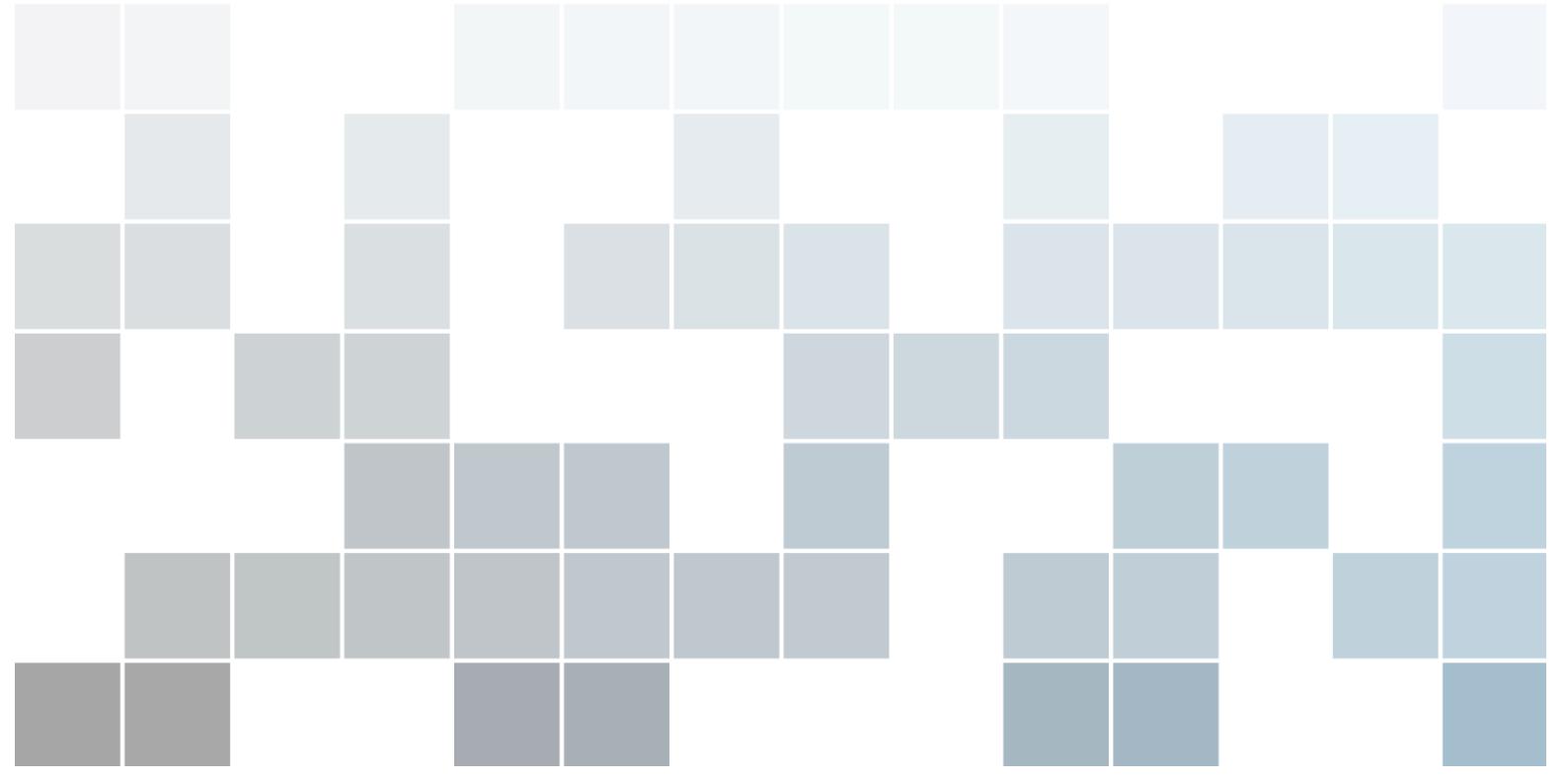


Úvod do fyziky plazmatu interaktivně

Část II: Praktická cvičení

Adam Obrusník, Lenka Zajíčková



Copyright © 2013 Masaryk University

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the “License”). You may not use this file except in compliance with the License. You may obtain a copy of the License at <http://creativecommons.org/licenses/by-nc/3.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, September 2014



Obsah

1	Úvod	5
1.1	Čím tento text je	5
1.2	Čím tento text není	5
1.3	Napojení na přednášku	6
1.3.1	2. a 3. týden: Příprava	6
1.3.2	4. až 6. týden: Částice v polích	6
1.3.3	7. a 8. týden: Zpracování dat	6
1.3.4	9. a 10. týden: Rozdělovací funkce	6
1.3.5	11. až 13. týden: závěrečná úloha	6
2	Základy Matlabu	7
2.1	Instalace Matlabu na MUNI	7
2.1.1	Licence	7
2.1.2	Získání instalacních souborů	7
2.1.3	Instalace	8
2.1.4	Spuštění Matlabu	9
2.2	Základy Matlabu	9
2.2.1	Proměnné a indexování	10
2.2.2	M-soubory	11
2.2.3	Kam dál	13
3	Pohyb částic v el. a mag. polích	15
3.1	Částice v konstantních elektrických a magnetických polích	15
3.1.1	Související rovnice	15
3.1.2	Řešení soustavy ODR v Matlabu	16
3.1.3	Implementace	17
3.1.4	Cvičení	19

3.2	Van Allenovy radiační pásy	21
3.2.1	Související rovnice	21
3.2.2	Implementace	22
4	Zpracování dat	29
4.1	Úvod	29
4.2	Smyčky a podmínky v Matlabu	29
4.2.1	Smyčka for	30
4.2.2	Podmínky if ... elif ... else	32
5	Interakce částic v plazmatu	35
5.1	Maxwell-Boltzmannovo rozdělení	35
5.1.1	Střední rychlosť	35
5.1.2	Tvar Maxwell-Boltzmannova rozdělení	38
5.2	Časový vývoj rozdělovací funkce	38
5.3	Rychlostní koeficienty	42
6	Rovnováha částic v plazmatu	45
6.1	Formulace problému	45
6.1.1	Srážkový člen	45
6.1.2	Reakční schéma	47
6.2	Implementace	48
6.3	Parametrická studie	52
6.4	Závěr	53
	Bibliography	55
	Books	55
	Articles	55
	Online resources	55

Čím tento text je

Čím tento text není

Napojení na přednášku

2. a 3. týden: Příprava

4. až 6. týden: Částice v polích

7. a 8. týden: Zpracování dat

9. a 10. týden: Rozdělovací funkce

11. až 13. týden: závěrečná úloha

1. Úvod

1.1 Čím tento text je

Tento text je **doplňujícím materiélem** ke cvičením z předmětu *F5170: Úvod do fyziky plazmatu*. Zatímco přednáška obsahuje mnoho teoretických odvození a úvah, cílem tohoto studijního materiálu je zejména zprostředkovat intuitivní zkušenosť s fyzikou plazmatu pro studenty, kteří se s plazmatem setkávají poprvé.

V průběhu textu budete jako studenti postaveni před řadu **úloh, z nichž většina musí být vyřešena s pomocí počítače**. Mimo jiné se naučíte jak řešit soustavy obyčejných diferenciálních rovnic, jak hromadně zpracovat data nebo jak numericky integrovat. Většinu schopností a dovedností, které se v tomto kurzu naučíte upotřebíte nejen ve fyzice plazmatu, ale také v jiných oblastech fyziky.

Praktická cvičení jsou navržena tak aby studentům pomohly vizualizovat a intuitivně pochopit těžce představitelné jevy probíhající v plazmatu. V textu je zohledněno, že většina studentů nemá rozsáhlé předchozí zkušenosť s programováním. Z tohoto důvodu je veškerý zdrojový kód bohatě komentován a dokumentován a **studenti by měli** být schopni **pochopit základy programování v Matlabu z praktických příkladů** v tomto textu.

1.2 Čím tento text není

V první řadě musí být řečeno, že tento text si neklade za cíl být vyčerpávající učebnicí programování v Matlabu. Existuje nespočet kvalitních knih a manuálů týkajících se Matlabu nebo jeho balíčků. Ty nejrelevantnější a nejsnáze dostupné jsou vyjmenovány v oddílu 2.2.3. Očekávejte, že textu může místo být příliš stručný, krkolomně napsaný nebo může mít nečekané logické členění. Důvodem je, že tento text není navržen tak, aby byl použitelný sám o sobě a nejlépe bude fungovat v kombinaci se cvičeními k předmětu *F5170: Úvod do fyziky plazmatu*.

Tento text také není vhodným zdrojem, který by jeho uživatel měl citovat v jakýchkoli vědeckých publikacích nebo diplomových pracích. Vždy je nejlepší odkázat se přímo na knihy nebo články, ze kterých tento text vychází. Ačkoliv se může zdát, že hledat původní reference jen abyste ověřili jeden vzorec nebo obrázek, je zbytečně složité, jedná se o schopnost, která vám usnadní vaši budoucí vědeckou práci.

Dále je nutno zmínit, že tento text není zaměřen na matematické principy numerických metod. Řešíce diferenciálních rovnic, které jsou v tomto textu použity, jsou provádzovány za

černé skříňky a princip jejich fungování není dále analyzován. Autoři se také vyhnuli transformaci rovnic do bezrozměrného tvaru, protože používání smysluplných fyzikálních jednotek vám umožní kromě kvalitativního náhledu také získání kvantitativní představy o důležitých veličinách ve fyzice plazmatu.

1.3 Napojení na přednášku

Přednáška je rozložena do 13 týdnů podzimního semestru na Masarykově Univerzitě. Tento text odráží obsah přednášky a zhruba jednou za dva týdny byste měli pokročit o jednu kapitolu. V prvním týdnu semestru cvičení většinou neprobíhá.

1.3.1 2. a 3. týden: Příprava

Vzhledem k tomu, že přednáška vyžaduje oproti cvičení určitý náskok, jsou celé první dva týdny vyhrazeny tomu, abyste si nainstalovali potřebný software a naučili se s ním zacházet. Jak uživatelé systému Widnows tak uživatelé Linuxových systémů si nainstalují univerzitní licenci Matlabu (vizte kapitolu 2). Studenti by si také měli nastudovat druhou část této kapitoly, aby získali základní představu o logice a syntaxi Matlabu.

1.3.2 4. až 6. týden: Částice v polích

V týdnech 4 až 6 byste měli pochopit všechny Matlabové programy představené v kapitole 3 a v šestém týdnu byste měli dokončit všechna relevantní cvičení. Cvičné programy se týkají pohybu částic v elektrických a magnetických polích a měly by vám pomoci pochopit povahu různých driftů v plazmatu.

1.3.3 7. a 8. týden: Zpracování dat

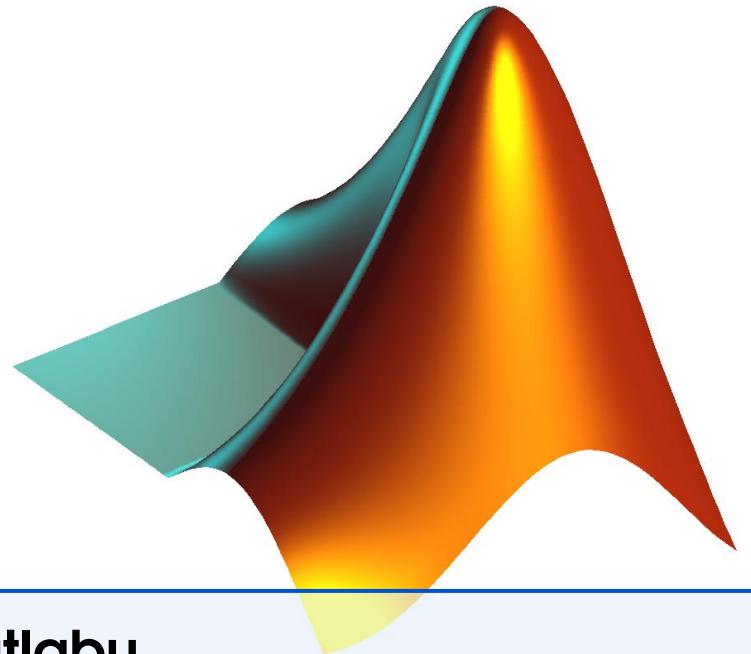
Na příkladech z fyziky plazmatu se naučíte se základy hromadného zpracování dat. Konkrétně se jedná o interpolaci a integraci diskrétních dat a také hromadný převod dat do jiných jednotek. Ačkoliv se budou úlohy týkat dat souvisejících s fyzikou plazmatu, budou se vám získané znalosti jistě hodit i v jiných oblastech fyziky.

1.3.4 9. a 10. týden: Rozdělovací funkce

Tato krátká kapitola by Vám měla pomoci pochopit rozdělovací funkce. Za tímto účelem vám bude poskytnuto několik programů, které jsou schopny vizualizovat časový vývoj rozdělovacích funkcí. Kromě toho budete analyzovat tvar Maxwell-Boltzmannova rozdělení za různých podmínek.

