Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

OCTAVE a OpenFOAM

a jejich využití ve fyzice

Bakalářská práce

Marian Viskup

školitel: RNDr. Jelínek Petr, Ph.D.

České Budějovice 2016

VISKUP, M., 2016: OCTAVE a OpenFOAM a jejich využití ve fyzice, bakalářská práce. (OCTAVE and OpenFOAM and their use in physics. Bc. Thesis, in Czech) – 58 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace:

Cílem této bakalářské práce je vytvoření několika vzorových příkladů v programech GNU Octave a OpenFOAM. Seznámit se s jejich užíváním a sepsat stručné návody na instalaci a práci s nimi pro předmět Software pro vědecko-technické výpočty, realizovaný na Přírodovědecké fakultě, Jihočeské univerzity.

Annotation:

The main goal of this bachelor thesis is to create a several examples of GNU Octave and OpenFOAM. To get acquainted myself with their usage and write short manuals for instalation and work with them for school subject Software for Scientific Computing, which is realized at the Faculty of Science, University of South Bohemia.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátu.

V Českých Budějovicích 15. listopadu 2016 ----------

Marian Viskup

Poděkování

Tímto bych rád poděkoval za spolupráci a odborný dohled vedoucímu mé bakalářské práce RNDr. Petru Jelínkovi, Ph.D. za jeho cenné rady, které mi pomohly úspěšně zhotovit tuto práci.

Obsah

1. Úvod a cíl práce

1.1 Úvod

Využití výpočetní techniky ve fyzice má širokou škálu možností. Přesněji řečeno, využívá se široká paleta softwarů pro nejrůznější vědecko-technické a numerické výpočty. Samy tyto programy by mnoho neposloužily bez adekvátních matematických a fyzikálních postupů. Pro jejich složitost jsou tyto programy velkými pomocníky z důvodu urychlení a řešení. Výběr těchto programů v dnešní době je velmi rozsáhlý a lze tedy vybírat ze široké škály, které poslouží k různý výpočtům.

Jak si ale vybrat ten správný software? Záleží na dané problematice a na účelu použití daného software. S některými programy lze lépe pracovat ve finanční sféře a s jinými třeba ve strojírenském průmyslu.

Dále je tu otázka finančních nákladů na takovýto software. Je známo, že licence špičkových výpočetních programů jsou velmi drahé a ne každý jedinec je schopen takovou investici učinit. V takovém případě tu existuje sorta tzv. free softwarů, neboli svobodný software, ke kterému je k dispozici zdrojový kód. Tento kód lze stáhnout na internetu, s možností užívat ho či modifikovat bez jakýchkoliv omezení. Tyto základní svobody pro svobodný software vyplývají z GNU projektu. Projekt definuje základní svobody: spouštět program, studovat a měnit zdrojový kód, šířit nezměněné kopie, šířit své upravené verze. Mezi tyto software spadá GNU Octave a OpenFOAM o kterých pojednává tato práce [1][5][6].

1.2 Cíl práce

První část této práce bude pojednávat o programu GNU Octave. Podrobně popíšeme jeho instalaci na různé operační systémy. Uvedeme si princip práce s programem, jeho rozhraní, psaní zdrojového kódu, chybové hlášky a grafické výstupy. Na konci první části práce bude nastíněno několik demonstračních úloh, které by měly sloužit jako ukázka práce s tímto programem. Úlohy budou směřovány na praktické využití ve fyzice.

Druhá část práce se bude věnovat software OpenFOAM, jehož oblíbenost v poslední době velice stoupá a bude zaměřena na instalaci, funkci a strukturu tohoto modelovacího programu. Konec této práce se bude zabývat využitím software v problematice proudění, zdrojovému kódu a grafickému výstupu.

2 GNU Octave

2.1 Co je GNU Octave

GNU Octave je svobodný software s volně šiřitelným zdrojovým kódem, který je do značné míry kompatibilní s výpočetním software MATLAB [2]. Díky této skutečnosti ho mnoho uživatelů používá jako jeho plnohodnotnou náhradu. Nebylo tomu tak vždy, a to díky špičkovým možnostem grafických výstupů v MATLAB a větší škály matematických funkcí, které v minulosti Octave velice předbíhaly. V nejnovější verzi 4.2.0 byly rozšířeny matematické knihovny o mnoho funkcí. K dispozici jsou funkce pro jednotlivá odvětví např. ekonomika, elektrotechnika, fyzika, dynamika atd. Grafické výstupy jsou zobrazovány pomocí grafického doplňku Gnuplot [3]. Gnuplot je program pro zobrazování dat s neomezenou škálou modifikací. Octave byl vyvinut Johnem W. Eatonem v roce 1992, nicméně projekt vznikl již v roce 1988. Nachází uplatnění v řešení numerických výpočtů, v lineární algebře, nelineárních a diferenciálních rovnicích a integrování funkcí. První verze 1.0 byla vydána v roce 1994.

V současnosti je dostupná již zmíněná stabilní verze 4.2.0, která je velice výhodná. Díky projektu GNU mohou uživatelé doplnit program o vlastní funkce, šířit je volně dalším uživatelům a lze jej volně stáhnout [1][2].

2.2 Instalace

Na internetu je dostupný ke stažení v mnoha různých verzích. Záleží pouze na uživateli, jakou verzi si vybere a na jaký operační systém ji chce nainstalovat. Práce se bude zabývat verzí 4.0.2, která byla vydána 21. 4. 2016. Nejvíce rozšířenými operačními systémy jsou OS Windows a Linux. Proto bude uvedena instalace pro obě varianty [1].

2.2.1 Instalace pro Windows

V ČR je nejrozšířenější varianta instalace na OS Windows. Tento fakt vyplývá z velké oblíbenosti tohoto OS v ČR. Některé instalace nesou jednu nevýhodu. Grafické nástavby, které umožňují zobrazit výsledky výpočtů např. výstupní graf, křivku atd. nejsou

v instalačním balíčku k dispozici. Tento nedostatek se projevoval zejména u starších verzí tím, že při zadání příkazu pro grafický výstup, program přestal odpovídat a následně se ukončil. Díky této skutečnosti je nutné nainstalovat i grafický doplněk např. Gnuplot. Doporučuje se tedy pečlivě vybrat instalační balíček, který již obsahuje vnořené funkce pro grafické výstupy, nebo externí program pro jejich vykreslení. Instalace, která byla použita v této práci je dostupná na následujícím odkaze [5]. Níže uvedený obrázek nám zobrazuje úvodní obrazovku instalace:

Obrázek 1.: GNU Octave instalace

Jelikož čeština pro verzi 4.0.2 není k dispozici, bude instalace probíhat v angličtině. Po stažení a otevření instalačního balíku se zobrazí uvítací okno a doporučení o ukončení všech běžících úloh na PC. Instalace se provede jako u většiny podobných programů. Proběhne seznámení s licenčními podmínkami a užíváním programu. Posledním krokem bude umístnění programu, dokončení instalace a doplňkových funkcí [1][2][3].

2.2.2 Instalace pro Linux

Instalace pro OS Ubuntu, probíhá za pomoci příkazové řádky, do které se vypíše několik příkazů. Základem je mít nainstalovanou verzi OS, která bude kompatibilní s verzí

zvoleného programu. Problémem, na který je možné narazit, bude při hledání stručného návodu jak program naistalovat a jaké verze jsou kompatibilní. Existuje mnoho postupů a návodů, ale většina nezaručí úplný chod programu. To může být způsobeno neoblíbeností OS Linux a nezkušeností s tímto OS. Pro instalaci stejné verze, jako v této práci lze použít uvedený odkaz [3].

Zde jsou popsány kroky pro instalaci. Zadáním následujících příkazů do příkazové řádky, bude spuštěna instalace.

```
$ sudo apt-get build-dep octave
$ wget ftp.gnu.org/gnu/octave/octave-4.0.0.tar.gz
$ tar xzvf octave-4.0.0.tar.gz
$ cd octave-4.0.0
$ ./configure
$ make
$ sudo make install
```

Tímto je instalace GNU Octave dokončena. Nainstalovaná verze je kompatibilní s Ubuntu verze 14.04 “Trusty Tahr”. Velkou výhodou této verze je fakt, že distributor ji doporučuje jako jednu z nejlepších pro začátečníky. Tato verze zaručí stabilní prostředí. Octave se spustí v příkazové řádce pomocí následujícího příkazu [1][2][3]:

```
$ octave
```

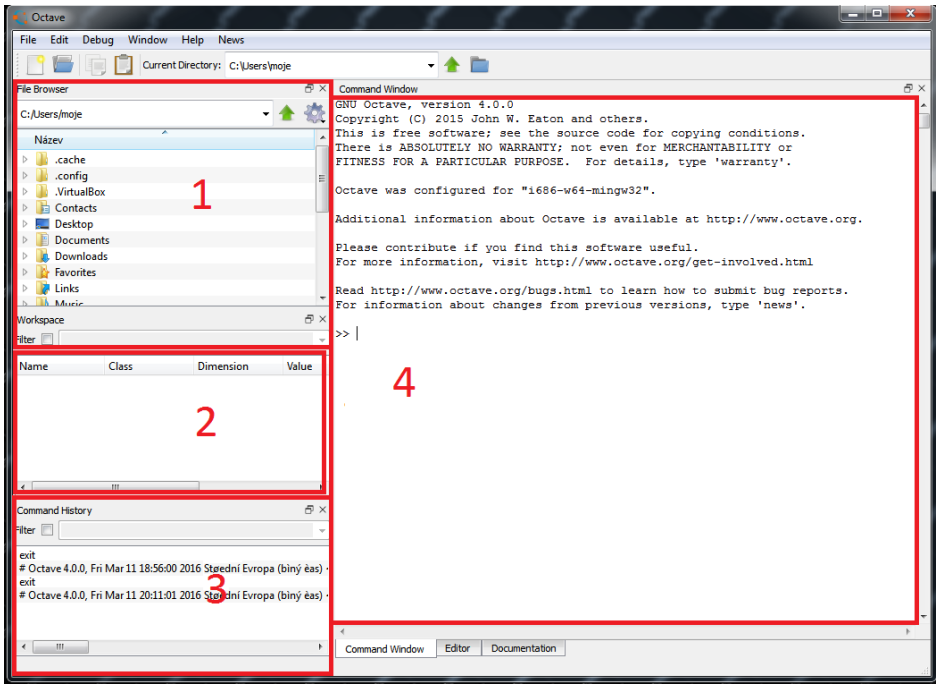
2.3 Pracovní prostředí

Práce s Octave probíhá ve dvou možných režimech, v grafickém prostředí GUI (Graphical User Interface), který je uveden na obrázku 1, nebo v příkazové řádce na obrázku 2. Práce v nich se nijak neliší. Záleží na vkusu uživatele, který ze způsobů zvolí a který pro něj bude vhodnější. V této práci budou uvedeny výstupy z GUI.

Pro základní orientaci v grafickém prostředí postačí znalost následujících čtyř vyznačených oblastní na obrázku 1 [1][2][4].

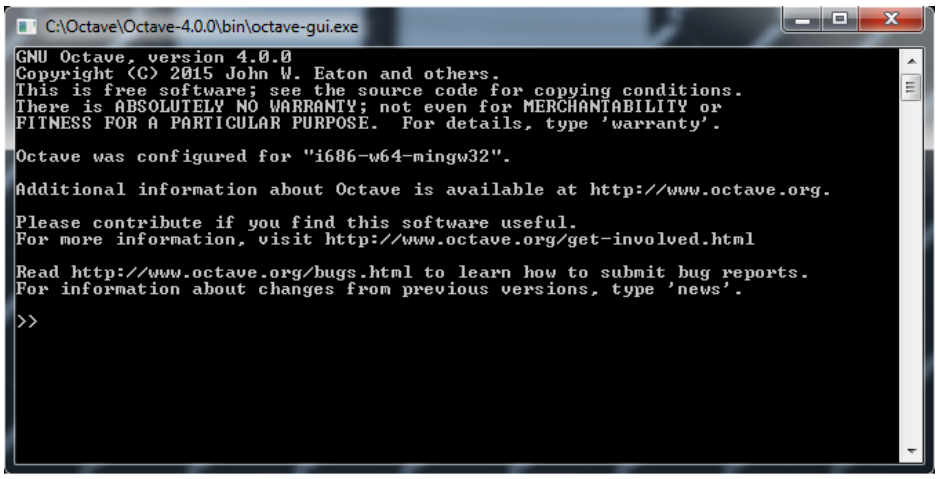
1) Kořenová složka a obsah disku.
2) Seznam použitých proměnných a jejich hodnot.

