_s$$

- Určení nul p_0, pólů p_∞ a násobného koeficientu k_p přenosové funkce *H(p)* pro normovanou dolní propust

  - Butterworthova aproximace

    ```
    [p_0,p_infty,k_p]=buttap_octave(N_NDP)
    ```

  - Čebyševova aproximace 1. druhu

    ```
    [p_0,p_infty,k_p]=cheb1ap_octave(N_NDP,a_p)
    ```

  - Čebyševova aproximace 2. druhu

    ```
    [p_0,p_infty,k_p]=cheb2ap_octave(N_NDP,a_s)
    ```

c) Zpětná frekvenční transformace *H(p) → H(s)*

- Určení charakteristických frekvencí analogové předlohy filtru

  - NDP → dolní propust

    $$\omega_n = \Omega_n \times \omega_p \left[s^{-1}; -, s^{-1} \right]$$

  - NDP → horní propust

    $$\omega_n = \frac{\omega_p}{\Omega_n} \left[s^{-1}; s^{-1}, - \right]$$

  - NDP → pásmová propust

    $$\omega_{nd} = \frac{-\Omega_n \times \left(\omega_{ph} - \omega_{pd}\right) + \sqrt{\Omega_n^2 \times \left(\omega_{ph} - \omega_{pd}\right)^2 + 4 \times \omega_{ph} \times \omega_{pd}}}{2} \left[s^{-1}; -, s^{-1}, s^{-1} \right]$$

    $$\omega_{nh} = \frac{\Omega_n \times \left(\omega_{ph} - \omega_{pd}\right) + \sqrt{\Omega_n^2 \times \left(\omega_{ph} - \omega_{pd}\right)^2 + 4 \times \omega_{ph} \times \omega_{pd}}}{2} \left[s^{-1}; -, s^{-1}, s^{-1} \right]$$

  - NDP → pásmová zádrž

    $$\omega_{nd} = \frac{-\frac{\left(\omega_{ph} - \omega_{pd}\right)}{\Omega_n} + \sqrt{\frac{\left(\omega_{ph} - \omega_{pd}\right)^2}{\Omega_n^2} + 4 \times \omega_{pd} \times \omega_{ph}}}{2} \left[s^{-1}; s^{-1}, s^{-1}, - \right]$$

    $$\omega_{nh} = \frac{\frac{\left(\omega_{ph} - \omega_{pd}\right)}{\Omega_n} + \sqrt{\frac{\left(\omega_{ph} - \omega_{pd}\right)^2}{\Omega_n^2} + 4 \times \omega_{pd} \times \omega_{ph}}}{2} \left[s^{-1}; s^{-1}, s^{-1}, - \right]$$

- Určení nul s_0, pólů s_∞ a násobného koeficientu k_s přenosové funkce *H(s)* pomocí funkce `sftransmoje`

  - Dolní propust

  ```
  [s_0,s_infty,k_s]=sftransmoje(p_0,p_infty,k_p,omega_n,1)
  ```

  - Horní propust

  ```
  [s_0,s_infty,k_s]=sftransmoje(p_0,p_infty,k_p,omega_n,0)
  ```

  - Pásmová propust

  ```
  [s_0,s_infty,k_s]=sftransmoje(p_0,p_infty,k_p,[omega_nd … omega_nh],1)
  ```

  - Pásmová zádrž

  ```
  [s_0,s_infty,k_s]=sftransmoje(p_0,p_infty,k_p,[omega_nd … omega_nh],0)
  ```

- Získání koeficientů b_s a a_s přenosové funkce *H(s)*, platí pro všechny typy filtrů

  ```
  b_s = poly(s_0)*k_s
  ```

  ```
  a_s = poly(s_infty)
  ```

d) Bilineární transformace *H(s)* → *H[z]* (platí pro všechny typy filtrů)

- Určení nul z_0, pólů z_∞ a násobného koeficientu k_z přenosové funkce *H[z]* pomocí funkce `bilinear`

  ```
  [z_0,z_infty,k_z] = bilinear(s_0,s_infty,k_s,T_vz)
  ```

- Získání koeficientů b_z a a_z přenosové funkce *H[z]*

  ```
  b_z = k_z*poly(z_0)
  ```

  ```
  a_z = poly(z_infty)
  ```