1.3.5 11. až 13. týden: závěrečná úloha

V závěrečné úloze využijete schopnosti, které jste získali v předchozích kapitolách. Budete muset vytvořit případně dotvořit program, který řeší rovnováhu částic v laboratorním argonovém plazmatu. Budete pozorovat jak se s měnícím tlakem či teplotou elektronů nebo těžkých částic mění složení plazmatu.



2. Základy Matlabu

Matlab (původně Matrix Laboratory) je komerční software a skriptovací jazyk vyvíjený společností MathWorks. Na rozdíl od jiných tradičních programovacích jazyků umožňuje Matlab intuitivní práci s maticemi a vektory. Uživatelé si také mohou definovat své vlastní funkce s takřka libovolným vstupem a výstupem.

2.1 Instalace Matlabu na MUNI

2.1.1 Licence

K roku 2014 vlastní Masarykova univerzita licenci nejnovějšího Matlabu se současným přístupem pro nejméně 250 uživatelů. **Aby instalace fungovala a Matlab šel spustit, musí být uživatel připojen k síti VPN Masarykovy univerzity nebo se nacházet v prostorách univerzity.** Licence je omezena na výzkumné nekomerční účely.

2.1.2 Získání instalacních souborů

Instalační soubory Matlab 2014b jsou k dispozici na intranetu Masarykovy univerzity. Instalační soubory mají zhruba 7.5 GB, ale v rámci MU (například z knihovny) by mělo stahování být extrémně rychlé. Od verze 2013 je Matlab k dispozici pouze pro 64bitové operační systémy. Pokud máte starší počítač vybavený 32bitovým procesorem, kontaktujte cvičícího.

Návod 2.1.1 — Stažení souborů. Pro stažení souborů postupujte následovně:

1. V prohlížeči otevřete inet.muni.cz
2. Přihlaste se pomocí *UČO* a primárního hesla
3. Jděte do *Software → Nabídka softwaru*
4. V seznamu najděte Matlab a klikněte *Získat*
5. Po odsouhlasení licenční smlouvy si do počítače uložte
 - soubor *license.dat*,
 - instalační obraz, *R201xx_Windows.iso* pro Windows nebo *R201xx_UNIX.iso* pro Linuxové systémy.
 - *Autorizační kód* – uložte si jej například do textového souboru, instalátor jej bude vyžadovat.

2.1.3 Instalace

Po stažení ISO obrazů je musíte připojít jako virtuální mechaniku a spustit instalaci. Alternativně můžete vypálit obraz ISO na DVD a nainstalovat Matlab z něj.

Návod 2.1.2 — Připojení ISO ve Windows. Pravděpodobně nejčastěji používaný software na připojování ISO souborů (tedy vytvoření virtuální mechaniky, která bude obsahovat data uložená v ISO souboru) je pravděpodobně [Daemon Tools Lite](#).

R Přestože je nejlepším softwarem pro připojování virtuálních mechanik, nutí Daemon tools Lite uživatele do instalace nechtěného software. Pokud se chcete vyhnout instalaci nechtěného panelu v prohlížeči, čtěte pozorně instrukce během instalace a ve správnou chvíli zvolte, že nechcete instalovat software třetích stran.

Po instalaci Daemon Tools připojte DVD s Matlabem.

1. Klikněte pravým tlačítkem na ikonu Daemon Tools v oznamovací oblasti.
2. Zvolte *Virtual DVD → Device 0: → Mount Image*.
3. Najděte na svém disku ISO obraz a potvrďte.
4. Nyní uvidíte DVD s Matlabem v *Počítač*

Návod 2.1.3 — Připojení ISO v Linu xu. Většina moderních linuxových distribucí obsahuje zabudované funkce pro připojování ISO souborů. Většinou stačí kliknout pravým tlačítkem na ISO soubor a kliknout na "Mount" či "Připojit". Pokud tuto funkci ve vaší distribuci nemůžete najít, můžete použít program [AcetoneISO](#), který by se měl nacházet v repozitářích.

Pokud dáváte přednost příkazové řádce, nemusíte instalovat nic a stačí do terminálu napsat následující:

```
mount -o exec R201xx_UNIX.iso /mnt/disk
```

1

kde /mnt/disk je adresář, kam chcete ISO soubor připojit.

Instalátor Matlabu spusťte ve Windows přes soubor `setup.exe` a v Linuxu pomocí binárního souboru `./install`. Oba tyto soubory jsou uloženy v kořenové složce instalačního DVD. Pokud se Vás instalátor zeptá, zda si přejete Matlab aktivovat, klikněte na *Yes*.

R Na Linuxu se doporučuje instalovat Matlab jako superuživatel. V opačném případě se nevytvoří odkaz v `/usr/local/bin` a nebude možné Matlab spouštět pomocí příkazu `matlab` z příkazové řádky.

Samotná instalace Matlabu je přímočará a uživatelský přívětivá. Nejprve se Vás zeptá na *Autorizační kód* následně na umístění, kam se má Matlab instalovat a nakonec Vás požádá o licenční soubor. Těsně předtím, než se na disk začnou soubory kopírovat budete dotázáni na *instalaci toolboxů*. Toolboxy jsou dodatečné knihovny funkcí, které rozšiřují funkčnost Matlabu. Pokud potřebujete ušetřit místo na disku, nemusíte instalovat toolboxy z následujících kategorií:

- Test and measurement,
- Computational finance,
- Computational Biology,
- Code generation.

2.1.4 Spuštění Matlabu

Uživatelé Windows mohou Matlab spustit přes zástupce na ploše, uživatelé Linuxu pomocí příkazu `matlab` (pokud jste software instalovali jako superuživatel) nebo z jeho instalačního adresáře.

```
/path/to/Matlab/R20xx/bin/matlab
```

1

Po spuštění Matlabu se již ukáže hlavní okno, které je už identické na všech systémech.

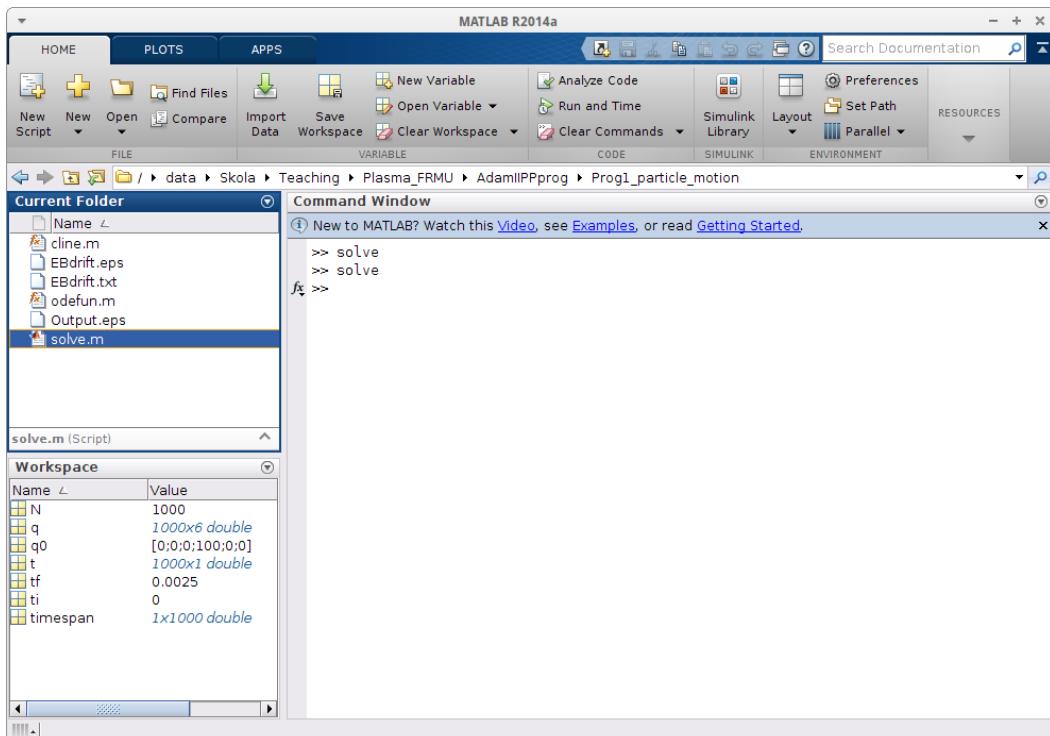


Figure 2.1: Hlavní okno Matlabu

Hlavní okno má tři důležitá podokna:

- **Current folder** zobrazuje obsah složky, ve které právě pracujete. To jsou zejména všechny matlabové skripty, které můžete spustit.
- **Workspace** obsahuje všechny proměnné (numerické hodnoty nebo matice), které jsou definovány. Dvojitým kliknutím na proměnnou ze zobrazí její hodnota a můžete ji editovat.
- **Command window** je nejzásadnější částí. Zde můžete definovat nové proměnné, volat funkce nebo spouštět Matlabové skripty, o kterých se toho více dozvíte v dalším oddílu.

2.2 Základy Matlabu

Tento oddíl stručně vysvětuje základy syntaxe a logiky Matlabu. Vysvětlení nejsou velmi vyčerpávající a rigorózní, protože se předpokládá, že si poznatky osvojíte na praktických příkladech níže. Na konci tohoto oddílu byste měli vědět zejména odpovědi na následující otázky:

- Co je *proměnná* v Matlabu a jak ji definovat.
- Jak přistupovat k prvkům vektorů a matic, jak vybírat submatice.
- Jaký je rozdíl mezi *soubory skriptů* a *soubory funkcí*.
- Co je v Matlabu *funkce* a jak ji volat.

Pro některé z Vás budou informace v tomto oddíle triviální, ale tento text myslí i ty, kteří se zatím neměli možnost setkat s programováním.

2.2.1 Proměnné a indexování

Práce s proměnnými v Matlabu je stejně jednoduchá jako na papíře, jen je důležité si uvědomit, že Matlab **provádí numerické operace, nikoliv analytické**. Co to znamená v praxi? Zkuste si definovat proměnnou tím, že do *Command window* napišete:

```
v = 10
```

1

a předpokládejme, že tato proměnná vyjadřuje rychlosť nějakého objektu. Nyní definujeme hmotnost našeho imaginárního objektu v kilogramech

```
m = 3
```

1

a jeho kinetickou energii tedy spočítáme jako

```
Ek = 0.5*m*v^2
```

1

Pokud nyní předefinujete rychlosť na jinou hodnotu a podíváte se do podokna *Workspace*, uvidíte, že ačkoliv jsme změnili rychlosť, hodnota proměnné *Ek* odpovídající kinetické energii se nezměnila. To je proto, že tato proměnná **obsahuje pouze numerickou hodnotu** a neobsahuje žádný odkaz na proměnné, pomocí kterých byla spočtena. Pokud chcete získat aktuální hodnotu *Ek*, musíte spustit odpovídající příkaz.

Pracovat s maticovými a vektorovými proměnnými je v Matlabu také jednoduché. Předpokládejme, že chceme rotovat vektor v 3D kartézském prostoru v *xy* rovině. Nejprve tedy definujeme vektor, který budeme rotovat:

```
r = [10, 20, 30]
```

1

a následně matici otočení pomocí vestavěných funkcí *sin()* ad *cos()*

```
M = [cos(pi/3), -sin(pi/3), 0; sin(pi/3), cos(pi/3), 0; 0, 0, 1]
```

1

Jak vidíte jsou matice definovány řádek po řádku. Prvky ve sloupcích jsou odděleny čárkami (,) a řádky jsou odděleny středníky (;). Vektorem se potom v podstatě rozumí matice $1 \times n$ nebo $n \times 1$. Nicméně, pokud se pokusíte *M* a *r* vynásobit a napišete

```
rrot = M*r
```

1

dostanete chybu, *inner dimensions do not agree*. To je způsobeno tím, že *r* je řádkový vektor a matici 3×3 nelze vynásobit vektorem 1×3 . Pro násobení tedy musíte napsat příkaz

```
rrot = M*r'
```

1

kde apostrof značí transpozici vektoru *r*.



Jak jste si všimli, zobrazí se při definování proměnné její hodnota v okně příkazů (*Command window*). Abyste tomu v budoucnu předešli, ukončujte každý příkaz středníkem (;).

Standardní operace, které můžete provádět na vektorech a maticích jsou sčítání (+), odčítání (-), násobení (*), dělení (/) a umocnění (^). Kromě toho existují v Matlabu **operace po prvcích**.

Například násobení po prvcích ($\cdot \ast$) dvou vektorů rozměru 1×3 funguje následujícím způsobem

$$(1, 4, 5) \cdot \ast (2, 4, 6) = (1 \ast 2, 4 \ast 4, 5 \ast 6) = (2, 16, 30). \quad (2.1)$$

Kromě násobení po prvcích existuje také dělení po prvcích ($\cdot /$) umocňování po prvcích ($\cdot ^\wedge$).