3) Seznam použitých příkazů (zjednodušuje orientaci), zůstává v paměti i po opětovném zapnutí programu, usnadňuje práci i díky tomu že pomocí kurzorové šipky nahoru je možné z paměti vyvolat provedené příkazy, které se byly zadány v příkazové řádce.
4) Příkazové okno pro implementaci kódu.



Obrázek 2.: Octave – grafické prostředí

Varianta příkazové řádky nenabízí žádné grafické úpravy, jen místo pro zdrojový kód.



Obrázek 3.: Octave – příkazový řádek

2.4 Práce s Octave

Základní pravidla pro práci s programem jsou následující. Spuštění programu pomocí příkazu `octave` v příkazovém řádku, nebo pomocí ikony. Po spuštění je program připraven přijímat první příkazy. Program je schopný pracovat s více výrazy na jednom řádku, což může velice ušetřit čas. Jako nápomocné se jeví komentáře, které usnadní orientaci a zpřehlední kód. Všechny komentáře začínají znakem `%`, nebo `#`. Při problémech, nebo zamrznutí GUI se probíhající operace ukončí pomocí stisku `ctrl - c`. Při potížích se používá, nápověda `help,` která se zapisuje následovně [2][4]:

```
octave:> help příkaz
```

Pro zobrazení kompletní nápovědy:

```
octave:> help help
```

Po zadání tohoto příkazu se zobrazí URL adresa na podrobný online návod dostupný na uvedeném zdroji [1]. Pokud není znám celý název příkazu, ale jsou známy počáteční písmena, lze tyto znaky napsat do řádky a pomocí dvojího poklepání na tabulátor zobrazit nabídku příkazů se zadanými znaky [2][4].

```
octave:> he
help     helpdlg  hess     hex2dec  hex2num
```

V případě zadání špatného názvu příkazu, nebo špatné syntaxe, program zobrazí chybovou hlášku [2][4].

```
octave:> sqrt 25
error: sqrt: expecting numeric argument
```

Je zřejmé, že došlo ke špatné syntaxi a program na tuto chybu upozorní. Zápis ve správném tvaru je následující [2][4].

```
octave:> sqrt (25)
ans = 5
```

Program se ukončí pomocí příkazů `exit,` nebo `quit.` Použití těchto příkazů se odvíjí od verze programu [2][4].

2.4.1 Kalkulačka

Základní funkce programu může být nazvána jako „lepší kalkulačka“ s možností grafického výstupu. Za výhodu proti programovacím jazykům jako C++, C# lze považovat fakt, že není nutné definovat žádné proměnné, a stejně jako u kalkulačky se čísla zapíší do příkazové řádky. Octave si sám určí hodnoty těchto proměnných a přiřadí jim typ proměnné, jako například double atd. Pro příklad si lze uvést sečtení dvou čísel [2][4]:

```
octave:> 5+12
ans = 17
octave:>
```

V případě, že uživatel chce potlačit výstup a nechce vidět výsledek provedeného příkazu, řádek kódu se ukončí pomocí středníku [2][4]:

```
octave:> 5+12;
octave:>
```

I když na příkazové řádce není vidět žádná výsledná hodnota, výsledek součtu je uložen v paměti. Název, typ a hodnota součtu je zobrazena v oblasti 2 znázorněné na obrázku 1. Občas je možné, že Octave vypíše nečekaný výsledkem operace. Například při použití funkce `sqrt` (odmocnina) se záporným argumentem [2][4].

```
octave:> sqrt (-25)
ans = 0+5i
```

Octave spočítal výsledek v komplexních číslech. Toto může být matoucí, pokud uživatel očekává chybový výstup. Tato vlastnost Octavu je výhodná pro automatickou detekci typu proměnných [2][4].

2.4.2 Matice

Octave je naprogramován pro práci s maticemi, proto je každá zadaná proměnná a každé zadané číslo, bráno jako matice o určité velikosti. V praxi to vypadá takto [1][2][4]:

```
octave:> a=6
octave:> a=[6]
```

Tyto dva zápisy jsou implicitní. Ve výše uvedeném případě je v proměnné „*A*“ uloženo jediné číslo a toto číslo je bráno, jako matice (1,1). Výpočty s těmito maticemi probíhají stejně, jako s obyčejnými čísly. V příkazovém řádku není potřeba u takovýchto hodnot psát hranaté závorky. Zápis s hranatou závorkou je formálně korektnější, ale pro efektivitu a rychlost práce se nepíše. Taktéž na program nemá žádný vliv. Výpis takto definovaných proměnných se provede příkazem `who`, nebo `whos` [1][2][4].

```
octave:> whos
Variables in the current scope:
Attr Name           Size        Bytes       Class
      A             1x1         8           double
      B             1x3         24          double
Total is 4 elements using 32 bytes
```

Při použití příkazu `whos` se zobrazí tabulka se všemi definovanými proměnnými, jejich názvy, velikosti a typy. Nyní je možné definovat matici. Tuto definici bude provedena pomocí hranatých závorek „[]“ [1][2][4]:

```
octave:> [1 2 3 4]
octave:> [1, 2, 3, 4]
x= 1 2 3 4
```

Takto vypadá definice řádková matice. Uvedené dva zápisy jsou naprosto implicitní. K oddělení prvků se používá mezera či čárka. Definice sloupcové matice je možno zapsat jako [1][2][4]:

```
octave:> [1; 2; 3; 4]
x=
   1
   2
   3
   4
```

Při zadávání matic je potřeba dbát na to, aby v každém řádku byl stejný počet čísel, jako ve sloupci. Jinak zadání takové matice skončí chybovým výstupem [1][2][4].

```
octave:> M= [1 2 3; 4 5 6; 7 8 9]
M=
   1   2   3
   4   5   6
   7   8   9
```

S maticemi je možno provádět všechny známé operace. Pro příklad je uvedeno sčítání a násobení matic. Nutno dbát na pravidla operací s maticemi [1][2][4].

```
octave:> [1 2; 4 5] + [2 5; 8 6]
ans=
       3    7
       12   11

octave:> [1 2; 4 5] * [2 5; 8 6]
ans =
        18   17
        48   50
```

Octave nabízí mnohé další operace jako odčítání, umocňování reálným číslem, operaci transpozice atd. Další možné operace jsou mezi maticemi a skalárem [1][2][4]:

```
octave:> 2* [1 2 3 ]
ans =
   2   4   6

octave:> [1 2 3] + 4
ans =
   5   6   7
```

Mezi tyto operace patří násobení, nebo součet matice se skalárem. Někdy je možné se setkat se situací, kdy je potřeba provést operaci s maticí tzv. po prvcích. Tato se varianta se provede pomocí tečky před symbolem operace. Tečka před znaménkem napoví, že operace bude provedena se stejným prvkem matice [1][2][4].

```
octave:> [1 2; 4 5].*[2 5; 8 6]
ans =
   2   10
   32   30
```

Při porovnání matice pro standartní násobení a matice pro násobení prvek po prvku, je možné vidět, že výsledné matice se liší. Toto nám potvrdí fakt, že zápisy představují jinou metodu řešení [1][2][4].

```
octave:> [1 2; 4 5].*[2 5; 8 6]
ans =
   2   10
   32   30
```

```
octave:> [1 2; 4 5] * [2 5; 8 6]
ans =
18   17
48   50
```

Stejný zápis se může použít pro další operace např. umocnění po prvcích. Pro práci s maticemi se mohou hodit i vestavěné funkce, jako [1][2][4]:

- výpočet inverzní matice `inv`
- hodnost matice `rank`
- determinant matice `det`
- exponenciela matice `expm`

2.4.3 Skripty

Octave může poskytnout mnohem zajímavější a širší uplatnění, než jen jako kalkulačka. Pro jeho širší využití se používají skriptovací vlastnosti. Je to podobné, jako u programovacích jazyků. Napíše se více příkazů, které provedou příslušnou operaci najednou. U Octavu se rozlišují dvě varianty, skripty a funkce. Skript je ve své podstatě souhrn příkazů, který se chová naprosto stejně, jako kdyby se tyto příkazy zapsali do příkazového řádku, i výsledky budou totožné. Funkce mají větší uplatnění, protože jsme schopni předávat jim vstupní parametry, a výsledky funkce předávat jako nové parametry dalším funkcím.

Skript plní funkci obyčejného přenesení příkazů z příkazové řádky do textového souboru, který se vyvolá pomocí jména skriptu. Pro vytvoření skriptu klikneme na `file-new-new script`, popřípadě použijeme klávesovou zkratku `Ctrl+n`. Velkou nevýhodu skriptu je, že má přístup ke všem proměnným, které byly vytvořeny i vně jeho těla. Tato nevýhoda se může projevit přepsáním různých proměnných, což může způsobit špatné výsledky výpočtů. Níže je uvedena šablona jednoduchého skriptu, který vypočítá rychlost ze dvou proměnných, dráhy a času [1][2][4]:

```
#Vypocet rychlosti
#Parametry pro vypocet
draha=hodnota;
cas=hodnota;
#vypocet
rychlost=draha/cas;
vypis reseni
```

Skript byl uložen pod jménem *Rychlost.m.* Vyvolání skriptu se provádí napsáním jeho názvu do příkazové řádky bez koncovky. m:

```
octave:> Rychlost
rychlost = výsledek
```

2.4.4 Funkce

Funkce se liší od skriptu zásadním slovem `function`. Uvádí se na začátku zdrojového kódu a ihned po něm následuje jméno funkce. Další velkou odlišností od skriptu je přístup

k proměnným. Funkce může pracovat jedině s proměnnými, které byly deklarovány v jejím těle, pracuje tedy s lokálními proměnnými. To nám zajistí bezpečnost globálních proměnných. Pro ukázku je níže uvedena šablona totožné funkce jako v předchozím případě [1][2][4]:

```
function Rychlost(draha, cas)
  #Funkce vypocte okamzitou rychlost
  #reseni
  rychlost=(draha/cas)
endfunction
```

Při vyvolání této funkce příkazem `Rychlost` a jejími parametry se nám zobrazí výsledek operace. Zde je vidět hlavní rozdíl mezi funkcí a skriptem. Skript má pevně zadané hodnoty, ale funkce se vždy při svém spuštění ptá na vstupní parametry.

```
octave:> Rychlost (parametr 1, parametr 2)
rychlost =  vysledek
```

Toto je jedna z možných definic. Další definicí funkce, může být funkce bez parametru, na který se bude následně zadávat až po spuštění.

```
function rychlost
  #Funkce vypocte okamzitou rychlost
  input draha;
  input cas;
  rychlost=(draha/cas);
endfunction
```

V uvedené funkci byl použit příkaz input, který pozastaví funkci a vyčká na zadání požadované hodnoty pro následující výpočet.

Při psaní skriptů a funkcí by mělo být bráno na zřetel několik pravidel, podle kterých pro bude práce snadnější [1][2][4]:

- psaní komentářů, velice usnadní a zpřehlední kód,
- jeden soubor může obsahovou pouze jednu funkci,

- jméno skriptu a funkce nesmí být totožné,

2.4.5 Práce se soubory

Pro hlavní práci se soubory se používá příkaz `load` a `save`. Díky nim je možno načítat data a proměnné, nebo naopak je můžeme ukládat. K uložení dat se používá příkaz `save`. Pro uložení matice prvočísel *„P“* a matice *„M“,* která bude obsahovat druhé mocniny těchto prvočísel, do proměnné vektor, se použijí následující řádky kódu [1][2][4]:

```
octave:> P=[1 2 3];
octave:> M=[1 4 9];
octave:> vektor=[M ‘ X’]
octave:> save vektor
octave:> save –ascii “vector.txt” vektor
```

Následně po definici dvou matic *„P“* a *„M“,* byly tyto matice sloučeny a uloženy dvojím způsobem. První způsob `save vektor`, nám vytvoří soubor v aktuální složce, ovšem kód není uživatelsky k přečtení v textovém editoru. Octave v tomto případě uložil obsah vektoru do binárního souboru. Pro uložení kódu tak, aby byl následně čitelný, použijeme druhý zápis, který nám vektor uloží jako čistý text [2][4].