Dále je důležité vědět, jak **přistupovat k jednotlivým prvkům matic a vektorů** nebo jak vytvořit sub-matice a sub-vektory. V případě vektorů můžete provádět následující operace

```
a = [1, 2, 3, 4, 5, 6, 7];  
1  
b = a(4) % ziska čtvrtý prvek "a" = 4  
2  
b = a(1:3) % subvektor od prvku 1 do 3 = [1, 2, 3]  
3  
b = a(4:end) % subvektor od prvku 4 do konce = [4, 5, 6, 7]  
4
```

V případě dvourozměrných matic vybíráte prvky podobně, potřebujete nicméně dva indexy kdy první udává řádek a druhý udává sloupec

```
a = [1, 2, 3; 4, 5, 6; 7, 8, 9];  
1  
b = a(1,2) % prvek na souradnicích 1x2 = 2  
2  
b = a(:,2) % druhý sloupec = [2, 5, 8]  
3  
b = a(3,:); % třetí řádek = [7, 8, 9]  
4  
b = a(1:2,1:2) % ziska submatice = [1, 2; 4, 5]  
5
```

2.2.2 M-soubory

Definování proměnných a práce s nimi v okně příkazů je sice užitečné, ale pokud by Matlab uměl jen to, byl by jen trochu lepší kalkulačkou. Typicky se Matlab (a jiné podobné skriptovací jazyky) používají pro automatizaci repetitivních nebo náročných úkolů, obsahujících určitou posloupnost operací. V Matlabu se posloupnosti příkazů ukládají do dvou typů souborů, do **souborů skriptů** a **souborů funkcí**. Trochu matoucí může být fakt, že **oba typy souborů mají příponu .m.**

Soubory skriptů

Soubor skriptu obsahuje posloupnost Matlabových příkazů, jeden příkaz na každém řádku. Tyto příkazy se postupně spouštějí, jeden za druhým. Ukážeme si použití skriptu na hypotetické situaci, kdy po delší dobu měříte určitou veličinu a chcete každý den **generovat vždy aktuální graf se stejnou úpravou**. V tomto případě je pohodlné vytvořit si skript

Příklad skriptu v Matlabu

```
day = [1, 2, 3, 4, 5, 7, 8, 10, 12, 15]; % days  
1  
count = [450, 363, 291, 220, 180, 150, 120, 100, 90, 80  
2  
]; % pocet bakterii  
mc = max(count); % najde maximum vektoru count  
3  
norm_count = count/mc; % provede normalizaci  
4  
  
% vykresli day na osy x a count na osu y  
5  
plot(day, norm_count, 'rx-');  
6  
xlabel('Days'); % nastavi popisek osy x  
7  
ylabel('Normalized cell count [CFU]'); % nastavi  
8  
popisek osy y  
title('Plasma sterilisation experiment') % nastavi  
9  
nadpis obrazku  
10
```

```
11
print -dpng -r72 'experiment.png'; % ulozi obrazek do
12
png
```

Na prvních dvou řádcích jsou uložena měřená data. Proměnná `day` obsahuje jednotlivé dny měření zatímco proměnná `count` obsahuje měřené hodnoty v jednotlivé dny. Na řádcích 3 a 4 jsou data normalizována a od řádku 7 dále jsou vykreslena. V naší modelové situaci by Vám takový skript jistě ušetřil čas.

R Zdrojové kódy výše obsahují komentáře. Komentáře můžete a měli byste používat k anotování veškerého vašeho kódu, abyste si byli schopni vybavit, co který příkaz dělá, i po delší době. Znak procenta v Matlabu tedy dává najevo, že cokoliv za ním (až do konce řádku) není součástí příkazu a Matlab to bude ignorovat.

Pokud chcete **spustit skript**, změňte **Current folder** na adresář, ve kterém je skript uložen a **napíšete název skriptu** (bez přípony .m) **do okna příkazů**.

Cvičení 2.1 Napište skript, který vykreslí následující měření koncentrace ozonu v logaritmické škále. Dějte pozor na to, že funkce `log()` počítá přirozený logaritmus a pro spočítání logaritmu se základem 10 je třeba použít funkci `log10()`

pologa [mm]	0.0	0.1	0.2	0.3	0.5	0.6
konz. O ₃ [m ⁻³]	$1.4 \cdot 10^{20}$	$1.0 \cdot 10^{20}$	$5.2 \cdot 10^{19}$	$1.6 \cdot 10^{19}$	$4.5 \cdot 10^{18}$	$9.1 \cdot 10^{17}$

Soubory funkcí

M-soubory obsahující funkce jsou poněkud odlišné od skriptů. V první řadě musí být zmíněno, že funkce už není jen prostá posloupnost příkazů ale je to objekt, který má vždy N vstupů a M výstupů, podobně jako funkce matematická. Pro příklad, funkce, která dokáže spočítat Larmorův poloměr a cyklotronovou frekvenci pro elektron by vypadala následovně:

Matlab function example: gyro.m

```
function [r, omega] = gyro(B, vp)
1
q = -1.602e-19; % elementarni naboj [C]
2
m = 9.109e-31; % hmotnost [kg]
3
4
r = m*vp/(abs(q)*B); % Larmoruv polomer
5
omega = q*B/m; % Cyklotronova frekvence
6
end % konec funkce
7
```

Chování této funkce je dáno na prvním řádku. Proměnné `r` a `omega` jsou výstupní proměnné zatímco `B` a `vp` jsou vstupní proměnné. **Název funkce na prvním řádku** (`gyro`) **se musí shodovat s názvem souboru**.

Tuto funkci můžete následně volat buď z **okna příkazů** nebo z jiného matlabového skriptu umístěného ve stejném adresáři. Pro vyzkoušení vyzkoušejte napsat

```
[r0, w0] = gyro(1e-4, 1e3)
```

do Vašeho **okna příkazů**. Ve **Workspace** by se měly objevit dvě nové proměnné, `r0` a `w0`.

Cvičení 2.2 Napište funkci, která bude mít na vstupu koncentraci elektronů v m⁻³ a teplotu

elektronů v eV. Na výstupu byla měla být Debyeova délka a plazmová frekvence. Potřebné vzorce můžete najít například v [Bit04]. ■

2.2.3 Kam dál

V tomto oddíle byly vysvětleny jen ty nejdůležitější základy programování v Matlabu. Pro detailnější interaktivní přehled syntaxe doporučují autoři nahlédnout na www.mathworks.com/help/matlab/getting-started-with-matlab.html.

Stejně jako u ostatních počítačových dovedností platí, že Google je Váš kamarád. Pokud budete hledat odpověď na nějaký problém, všímajte si zejména odkazů na stackexchange.com. Vzhledem k rozšířenosti Matlabu je skoro jisté, že nejste první, kdo narazil na určitý problém a odpověď na Vás čeká na jednom z fór.

Pokud budete chtít používat Matlab i nad rámec tohoto kurzu, například pro vlastní výzkum-nou činnost, je vhodné seznámit se i s portálem MatlabCentral www.mathworks.com/matlabcentral. Tento portál obsahuje detailní dokumentaci Matlabových funkcí a většina z nich je doplněna příklady použití. Dále je na MatlabCentral dostupné diskuzní fórum plné zkušených uživatelů a repositář užitečných funkcí (např. pokročilé fitování, jednoduší vykreslování dat, atp.).

Částice v konstantních elektrických a magnetických polích

Související rovnice

Řešení soustavy ODR v Matlabu

Implementace

Cvičení

Van Allenovy radiační pásy

Související rovnice

Implementace

3. Pohyb částic v el. a mag. polích

3.1 Částice v konstantních elektrických a magnetických polích

První úloha, kterou si implementujeme, se týká pohybu nabité částice v elektrickém a magnetickém poli. Tato úloha má dobře známé analytické řešení, ale my ji zde budeme řešit numericky právě proto, abychom demonstrovali, jak se liší analytické řešení obyčejných diferenciálních rovnic (ODR) na papíře od toho numerického. První program, se kterým budeme pracovat bude bohatě komentován a vysvětlen a je tedy důležité, abyste pochopili, co dělá každý rádek kódu.

Pohyb částic ve složitých a často neuniformních či časově závislých elektrických a magnetických polích hraje důležitou roli ve fyzice plazmatu. Aplikace, ve kterých je třeba dobré znát pohyb částic jsou například:

- **Hmotnostní spektrometry**, ve kterých jsou částice filtrovány podle poměru hmotnosti/náboj.
- **Fúzní reaktory** využívající magnetické pole pro udržení plazmatu [aa]
- **Elektronové mikroskopy**, kde je elektronový svazek směrován elektrickým polem [FEI10]
- **Hallův motor** používaný pro vesmírný pohon (vizte obrázek 3.1)
- a mnoho dalších...

3.1.1 Související rovnice

Program vyvinutý v tomto oddíle řeší pohybové rovnice ve trojrozměrných kartézských souřadnicích pro částici s určitou hmotností a nábojem na kterou působí Lorentzova síla.

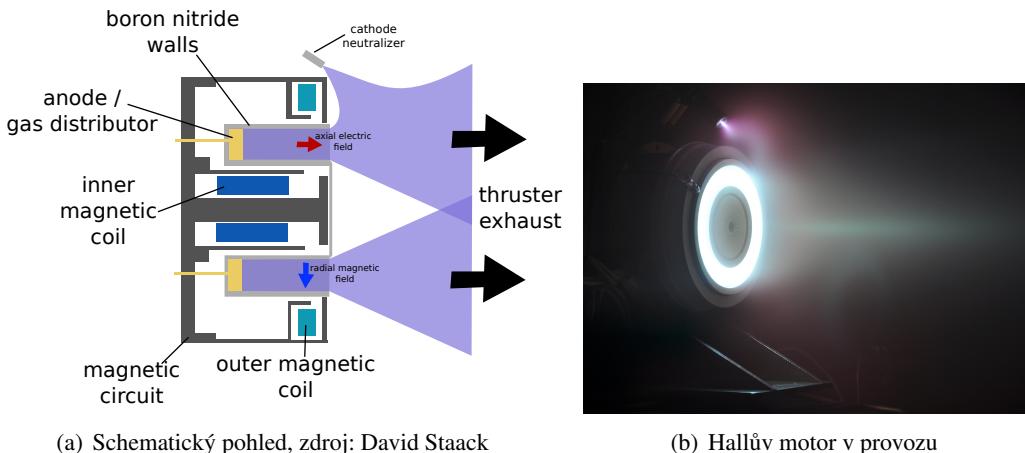
$$\mathbf{F}_L = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) = q(\mathbf{E} + \dot{\mathbf{r}} \times \mathbf{B}) \quad (3.1)$$

kde q je náboj částice, \mathbf{E} je elektrické pole, \mathbf{B} je magnetické pole, \mathbf{v} je rychlosť částice a \mathbf{r} je její poloha. Pohybová rovnice tedy nabývá tvaru

$$\ddot{\mathbf{r}} = \frac{q}{m} (\mathbf{E} + \dot{\mathbf{r}} \times \mathbf{B}) \quad (3.2)$$

kde m je hmotnost částice. Pohybová rovnice v tomto tvaru je užitečná při mnohých teoretických úvahách a při hledání analytického či přibližného analytického řešení této úlohy. Pokud ale chceme tu rovnici řešit numericky, budeme se na ni muset podívat z trochu jiného úhlu.

V první řadě musíme zdůraznit, že Matlab (stejně jako většina dalších numerických knihoven) neobsahuje funkce pro přímé řešení ODR druhého řádu, obsahuje ale velmi pokročilé



(a) Schematický pohled, zdroj: David Staack

(b) Hallův motor v provozu

Figure 3.1: Hallovy motory využívají elektrické a magnetické pole k urychlení iontů ven z plazmatu. V roce 2014 se již Hallovy motory běžně využívají pro geostacionární družice i extraterestriální mise [GK08, page 15]

a stabilní funkce pro numerické řešení ODR prvního řádu i jejich soustav. Použijeme-li standardní značení,

$$\mathbf{r} = (x, y, z), \quad (3.3)$$

$$\dot{\mathbf{r}} = (v_x, v_y, v_z), \quad (3.4)$$

$$\dot{\mathbf{B}} = (B_x, B_y, B_z), \quad (3.5)$$

$$\dot{\mathbf{E}} = (E_x, E_y, E_z) \quad (3.6)$$

je poměrně jasné, že rovnice (3.2) je ekvivalentní následujícím šesti rovnicím.

$$\dot{x} = v_x, \quad (3.7)$$

$$\dot{y} = v_y, \quad (3.8)$$

$$\dot{z} = v_z, \quad (3.9)$$

$$\dot{v}_x = \frac{q}{m} (E_x + v_y B_z - v_z B_y), \quad (3.10)$$

$$\dot{v}_y = \frac{q}{m} (E_y + v_z B_x - v_x B_z), \quad (3.11)$$

$$\dot{v}_z = \frac{q}{m} (E_z + v_x B_y - v_y B_x). \quad (3.12)$$

Jak vidíte, jednoduše jsme transformovali tři ODR druhého řádu popsané rovnicí (3.2) do šesti ODR prvního řádu.