Varianta binárního kódu se využívá v situacích jako:

- uložení více proměnných
- uložení názvů proměnných
- maximální přesnost proměnných
- úspora datového prostoru

Ascii kód se používá při požadavku:

- čitelnosti dat jako textu

Načtení probíhá obdobným způsobem pomocí příkazu `load` s názvem požadovaného souboru [2][4]:

```
octave:> load vektor.txt
```

Při nezdaru načítání, vypíše Octave chybovou hlášku. Toto chování může být zapříčiněno jiným umístněným souboru, než je kořenová složka. V takovém případě se využívám oblast jedna, na obrázku 1, kdy se pomocí kurzoru vyhledá požadovaný soubor. Vyhledávání v příkazové řádce a práce se soubory probíhá za pomoci následujících příkazů [2][4]:

- `ls:` výpis aktuální složky
- `cd:` otevření požadovaného adresáře
- `cat:` výpis souboru
- `pwd:` aktuální složka
- `mkdir:` vytvoření adresáře
- `rmdir:` smazání adresáře

2.4.6 Grafický výstup

Octave pro vykreslení grafu používá rozšíření jako Octaviz a Octplot. Častěji se však používá externí program Gnuplot. Díky Gnuplotu se otevírá možnost nejen vykreslení dat, ale i prokládání křivkami, nelineární fitování atd. Nutno říci, že je potřeba mít Gnuplot v Pc nainstalován pro případ, kdy není součástí instalačního balíčku Octave 4.0. Nejnovější verze Octave již obsahuje tento program pro vykreslení dat. Vykreslení dat spustíme např. pomocí příkazu `plot ()`. Tento příkaz zajistí výstup 2D grafu s rozsáhlým počtem úprav a modifikací. Nejjednodušší forma zápisu je uvedena níže [2][4]:

```
octave:> plot(x)
```

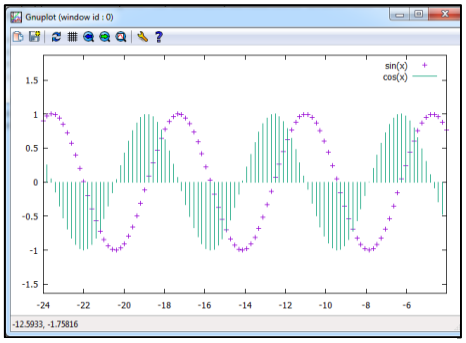
Proměnná „*x*“ musí obsahovat hodnoty, v opačném případě se spustí grafický výstup, který ale bude prázdný, což bývá častou chybou. Příkaz `plot` je multifunkční a umožnuje mu číst více argumentů najednou [2][4]:

```
octave:> plot(x,sin(x),x,cos(x))
```

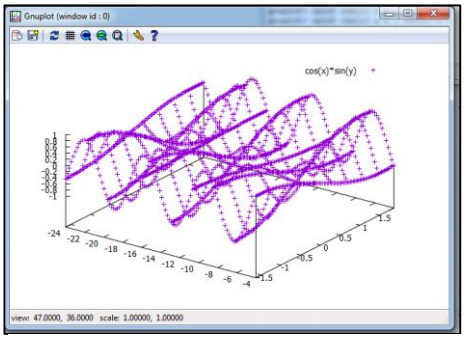
Gnuplot nabízí velké možnosti při úpravách grafů pro jejich větší přehlednost a srozumitelnost, jako styl čáry:

- bodová: přípona `w p`, zkratka pro „with points“
- plná s body: přípona `w lp`, zkratka pro „with linepoints“
- impulsová: přípona `w i`, zkratka pro „with impuls“

Pro 3D grafy se používá funkce `splot`, nebo `3dplot` která má stejné možnosti jako funkce plot. Návod pro správnou syntaxi a používání všech možných úprav grafů je uveden v nápovědě. Nápověda je vyvolána pomocí příkazu `help plot`. Syntaxe a ukázkové grafické výstupy jsou uvedeny níže na obrázcích [1][2]:



Obrázek 4.: Ukázka funkce plot



Obrázek 5.: Ukázka funkce splot

Obrázek 4 znázorňuje využití příkazu `plot` pro funkci s předpisem sin (x) w p, cos (x) w. Užití příkazu `splot` pro funkci cos (x) * sin (y) je znázorněno na obrázku 5.

2.5 Praktické využití

2.5.1 Skript, Funkce

Jako první si lze uvést rozdíly mezi skriptem a funkcí. Tyto rozdíly jsou uvedeny níže na konkrétních příkladech. Následující zdrojový kód reprezentuje skript, který provede matematické operace mezi zadanými maticemi.

```
# skript matice_1 #
```

```
1 disp('Tento script zobrazi nasledujici operace pro dve matice.')
2 disp('Jsou zadany dve matice 3x3.')
3 format short
4 A=[5,6,7;3.5,4,9;1.3,4.8,6]
5 B=[7,6.4,8;4.1,6,8;4.8,5,3]
6 Soucet=A+B
7 Odcitani=A-B
8 Nasobeni=A*B
9 disp('Transponovana matice A.')
10 A'
11 disp('Determinant matice B.')
12 det(B)
13 disp('Konec scriptu.')
```

Ze zdrojového kódu lze vidět, že první dva řádky obsahují příkazy `disp`. Na obrazovku se vypíší texty v závorkách. Následně skript obsahuje dvě předem zadané náhodné matice formátu `short`, tudíž není požadována žádná interakce s programem. Dále jsou uvedeny operace, které budou provedeny se zadanými maticemi. Posledním řádkem kódu je výpis o konci skriptu. Spuštění skriptu se provádí v příkazovém řádku pomocí jeho jména. Do příkazové řádky se vypíše `matice_1`. Po zadání tohoto příkazu se na displeji objeví výpis [1][2]:

```
octave:> matice_1

  Tento script zobrazi nasledujici operace.
  Jsou zadany dve matice 3x3.

  A =

  5.0000   6.0000   7.0000
  3.5000   4.0000   9.0000
  1.3000   4.8000   6.0000

  B =

  7.0000   6.4000   8.0000
  4.1000   6.0000   8.0000
  4.8000   5.0000   3.0000

  Soucet =

  12.0000   12.4000   15.0000
  7.6000   10.0000   17.0000
  6.1000    9.8000    9.0000
```

```
  Odcitani =

  -2.00000  -0.40000  -1.00000
  -0.60000  -2.00000   1.00000
  -3.50000  -0.20000   3.00000

  Nasobeni =

  93.200   103.000   109.000
  84.100    91.400    87.000
  57.580    67.120    66.800

  Transponovana matice A.
  ans =
  5.0000   3.5000   1.3000
  6.0000   4.0000   4.8000
  7.0000   9.0000   6.0000

  Determinant matice B.
  ans = -53.360
  Konec scriptu.
```

Stejný příklad uvedený jako funkce. Funkce byla nazvána `matice_2`.

```
  # funkce matice_2 #

1 function  matice_2
2 disp('Zadejte prosim matice A a B: ')
3 # inputy
4 A=input('A:  ')
5 B=input('B:  ')
6 # vypocty
7 Soucet=A+B;
8 Odcitani=A-B;
9 Nasobeni=A*B;
10 # vypisy
11 disp ('Soucet matic je: '), disp(Soucet)
12 disp('-----')
13 disp ('Odcitani matic je: '), disp(Odcitani)
14 disp('-----')
15 disp ('Nasobeni matic je: '), disp(Nasobeni)
16 disp('-----')
17 disp('Transponovana matice A.')
18 A'
19 disp('Determinant matice B.')
20 det(B)
21 endfunction
```

Rozdílu si lze povšimnout ihned na prvním řádku. Tělo funkce je ohraničeno příkazy `function` a `endfunction`. Dalším markantním rozdílem je, že funkci již interaguje s uživatelem. Po spuštění bude vyžádáno zadání dvou matic. Tyto příkazy jsou uvedeny na 4. a 5. řádku. Zbytek funkce již odpovídá předešlému skriptu. I výpis na obrazovku bude stejný. Funkce má několik dalších odlišných vlastností. Jedna z důležitých je ta, že všechny proměnné definované ve funkci jsou lokální. Mimo tělo funkce a po jejím skončení jsou již nedostupné. Funkci se vyvolá příkazem `matice_2`. Výše uvedená funkce nemá parametr. Může být vytvořena i parametrické funkce. Pro příklad je použita stejná funkce, s tím rozdílem, že operace se bude provádět pro dvě čísla x a y. Následující funkce s parametrem se vyvolá, jako `matice_2 (x, y)`, kde čísla v závorce jsou parametry. Dá se říci, že skript ve svém těle obsahuje příkazy stejné, jako z příkazové řádky, které jsou předem definované a nemohou být uživatelem ovlivněny. Po zavolání skriptu dojde k výpisu na obrazovku. Funkci lze označit za výhodnější z hlediska dynamického programování. Lépe se hodí pro interakci s uživatelem. Pomocí funkcí je možnost tvoření programů a aplikací pro různé vstupní parametry [1][2][4].

2.5.2 Balistická křivka

Funkce balistika_1.m byla naprogramována pro zobrazení balistické křivky s odporem prostředí. Jedná se o interaktivní funkci, která má následující zadání. Střelec vystřelí z počátku osových souřadnic kulový projektil o počáteční úsťové rychlosti *v0* pod elevačním úhlem *el_uhl*. Jako prostředí s odporem pro náš projektil byl vybrán vzduch. Součinitel odporu, který závisí na tvaru našeho projektilu, je v tomto případě *C=0,85*. Funkce se na všechny tyto parametry dotáže a po jejich zadání se zobrazí graf balistické křivky. Jedná se již o složitější diferenciální funkci, která ve svém těle obsahuje více kroků, které si popíšeme. Fyzikální podstata úlohy vychází z Newtonova vztahu uvedený v (Kvasnica, Havránek, Lukáč, Sprušil. *Mechanika.* : Academia. 2004. Kapitola 2, Dynamické částice, s. 57-128) [7][10]:

$$F_d = \frac{-C\rho S v^2}{2} \tag{1}$$

Zde vidíme zmíněnou konstantu *C,* součinitel odporu. Hodnotu hustoty prostředí ρ, S zastupující příční průřez střely. V tomto případě průřez kulového tělesa $S = \pi * r^2$. Parametr k získáme z výpočtu $k = \frac{C\rho Sv}{2}$. Pro vyřešení této úlohy dostáváme následující dvě diferenciální rovnice [7].

$$m\frac{d^2x}{dt} = -kv_x^2 \tag{2}$$

$$m\frac{d^2y}{dt} = -mg - kv_y^2 \tag{3}$$

Kde rovnice číslo dvě reprezentuje pohyb ve směru osy *„x“* a rovnice číslo 3 pohyb ve směru osy *„y“*. Pro tuto úlohu uvažujeme vyšší rychlosti, v řádu stovek m/s, z důvodu přiblížení reálným úsťovým rychlostem projektilů. Pro nižší rychlosti by následující postup nefungoval [7].

```
  # funkce balistika_1 #

1 function balistika_1
2 clear;
3 clc;
4 disp('Tato funkce vypocte a vykresli balistickou krivku
  projektilu.')
5 disp('Nyni prosim zadejte nasledne promenne.')
6 v0=input('Pocatecni rychlost projektilu [m/s]:  ');
7 el_uhl=input('Elevacni uhel [v uhlech]:  ');
8 # prepocet uhlu na radiany
9 rad=(el_uhl*pi)/180;
10 # pocatecni podminky
11 t0=0;
12 x0=0;
13 y0=0;
14 disp('Pocatecni podminky: ')
15 disp('Cas: '), disp(t0)
16 disp('Souradnice x: '), disp(x0)
17 disp('Souradnice y: '), disp(y0)
18 # parametry kulky
19 disp('Prosim zadejte parametry projektilu.')
20 r=input('Polomer [cm]:  ');
21 m=input('Hmotnost [g]:  ');
22 # parametry prostredi
23 disp('Parametry prostredi: ')
```

```
24 g=9.81 # tihove zrychleni
25 ro=1.3 # hustota vzduchu
26 C=0.85 # soucinitel odporu (zavissi na tvaru)
27 # vypocty
28 S=pi*r^2; # plocha prurezu projektilu
29 k=C*ro*S/2; # celkova zavislost odporu prostedi pro projektil
30 # vypocet rychlosti projektilu v prostredi
31 vx=v0*cos(rad);
32 vy=v0*sin(rad);
33 v=sqrt(vx^2+vy^2);
34 # integrace Eulerovou metodou
35 # pocatencni data
36 t(1)=t0;
37 x(1)=x0;
38 y(1)=y0;
39 i=1;
40 # integracni krok
41 dt = 0.05;
42 # vlastni cyklus testujici dopad na osu y, po dopadu se ukonci
43 while (y(i)>=0)
44 i=i+1;
45 x(i)=x(i-1)+vx*dt;
46 y(i)=y(i-1)+vy*dt;
47 ax=-k*v*vx/m;
48 ay=-g-k*v*vy/m;
49 vx=vx+ax*dt;
50 vy=vy+ay*dt;
51 v=sqrt(vx^2+vy^2);
52 t(i)=t(i-1)+dt;
53 max_i=i;
54 endwhile
55 # tisk funkce + popisy
56 plot(x,y,'+;krivka;');
57 grid off
58 xlabel('x[m]')
59 ylabel ('y[m]')
60 title('Balisticka krivka_1')
61 endfunction
```