R Transformování N diferenciálních rovnic druhého řádu do $2N$ diferenciálních rovnic prvního řádu je ve fyzice poměrně časté. Většina z Vás se s tím jistě setkala v kurzech teoretické mechaniky, kde se mimo jiné provádí přechod mezi Lagrangeovským a Hamiltonovským formalismem. Z transformace provedené výše je zřejmé, že formalismem vhodnějším pro numerickou analýzu je ten Hamiltonovský.

3.1.2 Řešení soustavy ODR v Matlabu

matlabová funkce, kterou budeme používat pro řešení soustavy ODR je pojmenovaná `ode45`. Tato funkce umí řešit diferenciální rovnice prvního řádu zadáné ve tvaru

$$\dot{\mathbf{q}} = f(t, \mathbf{q}). \quad (3.13)$$

kde t je nezávislá proměnná (v našem případě čas) a $\mathbf{q} = \mathbf{q}(t)$ je vektor závislých proměnných. Nyní by už mělo být zřejmé, proč jsme přeformulovali pohybovou rovnici do tvaru popsaného rovnicemi (3.7) až (3.12). Pokud tyto rovnice porovnáte s rovnicí (3.13) vidíte, že v našem případě je

$$\mathbf{q} = (x, y, z, v_x, v_y, v_z). \quad (3.14)$$

V Matlabu potom funkci `ode45` voláte následujícím způsobem

```
[t, q] = solver(@f, tspan, q0)
```

1

Prvním argumentem (vstupní proměnnou) je odkaz na funkci $f(t, \mathbf{q})$, která dává hodnoty pravých stran rovnic v naší soustavě. Druhým argumentem je interval nezávislé proměnné (u nás času), na kterém budeme systém ODR řešit a třetím je vektor počátečních podmínek pro \mathbf{q} .

Pokud zavoláte funkci `ode45` tak, jak je uvedeno výše, dostanete jako výstup vektor $\mathbf{t} = \mathbf{tspan}$ a matici \mathbf{q} v následujícím tvaru

$$\mathbf{t} = \begin{pmatrix} t \\ 0.0 \\ 0.0001 \\ 0.0002 \\ 0.0003 \\ 0.0004 \\ 0.0005 \\ \dots \end{pmatrix} \quad \mathbf{q} = \begin{pmatrix} x & y & z & v_x & v_y & v_z \\ 0 & 0 & 0 & 100.0 & 0 & 0 \\ -0.0000 & 0 & 99.91 & -3.95 & 0 & 0 \\ 0.0005 & -0.0000 & 0 & 99.65 & -7.91 & 0 \\ 0.0007 & -0.0000 & 0 & 99.21 & -11.84 & 0 \\ 0.0010 & -0.0001 & 0 & 98.60 & -15.76 & 0 \\ 0.0012 & -0.0001 & 0 & 97.82 & -19.64 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix} \quad (3.15)$$

Sloupový vektor \mathbf{t} obsahuje hodnoty naší nezávislé proměnné (času) zatímco sloupce matice \mathbf{q} obsahují rychlosti a pozice (tedy závislé proměnné) v odpovídajících časech.

3.1.3 Implementace

jak už možná tušíte, budeme pro řešení pohybové rovnice v Matlabu potřebovat dva soubory. První z nich, `odefun.m` obsahuje funkci, která dává pravé strany našeho systému obyčejných diferenciálních rovnic a druhý, `solve.m` obsahuje počáteční podmínky pro řešič `ode45`, časový interval, na kterém se bude soustava ODR řešit a příkazy pro vykreslování. Oba soubory musí být umístěny v tomtéž adresáři a celý program se spouští přes soubor `solve.m`. Podívejme se nejprve na soubor `odefun.m`.

Program 1: `odefun.m`

```
function dqdt = odefun(t,q),
    % Toto je vstupní funkce pro
    % řešicí diferencialních rovnic, ode45.
    % Vstup: nezávislá proměnná t(cas), 6-rozmerný
    % vektor q
    % q = [x,y,z,v_x,v_y,v_z]
    % výstup: časová derivace q, dqdt = dq/dt
    % Elementarní náboj [C]
    % Hmotnost částice [kg]
    % Nyní si definujeme promenne popisujici polohu a
    % rychlosť
```

1

2

3

4

5

6

7

8

9

10

11

```
% Pomoci komponent vektoru q. 12
% Toto není nezbytný krok, ale používá například 13
% "x" namísto q(1), atd.. cíni kod nize 14
% vice citelnym a intuitivním 15
x = q(1); 16
y = q(2); 17
z = q(3); 18
vx = q(4); 19
vy = q(5); 20
vz = q(6); 21
22
Ex = 0; 23
Ey = 2e-6; 24
Ez = 0; 25
Bx = 0; 26
By = 0; 27
Bz = 1e-7; 28
29
% Nyní spočteme komponenty 6rozmerného vektoru dqdt 30
% vector dq/dt. 31
dxdt = vx; % dx/dt = v_x 32
dydt = vy; % dy/dt = v_y 33
dzdt = vz; % dz/dt = v_z 34
dvxdt = qe/m*(Ex + vy*Bz - vz*By); % dv_x/dt 35
dvydt = qe/m*(Ey + vz*Bx - vx*Bz); % dv_y/dt 36
dvzdt = qe/m*(Ez + vx*By - vy*Bx); % dv_z/dt 37
% Nakonec sestavime samotny vektor dqdt 38
% z jeho komponent. Tim, ze vektoru dqdt 39
% priradime hodnotu jsme nastavili vystup funkce 40
    odefun
dqdt = [dxdt; dydt; dzdt; dvxdt; dvydt; dvzdt]; 41
42
end % konec funkce 43
```

I když je tento soubor obsahující funkci `odefun` k poměrně bohatě komentován, zaměřme se ještě jednou na jeho nejdůležitější části. Jak jste se již naučili v první kapitole, musí každý soubor obsahující funkci začít definicí této funkce, tedy specifikací vstupních a výstupních proměnných. V našem případě máme na vstupu nezávislou proměnnou t a vektor závislých proměnných \mathbf{q} a funkce vrací derivaci tohoto vektoru $\dot{\mathbf{q}}$ (v ASCII zápisu `dqdt`)

Na řádcích 8 a 9 jsou definovány elementární náboj a na řádcích 16 až 22 jsou přejmenovány elementy vektoru \mathbf{t} tak, aby se nám s nimi dále lépe pracovalo. Toto přejmenování samozřejmě není nutné a mohli bychom dále v kódu místo x psát $q(1)$ atd. na řádcích 35 až 37, kde vyjadřujeme právě derivaci jednotlivých komponent q . Na řádcích 23 až 28 potom definujeme komponenty elektrického a magnetického pole.

R Definovat konstanty q_0 a m v těle funkce není z programátorského hlediska optimální, pro lepší čitelnost kódu si to ale můžeme dovolit. Problém spočívá v tom, že během řešení soustavy ODR bude funkce `odefun()` zavolána mnohokrát a opakovaná definice též proměnné zabírá zbytečně systémové prostředky a navíc nemáte konstanty definovány všechny na jednom místě. Dále v tomto textu se naučíme, jak běh programu optimalizovat pomocí *globálních proměnných*.

Nyní, když jsme definovali funkci dávající pravou stranu systému obyčejných diferenciálních rovnic (3.13), můžeme konečně přistoupit k jejímu řešení a vykreslení výsledků. Potřebné příklady uložíme do souboru `solve.m`.

Program 1: `solve.m`

```
% Temto Matlabový skript \v{r}e\v{s}í pohybovou rovnici      1
    pro
% nabítou \v{c}ástici v p\v{r}ítom nosit elektrického      2
% a magnetického pole.                                     3
%                                                               4
% Nejprve musíme definovat po\v{c}áte\v{c}ní podmínky      5
% Pro p\v{r}ípomenutí, q = [x,y,z,v_x,v_y,v_z]           6
q0 = [0;0;0;1e2;0;0];                                     7
%                                                               8
% Nyní definujeme \v{c}asový interval, na kterém          9
% budeme \v{r}e\v{s}it soustavu ODE.                         10
ti = 0;                                                 11
tf = 2.5e-3;                                             12
N = 1000;                                              13
timespan = linspace(ti, tf, N);                           14
% funkce linspace(ti, tf, N) vytvo\v{r}í vektor            15
% s lineárn\v{e} rostoucí hodnotami od "ti"             16
% po "tf" s "N" kroky/prvky.                            17
%                                                               18
% Nyní spustíme \v{r}e\v{s}í samotný.                      19
% Zápis @odefun \v{r}íká matlabu, \v{z}e prvním          20
% argumentem funkce ode45 je jiná funkce.              21
[t, q] = ode45(@odefun, timespan, q0);                 22
% Výsledkem bude 1xN vektor t a 6xN matice q          23
%                                                               24
% Následující p\v{r}íkazy zobrazí spo\v{c}ítanou
% trajektorii                                         25
mesh([q(:,1) q(:,1)], [q(:,2) q(:,2)], [q(:,3) q(:,3)],   26
      [t(:) t(:)], 'EdgeColor', 'interp', 'FaceColor', 'none')
% vykreslení
view(2) % nastaví pohled shora                         27
set(gca, 'FontSize', 16, 'fontWeight', 'bold') % nastaví
    velikost písma
xlabel('x [m]') % popisek osy x                         29
ylabel('y [m]') % popisek osy y                         30
zlabel('z [m]') % popisek osy z                         31
title(['Particle trajectory, ti=', num2str(ti), ' s, tf='
       , num2str(tf), ' s']) % nadpis grafu            32
%                                                               33
print -depsc 'Output.eps' % uložení obrázku do souboru 34
```

3.1.4 Cvičení

Cvičení 3.1 Zkopírujte si zdrojové kódy z předchozího cvičení do počítače nebo použijte m-soubory v adresáři Prog1_Particle_motion.

- Spusťte program a ověřte, že jste dostali křivku podobnou té na obrázku 3.2.
- Jaký drift zde pozorujeme?
- Jaký je směr driftové rychlosti elektronu a pozitronu?

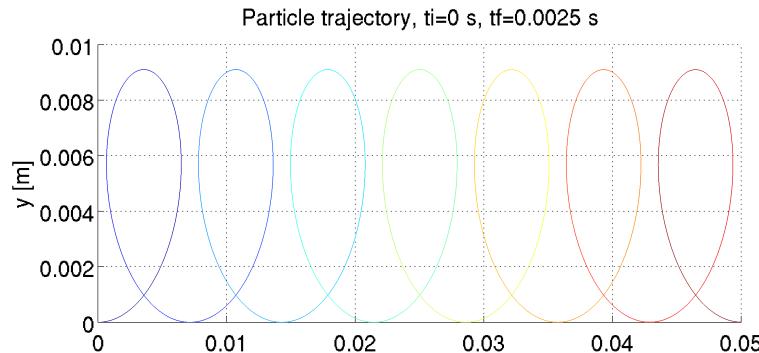


Figure 3.2: Trajektorie elektronu v $\mathbf{E} = (0, 10^{-6}, 0) \text{ V/m}$, $\mathbf{B} = (0, 0, 10^{-7}) \text{ T}$ a $\mathbf{q}(0) = (0, 0, 0, 100 \text{ m/s}, 0, 0, 0)$

Cvičení 3.2 Změňte náboj qe a hmotnost m tak, aby odpovídaly protonu to those of a proton.

- Kolikrát musíte zvětšit přibližně časovou škálu, aby jste mohli u protonu pozorovat trajektorii charakteristickou pro tento drift?
- Porovnejte amplitudy periodického pohybu elektronu a protonů a velikost jejich driftových rychlostí (data vyčtěte z grafu nebo z numerického řešení daného vektorem t a maticí q).

Cvičení 3.3 Upravte program ze cvičení 3.1 tak, aby byla pohybová rovnice řešena bez přítomnosti elektrického pole s magnetickým polem kvadraticky rostoucím ve směru osy y

$$\mathbf{E} = (0, 0, 0) \quad (3.16)$$

$$\mathbf{B} = (0, 0, B_z) \quad (3.17)$$

$$B_z = B_0 \left(\frac{y}{y_0} \right)^2 \quad (3.18)$$

$$\mathbf{q}_0 = (0, 0, 0, \sqrt{2}v_0, \sqrt{2}v_0, 0) \quad (3.19)$$

Zvolte konstantu y_0 a časový interval $\langle t_i, t_f \rangle$ tak, aby částice za tento čas urazila zhruba vzdálenost y_0 .

- Pozorujete nějaký drift? Pokud ano, pro jaké hodnoty B_0 , y_0 a v_0 jej pozorujete?
- Jaký je směr driftové rychlosti pro elektron a pro pozitron?