Zdrojový kód této funkce o délce 61 řádků má několik kroků. Hlavními kroky jsou:

- ř. 4-26: zadání a definice parametrů,
- ř. 27-33: výpočet mezihodnot,
- ř. 34-41: počáteční podmínky integrace,
- ř. 24-54: cyklus pro vlastní integraci,
- ř. 55-61: cyklus pro grafický výstup,

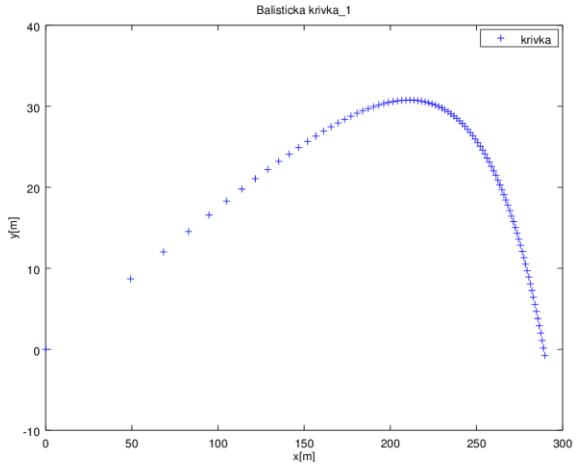
Pro tento konkrétní příklad byly zvoleny následující hodnoty:

- počáteční rychlost kulky $v = 1000\ m \cdot s^{-1}$,
- elevační úhel kulky $\lambda = 10°$,
- poloměr střely $r = 0.5\ cm$,
- hmotnost střely $g = \ 20\ g$,

Po potvrzení poslední zadané proměnné hmotnosti střely, funkce vypíše parametry prostředí, jako tíhové zrychlení, hustotu a součinitel odporu vzduchu a následně se zobrazí výstupy:

```
Tato funkce vypocte a vykresli balistickou krivku projektilu.
Nyni prosim, zadejte nasledne promenne.
Pocatecni rychlost projektilu [m/s]:  1000
Elevacni uhel [v uhlech]:  10
Pocatecni podminky:
Cas:
0
Souradnice x:
0
Souradnice y:
0
Prosim, zadejte parametry projektilu.
Polomer [cm]:  0.5
Hmotnost [g]:  20
Parametry prostredi:
g =  9.8100
ro =  1.3000
C =  0.4800
```

Tato funkce je příkladem dynamického modelování a užití programu GNU Octave ve fyzice. Balistika_1.m může být použita pro různé vstupní parametry, jejímž hlavním výstupem je obrázek 6, znázorňující balistickou křivku pro definovaný projektil.



Obrázek 6.: Balistická krivka_1

Nicméně tato funkce neznázorňuje maxima. Tím je myšlena vzdálenost, do které projektil doletěl, osa *„x“* a výšku, které dosáhl, osa *„y“*. To jsou důležité údaje, které jsou požadovány po této funkci. Mezi další nevýhody může být zařazeno nepřehlednost grafu a statistická definice proměnné *C*. Změny pro modifikaci funkce se provedou pomocí příkazu `edit balistika_1.m` [7].

Hlavními body modifikace budou:

1) dynamické zadávání proměnné *C* s výpisem,

2) funkce pro výpočet maxim pro osu *„x“* a *„y“*,

3) změna stylu grafického výstupu pro otestování jeho možností,

Tyto modifikace jsou zaměřeny na vlastní tvůrčí schopnosti. Po modifikaci zdrojového kódu lze provést kontrolu se vzorovým řešením, které je uvedeno v kapitole Modifikace zdrojových kódů.

2.5.3 Matematické kyvadlo

Matematické kyvadlo patří k velice známým zjednodušením reálného kyvadla, kde předmětem zájmu, je pouze hmotný bod. Další fyzikální veličiny se zanedbávají. Těmi jsou hmotnost závěsného lanka a odpor prostředí. Gravitační pole je považováno za homogenní a tření závěsu nebereme v potaz. Příklad je věnován simulaci tohoto kyvadla při zadání

počáteční výchylky. Tato funkce pomocí grafického výstupu zobrazí kmity kyvadla. Pro ulehčení práce se předpokládá, že kyvadlo má kmit 1s. Díky tomuto předpokladu musí pro délku kyvadla platit vzorec 4 [7][8].

$$l = \frac{g}{4\pi^2} \approx 0{,}25m \tag{4}$$

Diferenciální rovnice popisující tento model má tvar:

$$\frac{d^2\varphi}{dt^2} = -\frac{g}{l} sin\varphi \tag{5}$$

V uvedené rovnice se definují proměnné, jako *g* tíhové zrychlení, *l* délka závěsu, která byla pomocí vzorce 4 určena na 0, 25m. Zdrojový kód této funkce byl uložen pod názvem *kyvadlo_1* [7]:

```
  # funkce kyvadlo_1 #

1  function kyvadlo_1
2  clear;
3  clc;
4  disp('Tato funkce vypocte a vykresli Matematicke kyvadlo se zadanou
   vychylkou.')
5  disp('Nyni prosim zadejte nasledne promenne.')
6  vych=input('Veliskost vychylky [v uhlech]:  ');
7  # prepocet uhlu na radiany
8  rad0=(vych*pi)/180;
9  # pocatecni podminky
10 g=9.81;     #  tihove zrychleni
11 l=0.25; #  delka kyvadla
12 der0=0;
13 t0=0;
14 i=2;
15 # vektor rychlosti
16 v=[];
17 # derivace
18 t(1)=0;
19 rad(1)=rad0;
20 der(1)=der0;
21 # integrace
22 int=5;          # integracni mez
23 h = int/500;    # integracni krok
24 # integrace metodou 2. radu
```

```
25 while t<int
26 k1=-g/l*sin(rad(i-1));
27 php=rad(i-1)+der(i-1)*2*h/3+k1*2*h^2/9;
28 k2=-g/l*sin(php);
29 rad(i)=rad(i-1)+der(i-1)*h+(k1+k2)*h^2/4;
30 der(i)=der(i-1)+(k1+3*k2)*h/4;
31 t(i)=t(i-1)+h;
32 i=i+1;
33 endwhile
34 # maximalni vychylka
35 v=[v;der];
36 max_v=max(v);
37 disp('Maximalni vychylka v radianech:')
38 disp(max_v)
39 # cyklus pro plot
40 v=[v;der];
41 for j=1:columns(rad)
42 grid on
43 plot(v(j,:),t,'r;kmity;')
44 xlabel('omega[rad/s]')
45 ylabel('t[s]')
46 title('Matematicke kyvadlo_1');
47 endfor
48 endfunction
```

Z uvedeného zdrojového kódu lze vidět, že kyvadlo bylo naprogramováno pomocí bezparametrické funkce, kdy je dotazováno na zadání počáteční výchylky. Hodnota výchylky je převedena na radiány a následně se pomocí předem zadaných konstant provede požadovaný výpočet. Struktura kódu je následující:

- ř. 4-16: zadání a definice parametrů,
- ř. 17-20: definice hodnot derivace,
- ř. 21-23: počáteční podmínky integrace,
- ř. 24-33: cyklus pro vlastní integraci,
- ř. 34-38: výpočet maximální výchylky,
- ř. 39-48: cyklus pro grafický výstup,

Pro podrobnější pochopení integrační metody na řádcích 25-32 je nutno nastudovat teorii o samotné integrační metodě (Rektorys, *Přehled užité matematiky II*. Praha : Prometheus, 2003. 874 s. ISBN 80-7196-181-7). Zdrojový kód byl nejprve napsán a následně

zkompletován vložením části s integrační metodou. Tento kód není zaměřen, na odborné zpracovaní diferenciálních rovnic, nýbrž na pochopení struktury, syntaxe a psaní zdrojového kódu pro výpočet těchto rovnic. Pro funkci je zadána počáteční výchylku 10 stupňů. Tomuto zadání bude odpovídat následující výstup na displej a grafický výstup:

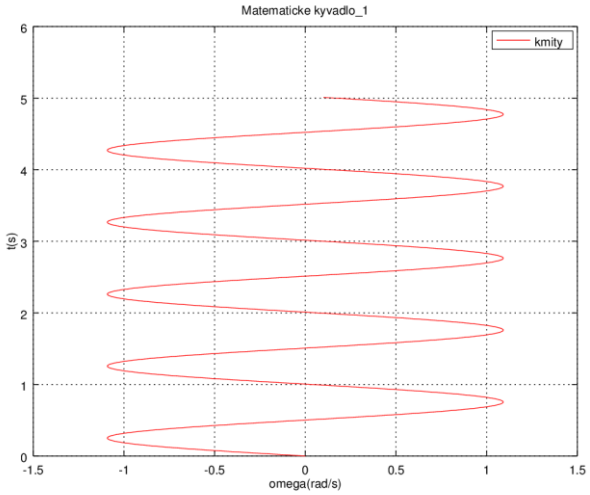
```
Tato funkce vypocte a vykresli Matematicke kyvadlo se zadanou
vychylkou.

Nyni prosim zadejte nasledne promenne.

Veliskost vychylky [v uhlech]:  10

Maximalni vychylka v radianech:

 1.0918
```



Obrázek 7.: Matematické kyvadlo_1

Výsledný výstup obsahuje krátkou informaci o účelu funkce. Následné vyzvání pro zadání počáteční výchylky a závěrečný grafický výstup. Poslední informací je maximální dosažená výchylka, kterou je uvedena na obrázku 10. Znázorněná křivka se podobá oscilaci a to díky zanedbání parametrů prostředí. Díky tomu lze převést matematické kyvadlo na mechanický oscilátor. Výsledným grafickým výstupem je graf oscilací. Pro úhly do 5º, je možno aproximovat průběh kmitání na průběh funkce sinus. I pro tuto úlohu jsou uvedeny cvičební modifikace. Tyto úpravy se budou týkat druhu funkce a změny grafického výstupu.

1) modifikace na parametrickou funkci,

2) změna stylu grafického výstupu,

2.6. Modifikace zdrojových kódů

Modifikace zdrojových kódů uvedených níže jsou pouze jednou z mnoha metod řešení daných problémů. Všechny zdrojové kódy jsou uloženy v příloze k této práci.

2.6.1 Balistická křivka 2

Dle zadání byly modifikovány tři body funkce balistika_1.m. Zdrojový kód celé funkce naleznete v příloze pod názvem balistika_2.m.

První modifikace se týkala proměnné *C*. Dynamické zadání této proměnné, pro větší využitelnost funkce, se provedlo pomocí následujícího řádku zdrojového kódu:

```
30 C=input('Zadejte prosim koeficient odporu: ')
```

Tento zdrojový kód zajistí výpis závorky na displej a program čeká na zadání hodnoty, která bude uložena do proměnné „*C*“. Tento příkaz, se nachází na řádku č. 30.

Důležité modifikace pro funkce, které vypočtou a zobrazí výsledné maximální hodnoty na displej, mohou být zapsány následovně:

```
68 # maximalni hodnota x
69 max_x=max(x);
70 disp('Projektil doletel do vzdadlenosti '), disp(max_x)
71 # maximalni hodnota y
72 max_y=max (y);
73 disp('Projektil dosahl maximalni vysky '), disp(max_y)
```

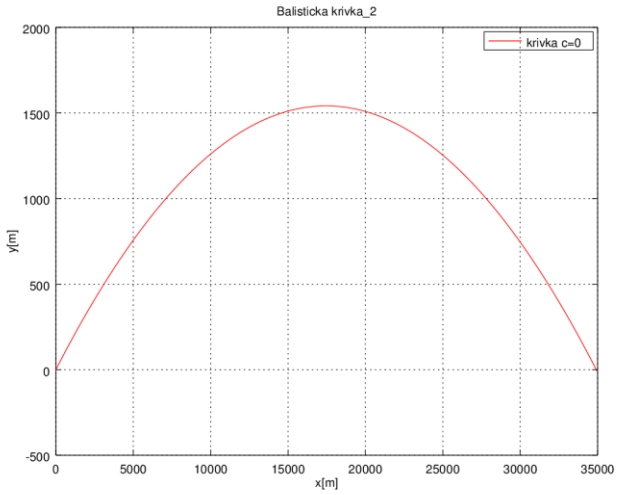
Ve zdrojovém kódu byly definovány nové proměnné, *max_x* a *max_y*, které používají funkci `max (parametr)`. Tato funkce prohledá matici hodnot zadaného parametru a vyhledá jeho maximum, které se následně uloží do našich proměnných. Poté již pomocí příkazu `disp`, neboli výpisu na displej, se hodnoty zobrazí.

Grafický výstup má širokou škálu možností. Pro jejich procvičení je uvedena modifikace pro obrázek 9 s pomocí základních příkazů pro lepší přehlednost:

```
75 # tisk funkce + popisy
76 plot(x,y,'r;krivka c=0;');
77 grid on
78 xlabel('x[m]')
79 ylabel ('y[m]')
80 title('Balisticka krivka_2')
```

Modifikace spočívá v převedení jednotlivých bodů do souvislé čáry. Této změny se dosáhne tím, že příkaz + uvedený v první úloze se smaže. Dále na stejnou pozici byl vložen příkaz `r` v uvozovkách na řádku 76. Změna se projevila tím, že byl nastaven výchozí styl čáry a příkazem *red*, byla tato čára obarvena na červeno. Další změnou na tomto řádku byla modifikace popisu křivky, kde jako legenda byla nastavena *krivka c=0*. Pomocí příkazu `grid on`, byla zobrazena souřadnicová mřížka. A poslední změnou byla změna názvu grafu, pomocí přepsání výpisu na řádce 80. Zadané hodnoty pro druhý příklad se nezměnili, až na parametr *C*, který byl nastaven na 0. Díky této změně může být očekáván parabolický průběh letu kulky. Výsledný grafický výstup a výstup na displej vypadá následovně [9]:

```
Tato funkce vypocte a vykresli balistickou krivku projektilu.
Nyni prosim zadejte nasledne promenne.
Pocatecni rychlost projektilu [m/s]:  1000
Elevacni uhel [v uhlech]:  10
Pocatecni podminky:
Cas:
0
Souradnice x:
0
Souradnice y:
0
Prosim zadejte parametry projektilu.
Polomer [cm]:  0.5
Hmotnost [g]:  20
Parametry prostredi:
Zadejte prosim koeficient odporu: 0
Kulka doletela do vzdalenosti 34960.6752
Kulka dosahla maximalni vysky 1541.2268
```



Obrázek 8.: Balistická křivka_2

Nyní modifikovaná funkce již podává konkrétní informace o vzdálenosti a dosažené výšce projektilu. Změna grafického výstupu je již přehlednější a více vypovídá o průběhu letu.

2.6.2 Matematické kyvadlo 2

Stejně jako v předchozí úloze byla pomocí příkazu *edit* upravena a uložena funkce s novým názvem kyvadlo_2.m. Hlavní modifikace se týkala změny typu funkce. Z bezparametrické funkce byla vytvořena funkce parametrická. Tím bylo ušetřeno pár řádků kódu, a celkové volání funkce je nyní rychlejší. Zmíněnou úpravu reprezentuje zdrojový kód níže:

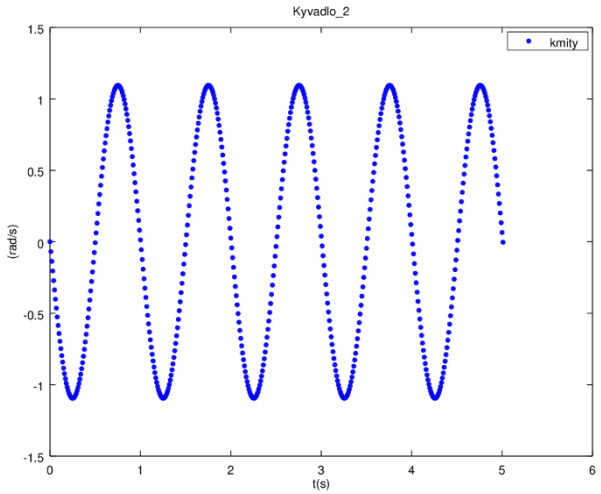
```
1 function kyvadlo_2 (vych)
```

Jednořádkovým příkazem je volána funkce s parametrem `vych=vychylka`. Pro funkčnost kódu je nutné smazání řádků číslo pět a šest. Takto byla úspěšně změněna funkce a zjednodušeno volání. Modifikace grafického výstupu byla požadována z důvodu nepřehlednosti grafu. Změna se týkala směru kmitů, který byl změněn na horizontální. Toho bylo docíleno prohozením datových os na řádce 39. Kvůli této úpravě byly změněny i popisy os. Další úprava se byla provedena u formátu osy. Formát byl modifikován na bodový [7].

```
40 plot(t,v(j,:),'.;kmity;')
```

```
41 xlabel('t(s)')
42 ylabel('(rad/s)')
43 title('Kyvadlo_2');
```

Výsledný grafický výstup je následný:



Obrázek 9.: Matematické kyvadlo_2

2.7 Shrnutí

Předešlé vzorové úlohy a jejich modifikace jsou směřovány na ukázku dynamického modelování, pochopení struktury a syntaxe zdrojového kódu. Úlohy slouží jako příkladné použití různých vnořených funkcí, tak i funkcí pro řešení diferenciálních rovnic. Zde uvedená řešení jsou jen jedním z mnoha možných postupů, které se mohou na dané úkoly aplikovat. Po nastudování těchto materiálů by měl být uživatel schopen napsat vlastní jednoduché funkce pro řešení diferenciálních rovnic, popřípadě být schopen upravit pokročilejší funkce.

3. OpenFOAM

3.1 Historie OpenFOAM

OpenFOAM se používá při spojitém modelování převážně mechaniky tekutin. Numerické metody řešení používané v tomto druhu modelování popisuje např. [14]. Jeho největší výhodou je jeho cena. Jelikož se jedná o open source, náklady za jeho pořízení jsou nulové. Jeho uplatnění je široké a proto ho využívají i známe firmy jako Škoda Auto, Porsche, nebo ZVVL. Software byl vyvinut Henrym Wellerem v roce 1980 na Imperial College v Londýně. Jako platformu umožňující simulace. Program byl napsán pomocí programovacího jazyka C++, aby se docílilo jeho maximální kompatibilitě s ostatními výpočetními systémy. Struktura OpenFOAM stojí na základně rozsáhlých knihoven, z nichž každá nabízí příslušné možnosti zdrojového kódu pro následné tvoření aplikací, či složitých simulací. Knihovny například obsahují řešení pro diferenciální a parciální rovnice a mnohé další fyzikální modely týkající se fyziky. Při dnešní síle výpočetní techniky se tyto programy stávají velkými pomocníky. Pro příklad si uveďme selhání inženýrů v roce 1940, kdy pouhé 4 měsíce po otevření se zřítil most v Tacome v USA. Díky OpenFOAM by tehdejší inženýři měli možnost vzít v potaz aerodynamické vlastnosti mostu. Díky silnému větru se most rozkmital na takovou frekvenci, která narušila strukturu, což způsobilo pád. V dnešní době je samozřejmostí, že se v těchto programech testují struktury staveb, konstrukce strojů, které by bez patřičných zkoušek a výsledných dat nemohly být postaveny, nebo zkonstruovány. Psaní zdrojového kódu probíhá v příkazové řádce, popřípadě pomocí vnořených textových editorů. Výsledný grafický výstup zajišťuje program ParaView [15], který je součástí instalačního balíčku. V tomto programu můžeme analyzovat výstupní data, grafické výstupy a popřípadě je modifikovat [11] [13].

3.2 Instalace

3.2.1 Instalace pro Windows

První verze programu a jeho architektura byly směrovány pro použití na operačních systémech Unix a následně pro Linux. Přesto je možné tento program používat i na OS Windows. Na internetových stránkách následujícího odkazu [13], můžete stáhnout instalační balíček, který dle přiloženého návodu jednoduše nainstalujete. Vývojové prostředí má podobu příkazového řádku, v němž jsou zahrnuty všechny knihovny a potřebné aplikace. Tato instalace je plně podporovaná s aktivním helpdeskem od českých uživatelů. Ti vytvořili tuto variantu programu na popud českých vývojových firem, které pracují s operačním systémem Windows [13].

3.2.2 Instalace pro Linux

Původní verze instalace neboli verze pro OS Linux je dostupná na oficiálních stránkách v odkaze [11]. Máme zde na výběr ze tří možností instalace. Třetí možností je instalace pro Windows. Pro OS Linux máme dvě možnosti, Binary Installation a Source Code Installation. Nejsnadnější variantou instalace je Source Code Installation. Po rozkliknuti této možnosti se otevře jednoduchý návod na instalaci, která byla v minulosti kritizována pro velkou nespolehlivost. Uživatel byl schopen naistalovat na PC program bez problému, ale problém mohl nastat při druhé a dalších instalacích, kdy bez zjevné příčiny program nefungoval, nebo postrádal důležité knihovny. Pro instalaci stačí do příkazové řádky zadat příkaz [11]:

```
$ tar  -xzf  OpenFOAM-v1606 + .tgz
```

Cílovou složkou instalace tohoto programu je domovský adresář *HOME*. Samozřejmě je možnost změnit umístnění instalace, pomocí příkazů uvedených na daných stránkách.

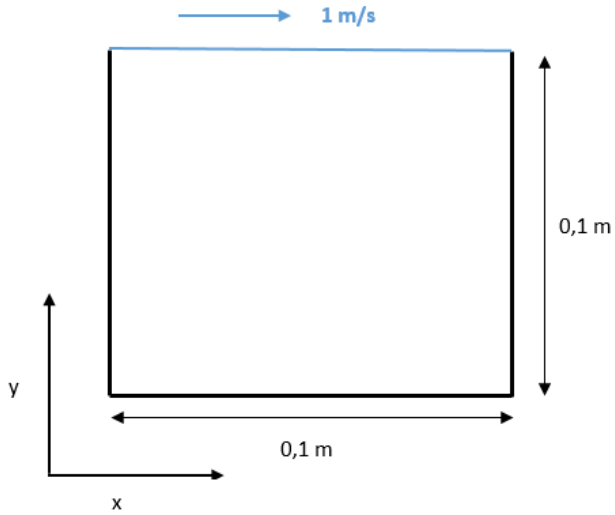
3.3 Práce s OpenFOAM

Práce s tímto programem bez grafického prostředí i samotná struktura a syntaxe kódu je složitější. Vývojáři doporučují navštívit alespoň jeden seminář, nebo školící kurz. Ty však mohou být finančně nákladné, tím pádem velká přednost, neboli cena programu pozbývá smyslu. Další možností je projít, nastudovat a pochopit vzorové příklady, které jsou uvedeny v návodu a jejich zdrojové kódy jsou obsaženy v instalaci. V této práci je uveden právě jeden ze vzorových příkladů, který bude v této práci rozebrán a vysvětlen z hlediska zdrojového kódu, jeho syntaxe. Následně bude uvedena práce s grafickým výstupem. Pro práci budeme využívat před programovaný příklad *Cavity,* kde každá část kódu má vlastní funkci pro následné spuštění celkové simulace [11].

3.4 Praktické využití: nestlačitelné proudění

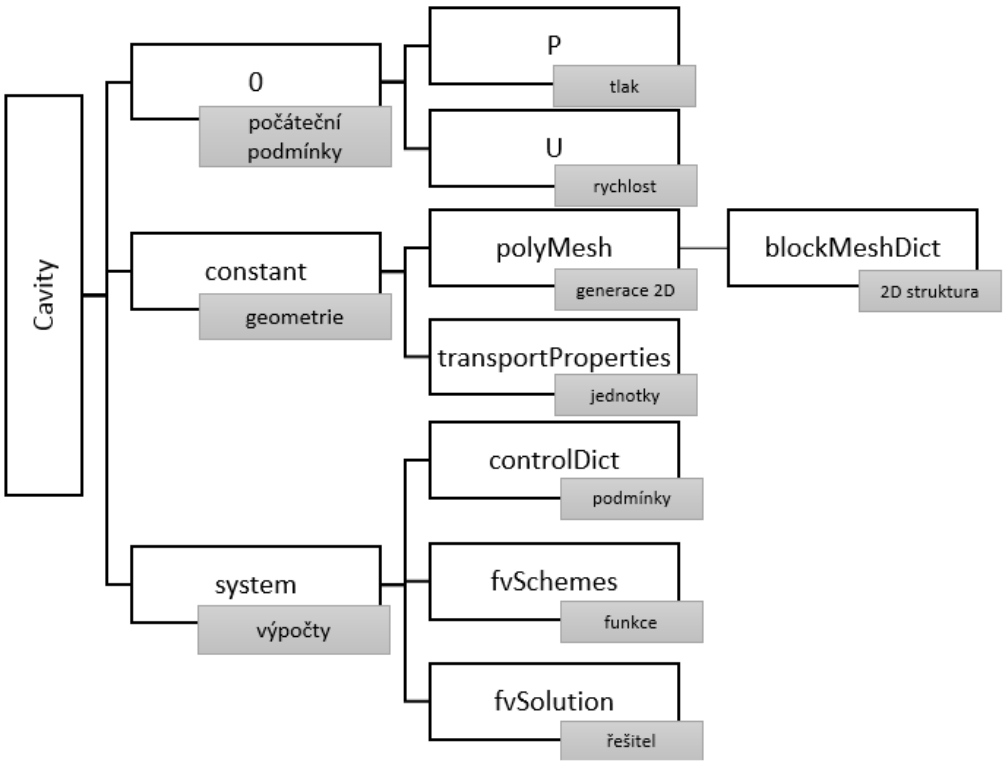
3.4.1 Zadání, struktura

Jako vzor je uveden příklad týkající se problematiky izotermického proudění v dvoudimenzionálním prostoru. V příkladu bude definováno nestlačitelné proudění kapaliny na 2D modelu dané geometrie, které bude simulováno pohyblivou stěnou. Zadanou geometrií bude čtverec o stranách deset centimetrů, kde proudění o velikosti 1 m/s bude umístěno na horní stěnu čtverce, viz obrázek 10 [12]:



Obrázek 10.: OpenFOAM – geometrie

Pro složitost struktury vzorového příkladu je uveden obrázek 11, který podává informaci právě o této struktuře. Ta se skládá z kořenového adresáře nazvaného *Cavity*, který obsahuje složky a textové soubory se zdrojovým kódem [12].