Pokročilé cvičení 3.1 Upravte program tak, aby se elektrické pole měnilo harmonicky v čase, například

$$E_x = E_0 \cdot \cos(\omega t), \quad (3.20)$$

$$E_y = 0, \quad (3.21)$$

$$E_z = 0, \quad (3.22)$$

a nastavte počáteční rychlosť na

$$v_x = 0, \quad (3.23)$$

$$v_y = v_0, \quad (3.24)$$

$$v_z = 0. \quad (3.25)$$

Nyní pro několik frekvencí ω (např. 10^6 , 10^7 , 10^8 a 10^9 Hz) pozorujte pohyb protonu a elektronu.

- Jak částice reagují na pole?
- Porovnejte, jak moc jsou elektron a proton ovlivněny elektrickým polem pro různé frekvence.

3.2 Van Allenovy radiační pásy

Objevení van Allenových radiačních pásů se často považuje za první velký objev v historii vesmírných letů. Jejich existence byla předpovězena Jamesem van Allenem a potvrzena družicí Explorer I v roce 1958. Pásy jsou přímým důsledkem magnetického pole Země, které zachycuje částice s vysokou energií a zabrání jim tak aby dopadly na povrch. Žádné stínění samozřejmě není ideální, takže vždy existuje určitý slabý tok částic směrem k povrchu. Tyto částice ionizují a excitují molekuly v nižších vrstvách atmosféry, což pozorujeme jako polární září (zobrazena v úvodu této kapitoly).

Existují dva stabilní van Allenovy pásy. První z nich je umístěn ve výšce cca 1 000 to 6 000 km a druhý cca 13 000 to 60 000 km [Bak12]. Třetí van Allenův pás byl objeven velmi nedávno a zdá se, že s postupem času mizí a objevuje se [Sci13; Shp+13].

Vzhledem k tomu, že nyní již máte povědomí o všemožných driftech, je formulace, že částice jsou *zachyceny* v magnetickém poli příliš vágňí. V tomto oddíle upravíme Program 1, který jsme použili dříve, a vyřešíme pomocí něj pohyb nabité částice v magnetickém poli Země. V průběhu se naučíme, co jsou *globální proměnné* a jak je použít v Matlabu. Také se seznámíme s několika pokročilými příkazy pro vykreslování. Tento oddíl je inspirován přednáškami S. Markidise [Mar13]..

3.2.1 Související rovnice

Řešené rovnice i metoda řešení jsou identické jako v předchozím oddíle 3.1. Hlavním rozdílem zde bude, že elektrické pole považujeme všude za nulové a magnetické pole bude silně záviset na poloze a budeme jej approximovat magnetickým polem dipólu s konstantním magnetickým dipólovým momentem \mathbf{m} .

R Považovat geomagnetické pole za čistě dipólové je poměrně hrubá aproximace, které s rozumnou přesností platí pouze do vzdálenosti několika R_e (poloměru Země). Dále od země je geomagnetické pole silně nesymetrické kvůli jeho interakci se solárním větrem. Většina experimentálního i teoretického výzkumu geomagnetického pole, je podporována

NASA v rámci programu **Living with a star** (lws.gsfc.nasa.gov). Pokročilé modely geomagnetického pole a vesmírného počasí potom naleznete například na ccmc.gsfc.nasa.gov.

Ve vektorovém zápisu je magnetické pole dipolu dáno výrazem

$$\mathbf{B}(\mathbf{r}) = \frac{\mu_0}{4\pi} \left(\frac{3\mathbf{r}(\mathbf{m} \cdot \mathbf{r})}{r^5} - \frac{\mathbf{m}}{r^3} \right) \quad (3.26)$$

kde \mathbf{m} je magnetický moment, μ_0 permeabilita vakua a

$$\mathbf{r} = (x, y, z), \quad (3.27)$$

$$r = |\mathbf{r}|. \quad (3.28)$$

Cvičení 3.4 Budeme pracovat v kartézské soustavě souřadnic se středem ve středu Země a osa z splývá se zemskou osou. V tomto systému je magnetický moment planety

$$\mathbf{m} = (0, 0, M). \quad (3.29)$$

- Vyjádřete komponenty magnetického pole $\mathbf{B} = (B_x, B_y, B_z)$ v kartézských souřadnicích.
- Jaká je přibližně hodnota M pokud na rovníku naměříme magnetické pole $3.12 \cdot 10^{-5}$ T?

■

3.2.2 Implementace

Stejně jako v předchozím případě musíme definovat funkci `odefun()`, která nám bude dávat pravou stranu naší soustavy ODR. Tato funkce bude skoro stejná jako v předchozím případě.

Program 2: `odefun.m`

```

function dqdt = odefun(t,q),
% Toto je vstupní funkce pro
% řešení diferenciálních rovnic, ode45.
% Vstup: nezávislá proměnná t(cas), 6-rozměrný
% vektor q
% q = [x,y,z,v_x,v_y,v_z]
% výstup: časová derivace q, dqdt = dq/dt
%
% V soubory solve.m, definujeme několik globálních
% proměnných. Zde rekneme
% Matlабu, chceme ve funkci použít promenne "m" a "
% qe" definované v jiném souboru
global m; global qe;
%
% Proznačení poloh a rychlosti.
x = q(1);
y = q(2);
z = q(3);
vx = q(4);
vy = q(5);
vz = q(6);
%
% Elektrické pole bude nulové

```

```

Ex = 0;                                21
Ey = 0;                                22
Ez = 0;                                23
                                    24
% Magnetické pole je nyní počítáno pomocí jiné
% externí funkce, bfield(r), která je funkci pouze
% polohy a je uložena v souboru bfield.m
r0 = [q(1), q(2), q(3)]; % vektor polohy      26
B = bfield(r0); % funkce vráci vektor mag. pole 27
% ... a my opět pro nazornost komponenty preznacíme.
Bx = B(1);                            29
By = B(2);                            30
Bz = B(3);                            31
                                    32
% Odsud dale je vše stejné jako v Programu 1
dxdt = vx;          % dx/dt = v_x           33
dydt = vy;          % dy/dt = v_y           34
dzdt = vz;          % dz/dt = v_z           35
dvxdt = qe/m*(Ex + vy*Bz - vz*By); % dv_x/dt   36
dvydt = qe/m*(Ey + vz*Bx - vx*Bz); % dv_y/dt   37
dvzdt = qe/m*(Ez + vx*By - vy*Bx); % dv_z/dt   38
dqdt = [dxdt; dydt; dzdt; dvxdt; dvydt; dvzdt]; 39
                                    40
end % konec funkce                      41
                                    42

```

První rozdíl oproti předchozímu programu vidíme na řádku 10. Místo toho, abychom proměnným m a qe přiřadili nějakou určitou hodnotu, prohlásili jsme je pouze za *globální proměnné*. Pokud vám myšlenka globálních a lokálních proměnných není známá, podívejte se na následující poznámku.

R Globální proměnná je proměnná, které je dostupná ve všech funkčích a programech, ve kterých byla prohlášena za globální. Běžné proměnné jsou naopak lokální a nejsou automaticky sdíleny mezi funkcemi a programy.

Představte si situaci, kdy chcete použít proměnnou m jak v `solve.m` tak v `odefun.m`.

- Pokud použijete **lokální proměnnou**, musíte ji nastavit číselnou hodnotu v obou těchto souborech tím, že do obou napíšete $m = 1.627 \cdot 10^{-27}$;
- Pokud ale použijete **globální proměnnou**, nastavíte numerickou hodnotu $m = 1.627 \cdot 10^{-27}$ v souboru `solve.m`, který se spustí jako první, a deklarujete, že proměnná je globální tím, že napíšete global m ; jak do `odefun.m` tak do `solve.m`. Tím pádem je **číselná hodnota uvedena jen na jednom místě**.

Výhoda druhého přístupu je zjevná. Pokud budete kdykoliv v budoucnu potřebovat hmotnost změnit najinou hodnotu, stačí je přepsat na jednom místě.

Druhý rozdíl v souboru `odefun.m` naleznete kolem řádku 26. Jak vidíte, magnetické pole zde není zadáno předpisem, ale pomocí další externí funkce uložené v souboru `bfield.m`, která rovněž používá některé z globálních proměnných. Níže vidíte neúplný zdrojový kód této funkce, do kterého musíte doplnit správné výrazy pro B_x , B_y , B_z (cvičení 3.4).

Program 2: `bfield.m`

```

function B = bfield(r0),
% Tato funkce spočítá geomagnetické pole B na pozici
% r0. Pracuje s kartézskými souřadnicemi.               1
2

```

```

% Funkce pouziva nasledujici globalni promenne (      3
    vizte solve.m):
global Re; global q; global m; global mu0; global M; 4
5
% opet preznaceni
x = r0(1); 6
y = r0(2); 7
z = r0(3); 8
r = sqrt(x^2+y^2+z^2); 9
10
% Doplnte nasledujici radky na zaklade
    odpovidajiciho cviceni. 11
12
Bx = XXX; 13
By = YYY; 14
Bz = ZZZ; 15
16
B = [Bx; By; Bz]; 17
18
end

```

Na závěr potřebujeme soubor se skriptem, obsahujícím všechny nezbytné proměnné, počáteční podmínky a vykreslovací příkazy.

Program 2: solve.m

```

% Tento skript resi pohybovou rovnici      1
% pro nabitu castici v magnetickem poli Zeme 2
3
% Abychom program zrychlili, deklarujeme vsechny nase
% konstanty jako globalni promenne      4
5
global Re; global m; global qe; global mu0; global M; 6
Re = 6378137;           % Polomer zeme v metrech 7
m = 1.627e-27;          % Hmotnost castice v kg 8
qe = 1.602e-19;         % Naboj castice v C 9
esc = 1;                % Skalovaci faktor pro vykreslovani 10
c = 299792458;          % Rychlosť svetla in m/s 11
mu0 = 4*pi*1e-7;        % Permeabilita vakua, V*s/(A*m) 12
M = 7.94e22;             % Magneticky moment Zeme, H/m 13
14

% Po\v{c}áte\v{c}ní podmínky pro polohu ...
% ... definujeme nejprve ve sferickyh souradnicich 15
16
r0 = 3*Re;
phi0 = 0; 17
theta0 = pi/2; 18
% ... a transformujeme do kartezskych 19
20
x0 = r0*sin(theta0)*cos(phi0)
y0 = r0*sin(theta0)*sin(phi0)
z0 = r0*cos(theta0)

% Pocatecni podminky pro rychlosť
% Definovane pomoci kin. energie a 2 uhlu 25
26
Ek_eV = 5e7;           % energie [eV] 27

```

```

Ek = Ek_eV*1.602e-19; % energie [J] 28
v_r0 = c*(1+m*c^2/Ek)^-0.5 % velikost rychlosti ( 29
    relativisticky) [m/s]
v_phi0 = 0; % azimut [rad] 30
v_theta0 = pi/4; % polarni uhel [rad] 31
                                         32
% Transformace souradnic pro rychlost
vx0 = v_r0*sin(v_theta0)*cos(v_phi0) 33
vy0 = v_r0*sin(v_theta0)*sin(v_phi0) 34
vz0 = v_r0*cos(v_theta0) 35
                                         36
                                         37
% Vektor pocatecnich podminek
q0 = [x0;y0;z0;vx0;vy0;vz0]; 38
                                         39
                                         40
% Definovani casoveho intervalu
ti = 0; % pocatecni cas [s] 41
tf = 20; % konecny cas [s] 42
N = 10000; % pocet kroku 43
timespan = linspace(ti,tf,N); 44
                                         45
                                         46
% Reseni systemu ODR
% Nezapomente, ze odefun(t,q) se nyni lisi od Prgramu 1 47
[t, q] = ode45(@odefun, timespan, q0); 48
                                         49
                                         50
                                         51
% Po vyreseni rovnice je treba data vykreslit
% Zde nemusite nutne rozumet, co dela ktery 52
% z prikazu nize 53
                                         54
                                         55
h = figure; % Vytvorime novy graf 56
                                         57
% Nasledujici prikazy vykresli magnetické silocary. [ 58
    Upraveno z MagLForce skriptu od A. Abokhodaira]
n = 4; 59
d2r = pi/180; r2d=1/d2r;
tht=d2r*(0:5:360)';
phi=d2r*(0:ceil(180/n):180);
hh=phi*r2d;
A=r0;
r=A*sin(tht).^2;
rho=r.*sin(tht);
x=rho*cos(phi); y=rho*sin(phi);
[nR,nC]=size(x);
u=ones(1,nC); z=r.*cos(tht)*u;
plot3(x/Re,y/Re,z/Re,'r','LineWidth',1.0); 60
                                         61
                                         62
                                         63
                                         64
                                         65
                                         66
                                         67
                                         68
                                         69
                                         70
                                         71
hold on; % Tento prikaz rika Matlabu, ze cokoliv budeme 72
dale vykreslovat ma byt soucasti tehoz grafu!!!
                                         73

```

```

% Vykresleni elipsoidu predstavujiciho Zemi.                                74
[u,v,w]=sphere(30);                                                        75
surf(esc*u, esc*v, esc*0.9*w);                                              76
colormap('default');                                                       77
camlight right;                                                               78
lighting phong;                                                               79
axis equal % Nastavi stejne meritko os.                                     80
                                                               81

% Vykresleni trajektorie, pridani popisku os                                82
plot3(q(:,1)/Re, q(:,2)/Re, q(:,3)/Re, 'LineWidth',1.0)                   83
set(gca,'FontSize',18) % velikost pisma                                         84
xlabel('x [R_e]') % popisek osy x                                           85
ylabel('y [R_e]') % popisek osy y                                           86
zlabel('z [R_e]') % popisek osy z                                           87
title(['Proton trajectory, E = 50 MeV, t_i=' , num2str(ti)                88
      ), ' s, t_f=' , num2str(tf), ' s']) % nadpis grafu
                                                               89

print -dpng -r200 'ProtonMagField.png' % ulozi obrazek                      90
do souboru ve formatu png s rozlisenim 200 dpi.