Obrázek 11.: OpenFOAM – struktura cavity

Ze struktury lze vidět, že kořenový adresář *Cavity*, který obsahuje tři hlavní složky. Šedé rámečky na obrázku 14 nesou informaci, která část zdrojového kódu je obsažena na dané pozici a k čemu v úloze slouží. Postupně bude probrán celý kód a všechny adresáře i s jejich obsahem. V případě absence vzorového příkladu *Cavity* se provede vložení pomocí následujícího postupu [12] [13]:

```
# cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity $FOAM_RUN

# cd $FOAM_RUN /cavity
```

Takto byla zkopírována složka cavity do instalace v PC.

3.4.2 BlockMeshDict

Základním stavebním kamenem úlohy je objekt, s kterým se bude pracovat. Zadání geometrie je uvedeno ve složce *polyMesh*, která je uložena ve složce *constant.* Textový soubor *blockMeshDict* nacházející se v této složce, se otevře pomocí několika příkazů. Příkaz `gedit` otevře textový soubor s možností editace textu. Pro prohlížení bez možnosti úpravy se použije příkaz `cat`. Pokud by žádná z variant nefungovala, je možnost tyto soubory nalézt v kořenovém adresáři instalace našeho programu a ve složce *run* ho pomocí dvojkliku otevřít.

```
1. /*---------------------------------*- C++ -*----------------------------------*\
2. | =========                 |
3. | \\      /  F ield         |         OpenFOAM: The Open Source CFD Toolbox
4. |  \\    /   O peration     | Version:  3.0.x
5. |   \\  /    A nd           |         Web:      www.OpenFOAM.org
6. |    \\/     M anipulation  |
7. \*---------------------------------------------------------------------------*/
```

Všechny textové soubory ve všech složkách začínají výše uvedenou hlavičkou na sedm řádků. Pro přehlednost zdrojového kódu je tato hlavička uvedena nyní a později již bude vynechána. Následuje zmíněný zdrojový kód *blockMeshDict* [12]*:*

```
8     FoamFile
9     {
10          version     2.0;
11          format      ascii;
12          class       dictionary;
13          object      blockMeshDict;
14     }
15
16    convertToMeters 0.1;
17
18    vertices
19    (
20       (0 0 0)
21       (1 0 0)
22       (1 1 0)
23       (0 1 0)
24       (0 0 0.1)
25       (1 0 0.1)
26       (1 1 0.1)
27       (0 1 0.1)
```

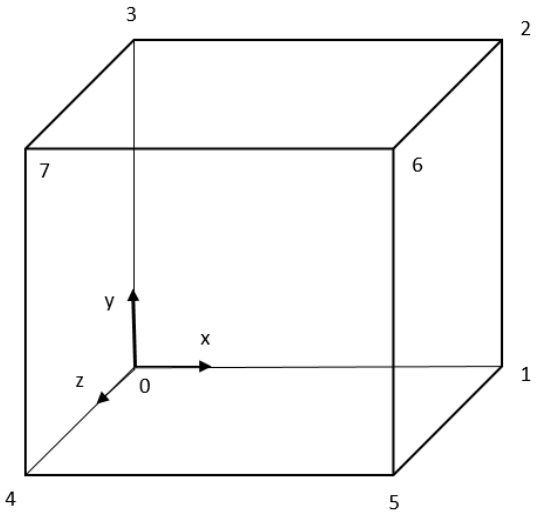
```
28    );
29
30     blocks
31           (
32           hex (0 1 2 3 4 5 6 7) (10 10 1) simpleGrading (1 1 1)
33    );
34
35     edges
36     (
37     );
38
39     boundary
40    (
41       movingWall
42           {
43               type wall;
44               faces
45              (
46                 (3 7 6 2)
47           );
48       }
49       fixedWalls
50           {
51               type wall;
52               faces
53              (
54                 (0 4 7 3)
55                 (2 6 5 1)
56                 (1 5 4 0)
57            );
58       }
59       frontAndBack
60           {
61               type empty;
62               faces
63               (
64                 (0 3 2 1)
65                 (4 5 6 7)
66              );
67           }
68     );
69
70     mergePatchPairs
71     (
72     );
```

Zdrojový kód v rozsahu 73 řádků definuje strukturu.

- ř. 8-13: Informace o zdrojovém kódu.

- ř. 17: Definice jednotky, která pro bude 0,1m = 10cm.
- ř. 19-28: Určení rohů obrazce.

Zadané souřadnice odpovídají 3D modelu znázorněnému na obrázku 12. Zde by mohlo být namítnuto, že v zadání je zmíněn dvoudimenzionální prostor, ale nyní byla uvedena definice 3D modelu. Tato skutečnost je způsobena vlastností OpenFoamu, který je nastaven na operace ve třech dimenzích. Pro převedení modelu do dvou dimenzí se použijí prázdné hranice, které budou uvedeny ve zdrojovém kódu.

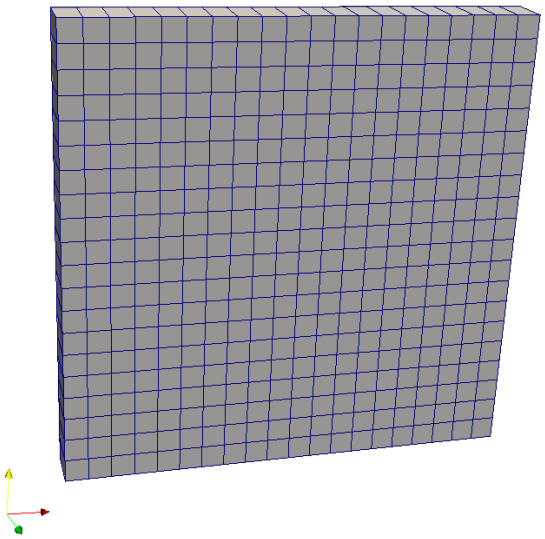


Obrázek 12.: 3D struktura

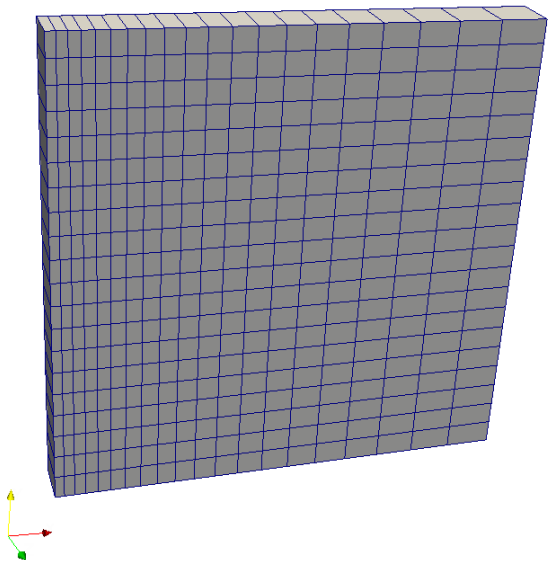
- ř. 33: Definice souřadnicových os *hex (0 1 2 3 4 5 6 7)*. První a druhá pozice s hodnotami nula a jedna nám definuje směr osy *„x“*. Osa *„y“* je definována hodnotami nula a tři (na čtvrté pozici). Osa *„z“* je určena ve směru spojnice bodů nula a čtyři. Tato definice je velice důležitá. Při špatném pořadí neproběhne vytvoření hran objektu. Vytvořilo by se osm samostatných rohů bez spojení. Dále je na tomto řádku uveden zdrojový kód *(10 10 1)* `simpleGrading` *(1 1 1),* který říká, že objekt se bude skládat ze čtvercové sítě deset na deset. Nutno definovat i třetí souřadnici pro tuto síť, která je v tomto případě zanedbatelná. Následující příkaz `simpleGrading` informuje o tom, v jakém poměru velikosti bude síť zobrazena. Zobrazení sítě pro (1 1 1) je uvedeno na obrázku 13. Při změně poměru na hodnoty (5 1 1) se změní síť, viz obrázek 14.
- ř. 36-38: Spojení definovaných rohů, dle směru os určených příkazem *hex.*

- ř. 40-69: Rozčlenění objektu na části. Nejdříve je definována *movingWall*, neboli horní stěna, která bude simulovat proudění. *FixedWalls* definují pevné stacionární stěny, v tomto případě levou, pravou a spodní stěnu. Nyní bude model převeden na dvoudimenzionální. To je provedeno pomocí příkazu `type empty`. Přední a zadní strana krychle je pro kompilátor označena za prázdné. Tím se spodní, levá a pravá stěna převedou z ploch na úsečky, které definují hranice čtverce. Posledním krokem je zadání příkazu `blockMesh`, který vytvoří model zadané struktury. Pokud bude v tuto chvíli spuštěn *paraView*, zobrazí stále 3D model, ale program již považuje tento model za dvoudimenzionální. Výpočty již nebudou probíhat pro osu z.

Následují obrázky, které prezentují zmíněné poměry v konstrukční síti mezi jednotlivými bloky:



Obrázek 13.: SimpleGrading (1 1 1)

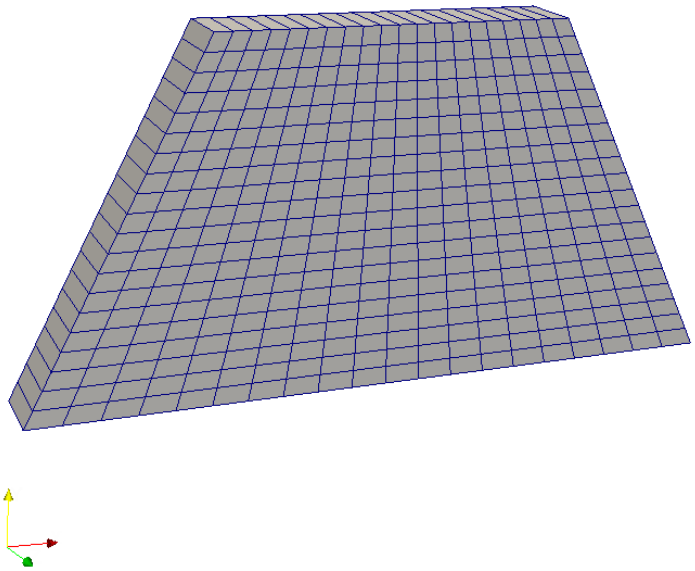


Obrázek 14.: SimpleGrading (5 1 1)

BlockMeshDict může být snadno modifikován. Pro změnu modelu na jiný čtyřúhelník byl použit kód uvedený na řádcích 20-27:

```
20      (0.5 0 0)
21      (1.5 0 0)
22      (1.25 0.5 0)
23      (0.75 0.5 0)
24      (0.5 0 0.1)
25      (1.5 0 0.1)
26      (1.25 0.5 0.1)
27      (0.75 0.5 0.1)
```

Po této změně je nutno opětovného spuštění funkce `blockMesh`, která přepočítá a znovu vygeneruje nový model. Editovaný model je zobrazen níže na obrázku 18:



Obrázek 15.: Modifikovaný model

3.4.3 TransportProperties

Nastavení fyzikálních parametrů pro zadanou simulaci. Funkce icoFoam vyžaduje několik podmínek. První podmínkou je zadání fyzikální veličiny, s kterou se bude simulovat a to s kinematickou viskozitou υ. V níže uvedeném kódu lze vidět tuto definici.

```
8       FoamFile
9       {
10        version     2.0;
```

```
11          format      ascii;
12          class       dictionary;
13          location    "constant";
14          object      transportProperties;
15      }
16
17      nu              [0 2 -1 0 0 0 0] 0.01;
```

Konstanta se zavádí pomocí základních jednotek SI. Čísla na pozicích v závorce prezentují fyzikální veličiny, jejichž pořadí je následovné:

1) hmotnost [kg]
2) délka [m]
3) čas [s]
4) teplota [K]
5) molární hmotnost [kg/mol]
6) elektrický proud [A]
7) svítivost [cd]

Definici viskozity pomocí základních jednotek SI, odpovídají uvedená čísla v závorce. Druhá pozice pro délku s hodnotou čísla dva a třetí pozice pro čas s hodnotou mínus jedna. Z toho vyplývá metr čtvereční za sekundu. Takto byla definována viskozita proudění. Definice všech proměnných na všech místech zdrojového kódu probíhá stejně. Nutno říci že všechny proměnné musí mít stejné jednotky. Např. definuje-li se simulace, která bude vyžadovat hodnotu počáteční energie a energie přidané, musí být obě proměnné definované pro jouly, nebo jinou odpovídající jednotku. Jako poslední při tvorbě nové proměnné je zadání její velikosti. Číslo následují za hranatou závorkou, je právě tato definice, kdy naší proměnné přiřadíme hodnotu 0, 01 [12].

3.4.5 Rychlost – u

Funkci *icoFoam* byla definována viskozita. Nyní je nutno nadefinovat počáteční podmínky. První podmínkou je rychlost proudění pohyblivé stěny [12].

```
8       FoamFile
9       {
10        version     2.0;
11        format      ascii;
12        class       volVectorField;
13        object      U;
14      }
15
16      dimensions      [0 1 -1 0 0 0 0];
17
18      internalField   uniform (0 0 0);
19
20      boundaryField
21      {
22        movingWall
23       {
24           type            fixedValue;
25          value           uniform (1 0 0);
26      }
27
28      fixedWalls
29      {
30          type       fixedValue;
31           value               uniform (0 0 0);
32       }
33
34       frontAndBack
35       {
36           type            empty;
37      }
```

- ř. 16: Nastavení hodnoty pomocí základních jednotek SI, dle postupu, který byl uveden v části *TransporPropeties*.
- ř. 18: V tomto řádku kódu se pro kompilátor nastaví hodnota rychlosti vnitřního pole na nulu. Nyní bude řešitel počítat s rychlostí jen na horní pohyblivé stěně a výpočty s vnitřním polem zanedbá.
- ř. 22-38: Zde jsou velmi důležité názvy stěn jako, *movingWall, fixedWalls* a *frontAndBack.* Nutno upozornit že tyto názvy se musí shodovat s názvy, které byly definovány v části *BlockMeshDict.* Pro pohyblivou stěnu byla přiřazena hodnota rychlosti proudění ve směru osy „*x*“ na 1 m/s. Pro pevné stěny bude tato hodnota nulová. Jelikož se zanedbává přední a zadní stěna, hodnotu pro tyto stěny není definována [12].

3.4.4 Tlak – p

Druhá počáteční podmínka tlak je definována následovně [12].

```
8       FoamFile
9       {
10        version     2.0;
11        format      ascii;
12        class       volScalarField;
13        object      p;
14      }
15
16      dimensions      [0 2 -2 0 0 0 0];
17
18      internalField   uniform 0;
19
20      boundaryField
21      {
22        movingWall
23       {
24           type            zeroGradient;
25       }
26
27       fixedWalls
28       {
29           type            zeroGradient;
30       }
31
32       frontAndBack
33      {
34           type            empty;
35       }
36      }
```

- ř. 16: Definice tlaku v základních SI jednotkách.
- ř. 18: Nastavení hodnoty tlaku na nulu ve vnitřním poli.

- ř. 22-34: Počáteční podmínka tlaku by mohla být vynechána z důvodu nestlačitelného proudění. V simulaci tato veličina nehraje roli. Ale i přes nulovou hodnotu tlaku je nutné tuto definici provést z důvodu požadavků funkce icoFoam, která očekává určité podmínky [2].

3.4.6 ControlDict

V části zdrojového kódu *ControlDict* se definuje, kdo a jak bude provádět simulaci [12].

```
8       FoamFile
9       {
10       version     2.0;
11       format      ascii;
12       class       dictionary;
13       location    "system";
14       object      controlDict;
15
16      application     icoFoam;

17
18      startFrom       startTime;
19
20      startTime       0;
21
22      stopAt          endTime;
23
24      endTime         0.5;
25
26      deltaT          0.005;
27
28      writeControl    timeStep;
29
30      writeInterval   20;
31
32      purgeWrite      0;
33
34       writeFormat     ascii;
35
36      writePrecision  6;
37
38      writeCompression off;
39
40      timeFormat      general;
41
```

```
42      timePrecision   6;
43      runTimeModifiable true;
```

Jako první informaci, která je programu předložena, je název funkce, která bude řešit naší simulaci.

- ř. 16: Název řešitele neboli funkce je *icoFoam*. Tato funkce se používá pro případy nestlačitelného laminárního proudění.
- ř. 18-24: Definice počáteční a konečné podmínky. Simulace začne v čase nula a ukončí se po 0, 5 sekundách. Důležitá podmínka pro funkci kódu je ta, že složka pro počáteční podmínky tlaku a rychlosti musí mít stejný název jako hodnota začátku simulace, neboli nula. Tato podmínka vychází z provázanosti adresářů.
- ř. 26-32: Nastavení ukládání výstupních dat. Typ zápisu a jeho frekvence je určena, jako zápis po dvaceti krocích s rozlišením 0, 005. Zápis bude probíhat každou 0, 1 sekundu. Toto číslo se získá z násobku hodnot *deltaT* a *writeInterval.*
- ř. 32: Zde se volí, do kolika adresářů se data zapíší. Pokud není požadováno více kopií, zvolí se číslo 0.

Zbytek zdrojového kódu nijak neovlivní simulaci. Jedná se zejména o nastavení výchozích dat. Pro začátek je dobré se držet defaultního nastavení, jelikož se jedná o základní a nejrozšířenější možnost zápisu. Při změně tohoto nastavení by mě být kladen důraz na výběr formátu zapisujících se dat, kde jsou k dispozici dvě možnosti. Ascii a Binary formáty. V případě volby ascii formátu, je nutno přidat řádku 38 s příkazem *writePrecision 6.* V opačném případě kompilace neproběhne. Pro formát Binary tento řádek odpadá. Z tohoto jednoduchého příkladu vyplývá, že pro první simulace je doporučeno se držet defaultního nastavení zápisu dat [12].

3.4.7 FvSchemes

Níže uvedený kód je zaměřen na definici konkrétních podmínek a rovnic pro řešení této simulace [12].

```
8       FoamFile
9       {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      location    "system";
14      object      fvSchemes;
15
16      ddtSchemes
17      {
18      default         Euler;
19      }
20
21      gradSchemes
22      {
23      default         Gauss linear;
24      grad(p)         Gauss linear;
25      }
26
27      divSchemes
28      {
29      default         none;
30      div(phi,U)      Gauss linear;
31      }
32
33      laplacianSchemes
34      {
35      default         Gauss linear orthogonal;
36      }
37
38      interpolationSchemes
39      {
40      default         linear;
41      }
42
43      snGradSchemes
44      {
45      default         orthogonal;
46      }
```

Tato část kódu představuje fyzikální model výpočtu, který jsme si uvedli jako informativní příklad. Tyto výpočty nejsou specifikovány pomocí OpenFoam, ale jsou to příklady matematických postupů, o kterých se můžete dozvědět více v extérních. Krátký popis těchto schémat a dalších možností zápisu lze nalézt v odkazu [12], nebo pomocí níže uvedeného příkazu:

```
# foamSearch $FOAM_TUTORIALS "Schemes" fvSchemes
```

3.4.8 FvSolution

Finální částí zdrojového kódu je nadefinování tzv. řešiče [12].

```
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "system";
14     object      fvSolution;
15
16       solvers
17       {
18       p
19       {
20             solver          PCG;
21             preconditioner  DIC;
22             tolerance       1e-06;
23             relTol          0;
24       }
31
32       U
33       {
34             solver          PBiCG;
35             preconditioner  DILU;
36             tolerance       1e-05;
37             relTol          0;
38       }
39       }
40
41       PISO
42       {
43       nCorrectors     2;
44       nNonOrthogonalCorrectors 0;
45       pRefCell        0;
46       pRefValue       0;
47       }
```

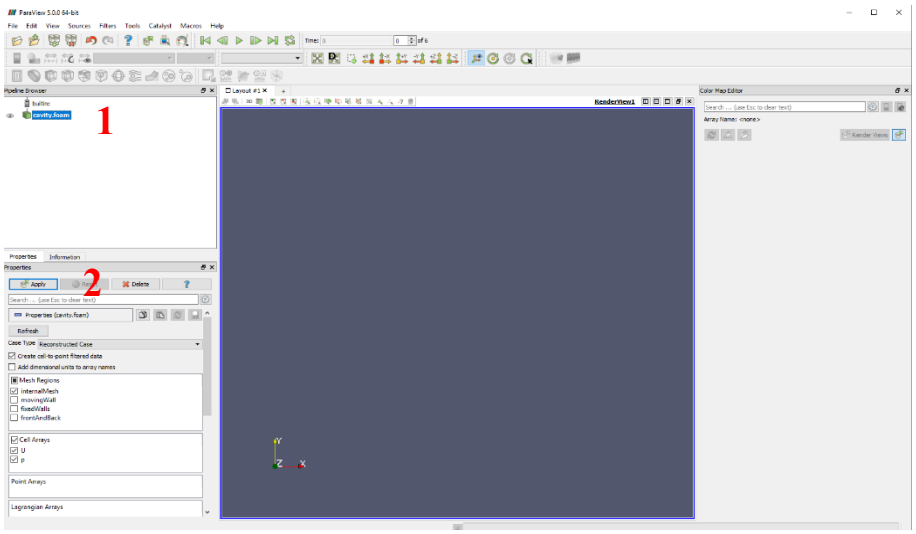
OpenFoam má defaultně na výběr ze tří řešičů, neboli solverů, které provedou výpočty a výsledné simulace. V kódu jsou uvedeny dva z nich.

- ř. 18-24: Nastavení řešiče pro tlak. PCG je řešič pro symetrické matice. Jeho podmínkou je již uvedená symetričnost matic, která se zajistí pomocí příkazu *DIC*. Dle požadavků je možnost nastavení tolerance výpočtů zadáním hodnoty do řádku 22 *tolerance*. Takto byl nastaven první řešič.
- ř. 32-39: Stejně jako u předchozího případu byl nastaven druhý řešič pro rychlost. Jediným rozdílem je, že se nejedná o symetrické matice, ale asymetrické. Této skutečnosti odpovídají příkazy PBiCG a DILU. Opět podrobnější vysvětlení solverů a jejich podmínek je uveden v návodu pod odrážkou *4.5.1 Linear solver* [12] [14].
- ř. 41-47: Jako poslední definuje algoritmus pro řešení naší simulace. PISO algoritmus, je používán pro výpočty v mechanice tekutin, což vyhovuje našemu případu. Více o PISO algoritmu můžeme dočíst v externích zdrojích, nebo v návodu pod odrážkou *4.5.3* [12] [14].

Následně po definici solveru a uložení všech zdrojových kódu, se zavolá funkce *icoFoam*, která provede výpočty a simulaci.