```

Obsah souboru solve.m je v principu také podobný tomu předchozímu. Na řádcích 6 až 13 definujeme globální proměnné. Konstanty, které budeme chtít používat jako **globální proměnné je nejlepší definovat a přidělit jim číselné hodnoty na začátku prvního souboru, který bude spuštěn**. Na řádcích 15 až 39 definujeme počáteční podmínky pro rychlosť a polohu. Pro lepší představitelnost definujeme počáteční podmínky nejprve ve sférických souřadnicích a následně je transformujeme do kartézských. Příkaz pro řešení systému ODR je na řádku 49 a od řádku 57 dále jsou vypsány příkazy pro vykreslování. Všimněte si zejména příkazu hold on, který Matlabu říká, že všechny další objekty se budou vykreslovat do téhož grafu. Také si povšimněte, že před vykreslením dat všechny proměnné škálujeme poloměrem Země Re.

Pokud jste chybějící výrazy pro mag. pole v souboru bfield.m uvedli správně, program by měl pro předdefinované počáteční podmínky vykreslit graf podobný tomu na obrázku 3.3.

Proton trajectory, E = 50 MeV, ti=0 s, tf=20 s

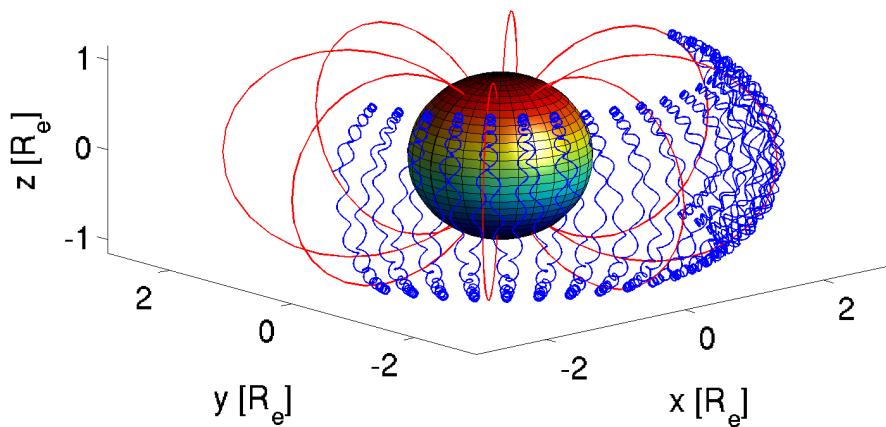


Figure 3.3: Pohyb protonu s vysokou energií v geomagnetickém poli

Cviení 3.5 Popište pohyb vysoce energetického protonu v geomagnetickém poli, jaké tři komponenty pohybu můžeme pozorovat? Podívejte se na pohyb částice v rovinách xy případně yz . ■

Cviení 3.6 Přestože je model, který jsme vyvinuli, poměrně jednoduchý, může nám pomoci v pochopení základní struktury van Allenových radiačních pásů. Zkuste změnit počáteční polohu protonu.

- Jaká je maximální počáteční vzdálenost od Země, pro kterou má proton ještě stabilní dráhu?
- Jaká je minimální počáteční vzdálenost od Země, pro kterou proton ještě nedosáhne povrchu? ■

Pokročilé cvičení 3.2 Nahradťte proton v předchozím cvičení elektronem a zkuste najít počáteční podmínky (zejména vzdálenost od Země), pro které bude elektron mít stabilní dráhu. Očekáváte, že to bude dále nebo blíže povrchu? Jak se mění drift elektronu oproti protonu? ■

4. Zpracování dat

4.1 Úvod

Znalosti Matlabu, které získáte v této kapitole se Vám budou hodit nejen ve fyzice plazmatu. Kapitola se zabývá zpracováním dat včetně interpolace a integrace diskrétních dat. Kromě toho se naučíte jak používat *smyčky* a *podmínky*, jejichž znalost je velmi užitečná při hromadném zpracování velkých souborů dat.

4.2 Smyčky a podmínky v Matlabu

Podívejme se nejprve na dva velmi důležité prvky programování (nejen) v Matlabu, *smyčky* a *podmínky*. Pokud již máte předchozí zkušenosti s programováním, bude pro Vás tento oddíl poměrně triviální. Pokud jste se se smyčkami a podmínkami v programování zatím nesetkali, bude se Vám jejich znalost jistě hodit ve vaší vědecké kariéře.

Smyčky jsou programátorské struktury, které Vám umožní mnohokrát opakovat určitou sekvenci příkazů. Ve fyzice se může jednat například o

- vykreslování mnoha datových souborů v témž formátu,
- hromadný převod velkého objemu dat,
- řešení diferenciálních rovnic pro několik různých počátečních/okrajových podmínek.
- ...

V Matlabu, stejně jako v mnoha ostatních programovacích jazycích, existují dva druhy smyček, smyčky typu *for* a smyčky typu *while*. Smyčku typu *for* použijete, pokud chcete danou sekvenci příkazů spustit N -krát (například "zintegruj data v prvních 50 souborech"), zatímco smyčka typu *while* se bude spouštět neustále dokola, dokud se nesplní určitá podmínka (např. "integruj data, dokud nenajdeš integrál větší než 10"). Ve fyzice je nicméně použití smyčky typu *while* poměrně omezené a navíc je v případě nepozornosti možné vytvořit nekonečnou smyčku *while*. Z toho důvodu se zde zaměříme pouze na smyčku typu *for*, která by Vám ve většině fyzikálních aplikací měla stačit.

Podmínky se používají, má-li se určitá sekvence příkazů spustit jen tehdy, když je určitý výrok pravdivý a obzvláště se Vám budou hodit v kombinaci se *smyčkami*. Příkladem použití může být například hromadné vykreslení dat, která se vykreslí v logaritmické škále pokud je rozdíl mezi největší a nejmenší hodnotou alespoň dva řády.

4.2.1 Smyčka for

Jak už bylo zmíněno, smyčky typu *for* Vám umožní N -krát provést určitou operaci. Pokud má smyčka proběhnout desetkrát, použijete ji následujícím způsobem

```
for j=1:10,  
    % vase prikazy  
end
```

1
2
3

Proměnná *j* je proměnnou cyklu, která postupně nabývá hodnot od 1 do 10 a můžete ji používat uvnitř cyklu. Ukažme si nyní použití cyklů na konkrétním programu, konkrétně vykreslí funkci $y = x^n$ pro $n \in \{1, 6\}$ a $x \in \langle 0, 1 \rangle$.

```
figure; % vytvorí grad  
colors = [ 'r', 'g', 'b', 'm', 'k', 'y', 'c', 'r' ]; %  
    toto je vektor textových promenných  
for j=1:8, % will run for 8 values of j  
    x = linspace(0,1,50); % vektor délky 50 s komponentami  
        od 0 do 1  
    color = colors(j); % vybere j-ty prvek vektoru colors  
    plot(x, x.^j, color);  
    hold on; % chceme vše v jednom grafu  
end
```

1
2
3
4
5
6
7
8

Cvičení 4.1 Upravte program výše tak, že vykreslí

$$y = \frac{\sin(nx)}{x} \quad \text{pro } x \in \langle 0, 2\pi \rangle \text{ a } n \in \langle 1, 5 \rangle \quad (4.1)$$

Nezapomeňte, že *x* je vektor a musíte tedy použít prvkové operace. ■

Ve fyzice plazmatu i v dalších odvětvích fyziky je často třeba pracovat s daty z literatury a často se stane, že data, která jsou k dispozici jsou v jiných jednotkách, než potřebujete. V dalším cvičení proto vytvoříte program, který bude hromadně konvertovat účinné průřezy zadané v jednotce cm^2 do jednotky m^2 . Abyste tento program mohli napsat, budete potřebovat navíc funkce **load()** a **dlmwrite()**.

Funkce **load()** funguje velmi intuitivně, načte data ze souboru do Matlabové proměnné. Vstupní data mohou mít několik formátů (hodnoty oddělené tabulátorem, středníkem, csv, atd...) a do proměnné je načtete příkazem

```
matrix = load('datafile.dat');
```

1

Naopak pokud chcete uložit data do ASCII souboru, použijete funkci **dlmwrite()**, která funguje podobně jednoduše.

```
dlmwrite('new_datafile.dat', matrix, '\t');
```

1

Jak vidíte, bere si funkce tři parametry, název výstupního souboru, proměnnou, kterou chcete uložit a symbol oddělující sloupce dat. Běžně používané oddělovače sloupců jsou ', ' (čárka), ';' (středník) a '\t' (tabulátor).

Poslední věc, kterou musíte znát jsou základy práce s řetězci v Matlabu. Řetězec je termín používaný v programování de facto pro textové proměnné a vzhledem k tomu, že řetězec v Matlabu je vlastně jenom vektor znaků s ním můžete pracovat podobně s tím rozdílem, že části řetězce musí být vždy uzavřeny v jednoduchých uvozovkách (').

Ukažme si spojování řetězců na příkladu. Následující skript otevře a vykreslí postupně obsah souborů csk1.dat až csk3.dat (tyto soubory můžete najít v adresáři Prog3_cross_section_convert).

```

colors = [ 'r' , 'g' , 'b' , 'm' , 'k' , 'y' , 'c' , 'r' ] ;           1
figure;                                         2
for j=1:3,                                     3
    color = colors(j);
    matrix = load(['csk', num2str(j), '.dat']);          4
    xdata = matrix(:,1); % první sloupec dat            5
    ydata = matrix(:,2); % druhý sloupec dat           6
    plot(xdata, ydata, color);                         7
    hold on;
end                                              8
xlim([0,1e6]); % nastavi rozsah osy x           9
                                                 10
                                                 11

```

Data, která vám tento program vykreslí nejsou náhodné křivky, ale jsou to účinné průřezy srážek elektronů s argonovými atomy. Na ose x je elektronová teplota v jednotce Kelvin a na ose y účinný průřez v m^2 . Konkrétně soubor csk1.dat obsahuje data pro elastickou srážku



soubor csk2.dat data pro excitaci argonu



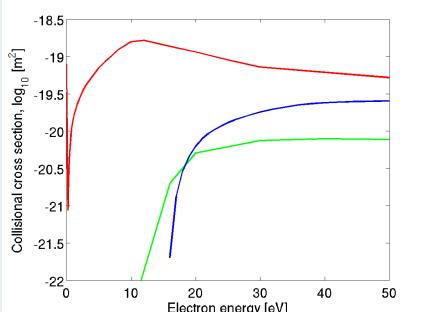
a soubor csk3.dat data pro ionizaci



Cvičení 4.2 Program uvedený výše vykreslí graf, který není vzhledem k velmi rozdílným magnitudám účinných průřezů příliš intuitivní. Upravte program tak, aby

- byla elektronová teplota vyjádřena v jednotce eV,
- a osa y byla logaritmická.

Výsledný graf by měl vypadat podobně jako ten následující



kde červená křivka odpovídá elastické srážce, zelená křivka excitaci a modá křivka ionizaci.

- Vyčtěte z obrázku, jaký je ionizační a excitační práh argonu.

Cvičení 4.3 Upravte program tak, že převede teplotu elektronů z Kelvin na elektronvolty a uloží konvertovaná data do souboru csevN.dat (kde N je odpovídající číslo souboru). ■

4.2.2 Podmínky if ... elif ... else

V předchozím oddíle byla popsána smyčka typu *for*, která umožňuje opakování provádět nějaký úkon. Dalším důležitým komponentem pokročilejších programů jsou *podmínky*. Použitím podmínky můžete dosáhnout toho, že pokud je určitý výrok pravdivý, provede se jiný příkaz než když je výrok nepravdivý.

Předpokládejme například, že vyvíjíte určitý složitý program, ve kterém často potřebujete hledat inverzní matice. Jak jistě víte, inverzní matice A^{-1} k matici A je taková, že

$$A^{-1}A = I \quad (4.5)$$

kde I je jednotková matice.