3.4.8 Grafický výstup

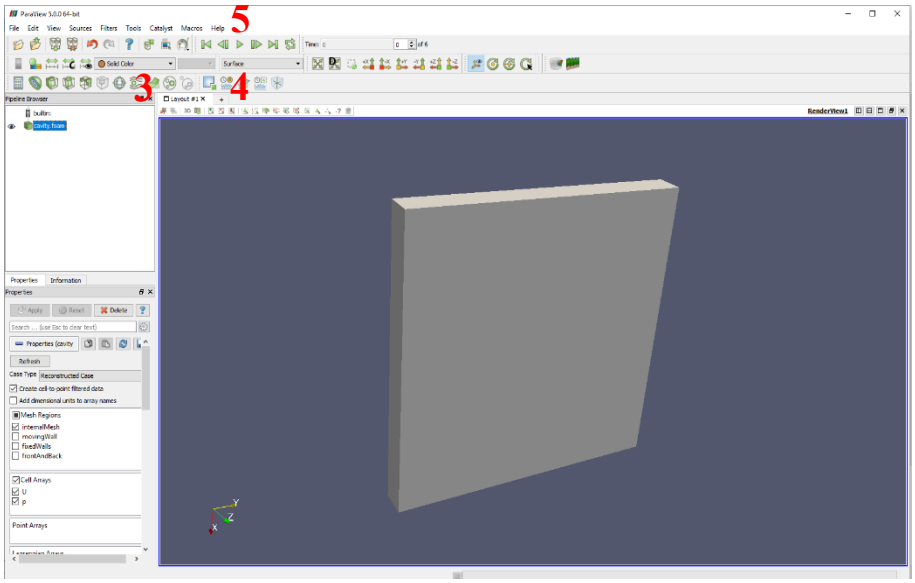
Posledním krokem úlohy je zobrazení výsledku. Grafický výstup se spustí pomocí příkazu *paraFoam*. Na příkazové řádce se zobrazí hláška (Created temporary cavity.foam). Následně dojde ke spuštění programu ParaView, s následující obrazovkou [15]:



Obrázek 16.: Grafické rozhraní ParaView

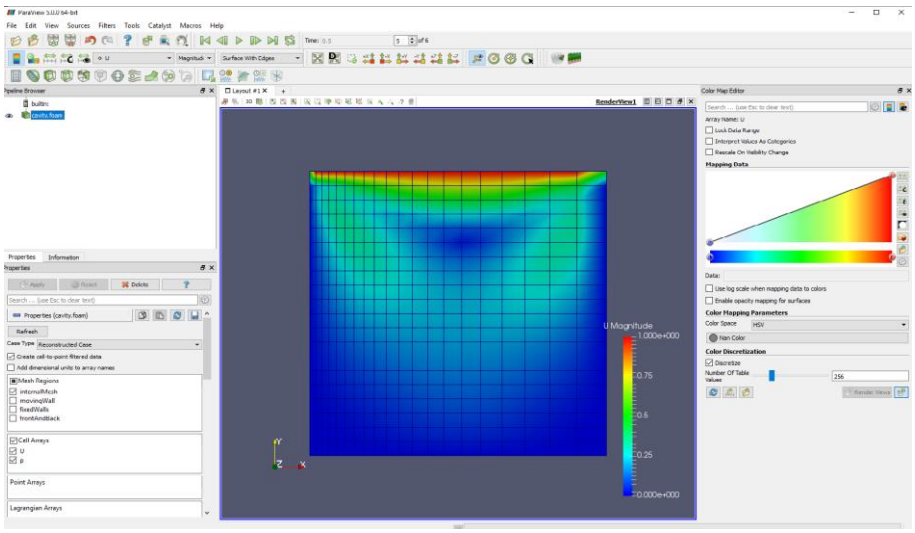
Na obrázku 16 je možno vidět grafické rozhraní ParaView. V oblasti označené číslem jedna, je zobrazen prostor pro načtená data, kde je načten náš příklad cavity. V této oblasti se též zobrazují všechny modifikace, které se s daty provádí. Pro zobrazení dat, se označí cavity.Foam a pomocí okénkem *Apply,* které se nachází v levé části u čísla dvě, se potvrdí volba.

Nyní je možno pracovat s modelem a provádět vizuální simulaci. Tento postup je znázorněn na obrázku 17. Nejprve je nutno nastavit, pro jakou podmínku se bude simulaci provádět. Ve stahovací roletce označené číslem tři *Solid Color* se vybere odrážka U neboli rychlost proudění. Následně se pro přehlednost v roletce u čísla čtyři, vybere rozhraní pro zobrazenou sítě *Surface With Edges.*



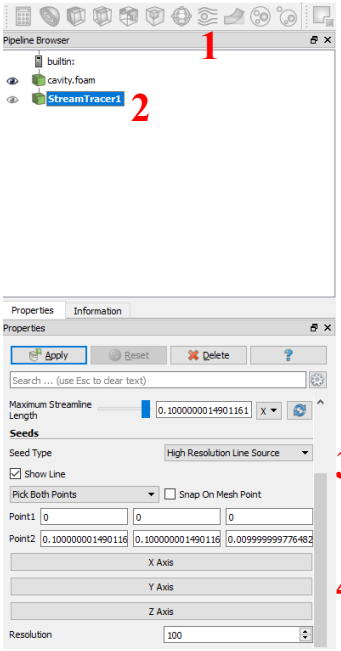
Obrázek 17.: Data přikladu cavity

*S*puštění simulace proběhne pomocí zelené šipky u čísla pět. Výsledný výstup je zobrazen na obrázku 18. Výstup zobrazuje barevné schéma velikosti nestlačitelného proudění. Pro zobrazení legendy se použije symbol označený číslem šest. Pro různé verze ParaView je možno vidět různé barevné kombinace legendy. Pro nastavení těchto barev se v záložce *View* zobrazí *Color Map Editor*, který již nabízí možné úpravy. Tento editor se nachází v pravé části obrázku 18.



Obrázek 18.: Simulace cavity

Poslední úprava, která bude uvedena, je zobrazení proudnic pomocí funkce StreamTracer, která se zvolí následovně [15]:

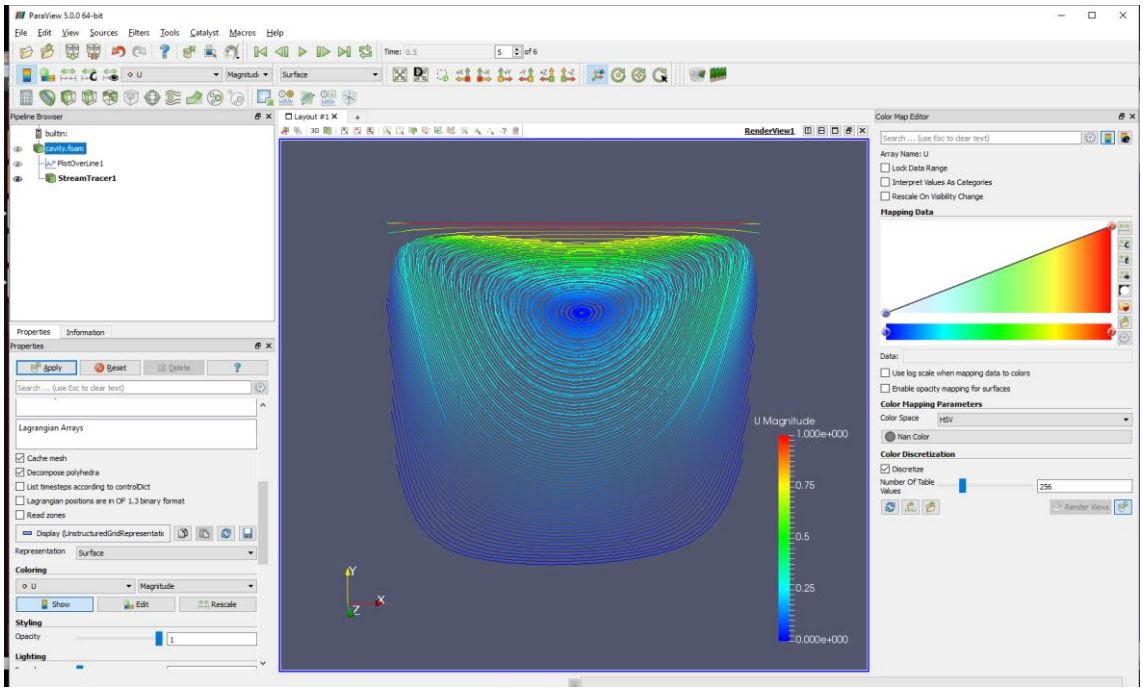


Obrázek 19.: StreamTracer

1) Stisknout ikonu *StreamTracer*.
2) Označit novou strukturu.
3) V roletce *Seed Type* přepnout možnost *High Resolution Line Source*.

4) Vybrat *Y axis* a potvrdit pomocí tlačítka *Apply*.

Výstup ParaView bude přehledný proudnicový model uvedený na obrázku 20:



Obrázek 18.: Proudnicový model

Tento výstup již nabízí přehledný a vypovídající obraz o proudění s větším detailem. Samozřejmě ParaView nabízí širší a modifikovanější grafické úpravy, které již nejsou obsahem této práce [5].

3.5. Shrnutí

Pro přehled je uveden celkový postup pro spuštění simulace cavity [11] [12]:

1) Vstup do kořenové složky *cd cavity*.
2) Spuštění funkce *blockMesh* pro vytvoření geometrie.
3) Spuštění simulace pomocí funkce *isoFoam*.
4) Zobrazení grafického výstupu funkcí *paraFoam*, která spustí ParaView.

Po nastudování vzorového příkladu Cavity, který byl podrobně rozebrán a vysvětlen, by měl být uživatel seznámen s principem a prací v prostředí programu OpenFoam. Měl by chápat syntaxi a strukturu zdrojového kódu. V rámci této práce byl vytvořen modifikovaný příklad, pro ukázku práce s programem OpenFoam. Kvůli rozsahu této práce nebyl příklad uveden, ale bude součástí prezentace. Následně se bude moci příklad využít při výuce předmětu Software pro vědecko- technické výpočty. Pro další rozvoj vědomostí je doporučeno pokročilejších úprav uvedeného přikladu, nebo dalších. Jedny ze vzorových příkladů jsou dostupné na následujícím odkaze [13], kde si můžeme stáhnout konkrétní řešené úlohy z praxe. Jedná se např. o problematiku turbodmychadel, odporu vzduchu křídel letadla a mnohá další.

4. Závěr

Tato bakalářská práce pojednává o softwarech pro vědeckotechnické účely a jejich konkrétním využití ve fyzice. Práce byla koncipována, jako výukový materiál, po jehož nastudování, by měl být uživatel schopen porozumět struktuře a práci se softwarem Octave a OpenFoam.

První část představila program Octave, práci s tímto programem i jeho ovládání a základní příkazy. Následně se zaměřila na konkrétní využití ve fyzice, kde jsou uvedeny příklady zabývající se problematikou Balistické křivky a Matematického kyvadla. Zdrojový kód těchto funkcí je zde uveden a podrobně popsán. Následuje zadání modifikací pro rozšíření znalostí ohledně práce s tímto programem. Zdrojové kódy jsou připojeny k této práci na CD.

Druhá část práce se zabývá programem OpenFoam. Dle doporučení vývojářů se práce zaměří na vzorový příklad, který je podrobně rozebrán. Důraz je kladen na pochopení struktury a syntaxe zdrojového kódu.

Při kompletaci této práce jsem naistaloval a odzkoušel oba zmíněné programy. Naučil jsem se pracovat v jejich prostředí, vytvořil, nebo modifikoval zdrojové kódy k uvedeným úlohám a sepsal stručný návod jak při práci s nimi postupovat a čeho se vyvarovat.

I. Seznam obrázků

II. Seznam použité literatury a zdrojů

Octave:

[1] Daniš, S.: *Základy programování v prostředí Octave a Matlab*. Praha: Univerzita Karlova, 2009, ISBN 9788073780821

[2] Mathworks.com [online]. [vyd. 1994]
URL https://www.mathworks.com/products/matlab.html

[3] Gnuplot.info [online]. [vyd 2016]
URL http://www.gnuplot.info/

[4] Just, M.: *Český průvodce programem Octave*. Červen 2006.
URL http://www.octave.cz/

[5] Gnu.org [online]. [vyd. 1998].
URL https://www.gnu.org/software/octave/

[6] Poláček, J.: *Seriál: Octave.* Leden 2006.
URL http://www.abclinuxu.cz/serialy/octave

[7] Projekt_GNU. Wikipedia.org [online]. [vyd. 2014]
URL https://cs.wikipedia.org/wiki/Projekt_GNU

[8] Free-sw. Gnu.org [online]. [vyd. 2013]
https://www.gnu.org/philosophy/free-sw.html

[9] Richterek,L.: *Dynamické modelování v program GNU OCTAVE.* Září 2007.
URL http://muj.optol.cz/richterek/lib/exe/fetch.php?media=texty:dynmod.pdf

[10] Reichl, J., Všetička, M.: Matematické kyvadlo. 2006.
URL http://fyzika.jreichl.com/main.article/view/205-matematicke-kyvadlo

[11] Reichl, J., Všetička, M.: Vrh šikmý. 2006
URL https://www.gnu.org/philosophy/free-sw.html

[10] Vnejsi_teorie. Balistika.cz [online]. [vyd. 2007]
URL http://www.balistika.cz/vnejsi_teorie.html

OpenFoam:

[11] OpenFoam. Openfoam.org [online]. [vyd. 2011]
URL http://openfoam.org/

[12] User-guide. Cfd.direct [online]. [vyd. 2011]
URL http://cfd.direct/openfoam/user-guide/

[13] OpenFoam-for-windows. Cfdsupport.com [online]. [vyd. 2013]
URL http://www.cfdsupport.com/openfoam-for-windows.html

[14] Gung, T.J., Computational Fluid Dynamics, Cambridge University Press, Cambridge, UK, 2002.

[15] Paraview. Paraview.org [online]. [vyd. 2005]
URL http://www.paraview.org/