Cvičení 4.4 Funkce, která za Vás v Matlabu spočítá inverzní matici se jmenuje `inv()` a jejím jediným argumentem je matice, kterou chcete invertovat.

- Definujte 3×3 singulární matici A a pokuste se ji invertovat funkcí `inv()`. Jak vypadá výsledná matice?

Jak jste viděli v předchozím cvičení a jak i jistě víte, snaha spočítat inverzní matici k singulární matici nevede k ničemu příliš užitečnému a pokud k něčemu takovému dojde ve vašem složitém hypotetickém programu, nebudeste tušit, kde se stala chyba. Mnohem praktičtější by bylo definovat vlastní funkci, která inverzi provede jen když vstupní matice není singulární. Pokud matice singulární je, měla by Vaše funkce vypsat chybu a ukončit program. Funkce, kterou jsme právě popsali může vypadat následovně.

```

function inverted = myinv(matrix),
    rows = length(matrix(:,1)); % spočita delku prvního
    sloupce = pocet radku
    cols = length(matrix(1,:)); % spočita delku prvního
    radku = pocet sloupca
    if(rows ~= cols), % pokud se rows NEROVNA cols
        disp('The supplied matrix is not a square matrix
              ')
        return % ukonci skript
    elseif(det(matrix) == 0); % pokud je determinant
        nula
        disp('The matrix is singular.')
        return % ukonci skript
    else, % pokud jsou obe podmínky vyše nepravda
        inverted = inv(matrix); % provede inverzi a
        vrati invertovanou matici
    end
end

```

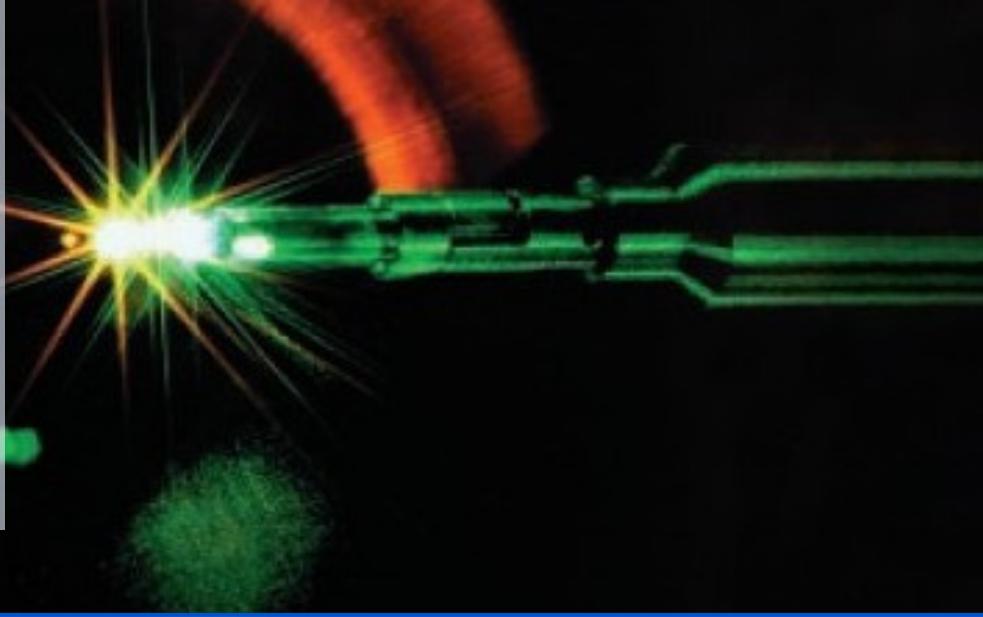
Syntaxe podmínky `if...elseif...else` by měla být zjevná z kódu uvedeného výše. Vidíte, že výraz, jehož pravdivostní hodnota se ověřuje, je uveden v závorkách za `if` nebo `elseif`. Matlab postupuje procedurálně a podmínky testuje od první k poslední, tedy

- Nejdříve zkонтroluje tvar matice. Pokud není čtvercová, zobrazí se chyba.
- Pokud je matice čtvercová, otestuje se její determinant. Pokud je nulový, zobrazí se chyba.
- Pouze pokud je matice čtvercová a není singulární** se provede inverze.

Cviení 4.5 Ověřte, že funkce `myint()` se chová správně, vyzkoušejte ji na několika maticích, které nejsou čtvercové a na maticích, které jsou singulární. Přidejte další `elseif`, které vypíše varování pokud je hodnota vstupní matice vyšší než 10. Vyzkoušejte. ■

Pokročilé cvičení 4.1 Podívejte se do adresáře `Prog4_cross_section_convert_if`, který obsahuje tři soubory s účinnými průřezy. Dva z těchto souborů obsahují teplotu elektronů v eV a jeden z nich v Kelvin.

- Upravte program ze cvičení 4.2 přidáním smysluplné `if...else` podmínky, která rozhodne, zda by soubor měl být konvertován nebo ne.
- Podobný jednoduchý program by byl velice užitečný například při vývoji kolizně-radiačního modelu plazmatu, kde je třeba hromadně předzpracovat tisíce podobných souborů. Jaká jsou omezení tohoto programu? ■



5. Interakce částic v plazmatu

Tato kapitola ukazuje rozdíly mezi rovnovážnými a nerovnovážnými rozdělovacími funkcemi a měla by Vám zprostředkovat to, jak závisí makroskopické proměnné na tvaru rozdělovací funkce. Na příkladech z fyziky plazmatu se zde naučíte základy **numerického integrování** a **interpolace** v Matlabu.

5.1 Maxwell-Boltzmannovo rozdělení

5.1.1 Střední rychlosť

Jak již víte z přednášky, rozdělení rychlosti se v rovnovážném stavu řídí **Maxwell-Boltzmannovým** rozdělením. Rozdělovací funkce velikosti $F(v)$ rychlosti pro částice se třemi stupni volnosti potom je

$$F(v) = 4\pi nv^2 \left(\frac{m}{2\pi kT} \right)^{3/2} \exp \left(\frac{-mv^2}{2kT} \right). \quad (5.1)$$

Rozdělovací funkce, zejména ty elektronové, Vám toho o plazmatu mohou poměrně hodně prozradit, pokud víte, jak je interpretovat. Především je důležité pochopit, jaký vliv mají rozdělovací funkce na makroskopické proměnné, které vnímáme intuitivně. Vzhledem k tomu, že mnoho makroskopických veličin může být velmi často vyjádřena jako integrály rozdělovací funkce, měli byste se v tomto oddíle naučit, jak numericky integrovat v Matlabu.

R Některé z integrálů, které v tomto oddíle budeme počítat numericky mají také analytické řešení. Nicméně ve většině laboratorních výbojů platí, že rozdělovací funkce určitých typů částic (především elektronů) nelze popsat analytickým výrazem. Proto zde upřednostníme numerickou integraci před analytickou. Schopnost numericky integrovat data samozřejmě upotřebíte i v jiných odvětvích fyziky.

Existuje několik způsobů, jak numericky integrovat data. Nejjednodušším z nich je *lichoběžníkové pravidlo*, které si pravděpodobně pamatuji z matematické analýzy. Při použití lichoběžníkového pravidla je funkce approximována lichoběžníky a celková plocha těchto lichoběžníků je potom odhadem určitého integrálu mezi body a a b . Lichoběžníkové pravidlo je schematicky znázorněno na obrázku 5.1(a). Pokročilejší metodou integrace je takzvané *Simpsonovo pravidlo*, které funkci $f(x)$, kterou chceme integrovat, approximuje několika po

částech spojitymi polynomy $P_i(x)$ (viz obrázek 5.1(b)). Proto dosahuje Simpsonova metoda vyšší přesnosti pro rychle se měnící funkce.

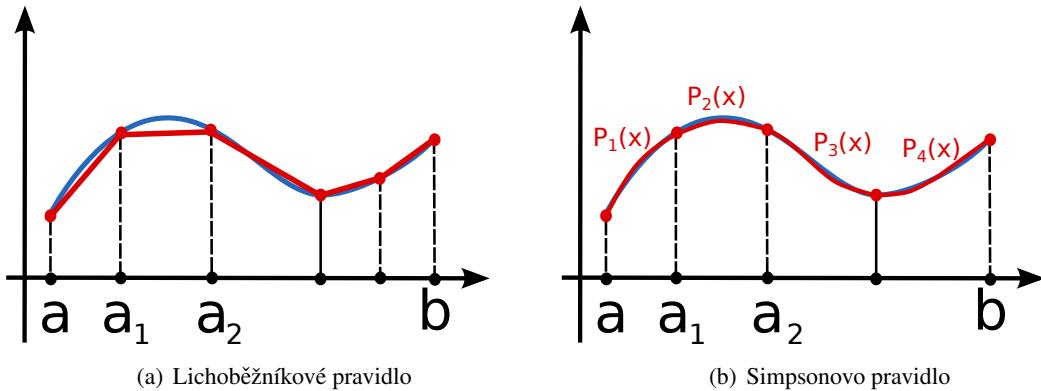


Figure 5.1: Ilustrace lichoběžníkového pravidla pro numerickou integraci (vlevo) a Simpsonova pravidla (vpravo). V případě Simpsonova pravidla je horní strana lichoběžníků dáná polynomem, který je analyticky integrovatelný.

Z obrázku 5.1(a) je také zřejmé, že při snižování šířky lichoběžníků poroste přesnost této metody. Pro naše účely **bude lichoběžníkové pravidlo mít dostatečnou přesnost**. Algoritmus, který provádí integraci lichoběžníkovým pravidlem je v Matlabu implementován pod jménem `trapz()`. Vstupní argumenty funkce jsou dva vektory, první z nich obsahuje nezávislou proměnnou x a druhý funkci, kterou chceme integrovat, $f(x)$.

```

x = linspace(0, pi, 100);
fx = sin(x.^2) ./ log(x);
I = trapz(x, fx);

```

Jak vidíte, není u integrálu třeba udávat integrační meze. Je to proto, že integrační meze jsou dány minimální a maximální hodnotou vstupního vektoru x . Jak již chápete, tři řádky kódu výše spočítají následující určitý integrál

$$I = \int_{x_{\min}}^{x_{\max}} \frac{\sin(x^2)}{x} dx = \int_0^\pi \frac{\sin(x^2)}{x} dx \quad (5.2)$$

V navazujícím cvičení se naučíte používat funkci `trapz()` a otestujete její přesnost na příkladu, který má i analytické řešení.

Cvičení 5.1 Program 5 uvedený níže je šablona pro vykreslení Maxwell-Boltzmannova rozdělení a pro spočtení středních rychlostí odpovídajících určité rozdělovací funkci. Program dokončete tím, že

- dopолните выражение для $F(v)$ на строке 7 (рассчитав функцию распределения скорости частиц в 3D, концентрации частиц забудьте ознакомиться),
- вычислите среднюю скорость на строке 12 и среднюю квадратичную скорость на строке 13 (оба помочь ломаного правила),
- дополните аналитически вычисленные выражения для средней скорости, средней квадратичной скорости и наиболее вероятной скорости на строках 16 и 18.

Až dokončíte program, odpovězte na následující otázky

- Проверяется ли численно полученные и аналитически вычисленные значения?

- Která ze tří rychlostí v grafu je nejnižší a která nejvyšší? Mění se jejich pořadí pokud měnите teplotu?
- Zkuste změnit počet kroků ve funkci `linspace()`, například na 10, 50, 100, 500. Jak se při tom mění shoda numericky spočítaných hodnot s analytickými?

Program 5: Maxwell-Boltzmann distribution and speeds

```

m0 = 1.6605e-27; % a.m.u. v kilogramech          1
m = 40*m0;       % hmotnost castice v kg        2
kB = 1.380e-23;  % Boltzmannova konst. in m^2 kg s^-2 K 3
    ^-1
T = 1000;         % teplota v Kelvin           4
                  5
v = linspace(0,4000,500); % x data             6
Fv = ...; % rozdelovaci funkce                 7
plot(v, Fv);           8
hold on;               9
                  10
% numericka integrace
v_mean = ...; % stredni rychlos          11
v_sq_mean = ...; % stredni kvadraticka rychlos 12
                  13
% analyticke vyrazy
v_mean_an = ...; % stredni rychlos          14
v_sq_mean_an = ...; % stredni kvadraticka rychlos 15
v_mp_an = ...; % nejpravdepodobnejsi rychlos 16
                  17
lineht = max(Fv)*1.1;      % vyska car v grafu 18
plot([v_mean, v_mean], [0, lineht], 'r'); % stredni 19
    rychlost (cervena)
plot([v_sq_mean, v_sq_mean], [0, lineht], 'g'); % 20
    stredni kvadraticka rychlost (zelena)
plot([v_mp_an, v_mp_an], [0, lineht], 'm'); % 21
    nejpravdepodobnejsi rychlos (modra)
hold off                         22
                                23
                                24

```

Pokroilé cvičení 5.1 V předchozím programu jsme numericky spočítali střední rychlos a střední kvadratickou rychlos, ale nejpravdepodobnejší rychlos byla spočtena analyticky. Abyste mohli spočítat nejpravdepodobnejší rychlos analyticky, musíte najít maximum rozdělovací funkce (zadaná vektory v a Fv). Zeptejte se Internetu, jak to udělat.

Jak víte z přednášek, rozdělovací funkce musí splňovat normovací podmínku

$$\int_0^\infty F(v)dv = n. \quad (5.3)$$

Proto, pokud integrujete rozdělovací funkci od určité rychlosti v_0 do ∞ , dostanete počet částic, které mají velikost rychlosti větší nebo rovnou v_0 . Pokud integrujete $F(v)$ od v_0 do v_1 , dostanete počet částic s velikostí rychlosti v intervalu $\langle v_0, v_1 \rangle$. Tohoto se týká cvičení 5.2.

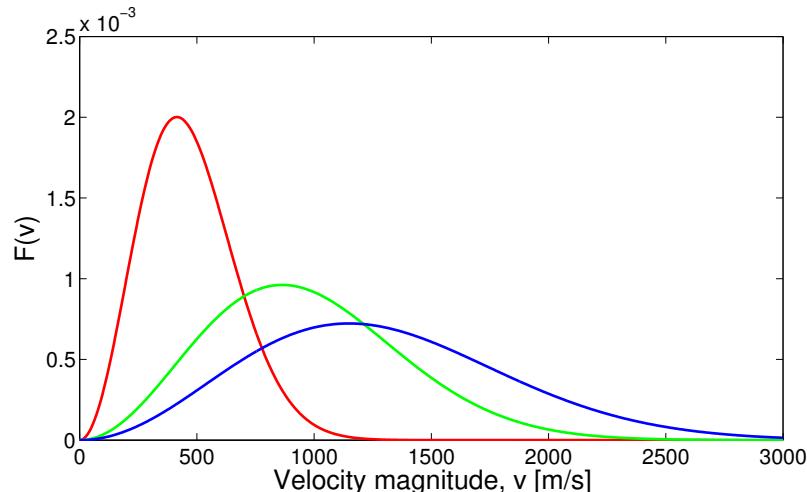
Cvičení 5.2 Hmotnost dusíkových molekul je 28 a.m.u. a jejich koncentrace za standardních podmínek je cca $1.7 \cdot 10^{25} \text{ m}^{-3}$. Kolik dusíkových molekul v místnosti, kde se právě nacházíte je rychlejších než 50, 500, 1000, 2500, 5000 a 10 000 m/s? ■

5.1.2 Tvar Maxwell-Boltzmannova rozdělení

Pravděpodobně jste si všimli, že rovnovážné **Maxwell-Boltzmannovo rozdělení je dáno jen dvěma makroskopickými parametry**, konkrétně hmotností částice m a teplotou T . Následující cvičení nevyžaduje žádné programování, pokud ale náhodou nevíte odpověď, můžete použít program výše.

Cvičení 5.3 Následující obrázek ukazuje tři rovnovážná rozdělení velikosti rychlosti

- Pokud je hmotnost částic m stejná ve všech třech případech, která z rozdělovacích funkcí odpovídá nejnižší teplotě a která nejvyšší?
- Pokud je konstantní teplota T , které rozdělení odpovídá částicím z nejnižší hmotnosti a které s nejvyšší?



5.2 Časový vývoj rozdělovací funkce

V předchozím oddíle jsme rozebrali klíčové vlastnosti Maxwell-Boltzmannova rozdělení. Nicméně tohoto rozdělení se dosahuje obecně pouze v termodynamické rovnováze a mimo ni může být rozdělovací funkce v podstatě libovolná. Navíc jsme zatím uvažovali pouze rozdělení velikosti rychlosti, čímž jsme předpokládali homogenní a izotropní rozdělovací funkci rychlosti. Zde budeme stále předpokládat, že prostor je homogenní (nezávisí na souřadnici \mathbf{r}), ale rozdělovací funkce již nebude izotropní (tedy může záviset na vektoru rychlosti \mathbf{v} , nejen na jeho velikosti).

Program, pomocí kterého budeme analyzovat časový vývoj rozdělovací funkce není příliš komplikovaný. Je založen na nejjednodušší nerovnovážné formulaci Boltzmannovy kinetické rovnice, ve které je prostor homogenní, síla působící na částice je nulová

$$\mathbf{F} = 0, \quad (5.4)$$

a srážkový člen je vyjádřen v Krookově tvaru

$$\left(\frac{\partial f}{\partial t} \right)_{\text{coll}} = -\frac{f - f_0}{\tau} = -v_m \cdot (f - f_0) \quad (5.5)$$

kde f_0 je rovnovážná rozdělovací funkce rychlosti, τ je relaxační doba a v_m je srážková frekvence. Celá Boltzmannova kinetická rovnice se tímto redukuje na

$$\frac{\partial f(\mathbf{v}, t)}{\partial t} = -v_m \cdot (f(\mathbf{v}, t) - f_0(\mathbf{v})) \quad (5.6)$$

a má následující analytické řešení

$$f(\mathbf{v}, t) = f_0(\mathbf{v}) + [f(\mathbf{v}, 0) - f_0(\mathbf{v})] e^{-v_m t}. \quad (5.7)$$

Pokud přijmeme tuto nejjednodušší approximaci, není ani nutné řešit Boltzmannovu kinetickou rovnici a stačí jen vykreslit řešení. A právě toto dělá program, jehož zdrojový kód je uvedený níže. Kód je opět komentován, zaměřte se ale zejména na následující řádky.

- **Řádek 5** nastavuje časový interval, po který je řešení vykreslováno.
- **Řádek 6** nastavuje rozpětí rychlostí, podobně jako předchozí program.
- **Řádky 11 až 13** nastavují počáteční rozdělovací funkci rychlosti
- **Řádek 32** Udává škálu grafů. Pokud jej zakomentujete, budou mít grafy proměnlivou škálu a pokud bude odkomentován, bude škála stále stejná.

Program 6: Vykreslení časového vývoje rozdělovací funkce.

```

m0 = 1.6605e-27; % a.m.u. v kilogramech          1
m = 40*m0;      % hmotnost castice e v kg        2
kB = 1.380e-23; % Boltzmannova konstanta v m^2 kg s^-2 3
K^-1
Vm = 5e8;       % srazkova frekvence v Hz           4
time = linspace(0, 1e-8, 100); % casovy interval    5
v = linspace(-2000,2000,200); % interval rychlosti   6
                                         7
% pocatecni rozdelovaci funkce, f_0            8
sigma = 10; % sirka                            9
v0 = 500; % nejpravdepodobnejsi rychlost        10
fx_init = 1/(sqrt(2*pi)*sigma)*exp(-(v-v0).^2/(2*sigma^2)) 11
);
fy_init = v*0;
 fz_init = v*0;

% Spocteni rovnovazne teploty M-B rozdeleni (plyne ze Z.
Z.E)
v_sq = sqrt(trapz(v, (fx_init+fy_init+fz_init).*v.^2));
Teq = 1/3*m*v_sq^2/kB;                         16
                                         17
                                         18
% rovnovazna rozdelovaci funkce
fx_eq = sqrt(m/(2*pi*kB*Teq)) * exp(-m*v.^2/(2*kB*Teq)); 19
fy_eq = sqrt(m/(2*pi*kB*Teq)) * exp(-m*v.^2/(2*kB*Teq)); 20
fz_eq = sqrt(m/(2*pi*kB*Teq)) * exp(-m*v.^2/(2*kB*Teq)); 21
                                         22
                                         23
fx = fx_init;
fy = fy_init;
fz = fz_init;
                                         24
                                         25
                                         26
hFig = figure; % vytvorime graf, nastavime rozmery 27
                                         28

```

```

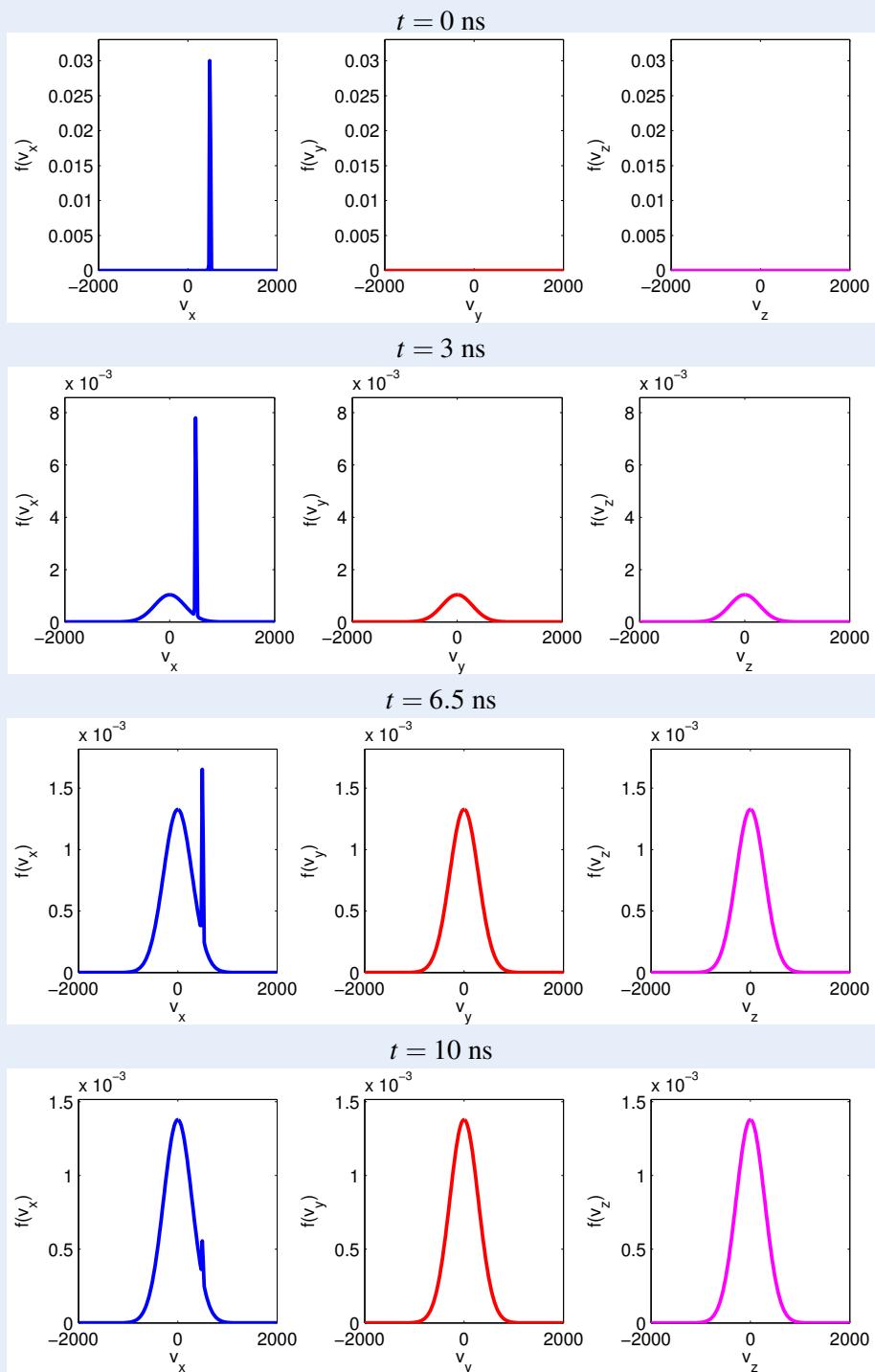
set(hFig, 'Position', [100 300 1000 300]);                                29
for j=1:length(time), % smycka for dela kroky v case
    absmax = max([fx, fy, fz])*1.1; % promenliva skala
    osy y
% absmax = max([max(fx_init),max(fy_init),max(
    fz_init)]); % pro konstantni skalu osy y
    odkomentujte
subplot(1,3,1); % první podobrazek - v_x
plot(v, fx, 'b', 'LineWidth', 2);
ylabel('f(v_x)');
xlabel('v_x');
ylim([0, absmax]);
whitebg('white')
subplot(1,3,2); % druhý podobrazek - v_y
plot(v, fy, 'r', 'LineWidth', 2);
ylabel('f(v_y)');
xlabel('v_y');
ylim([0, absmax])
subplot(1,3,3); % třetí podobrazek - v_z
plot(v, fz, 'm', 'LineWidth', 2);
ylabel('f(v_z)');
xlabel('v_z');
ylim([0, absmax])
% nastaveni zahlavu obrazku
ax=axes('Units','Normal','Position',[.075 .075 .85 .
    85], 'Visible','off');
set(get(ax,'Title'),'Visible','on')
title(['t = ', num2str(time(j)), ' s']);
% spočteni novych hodnot fx, fy, fz
fx = fx_eq + (fx_init - fx_eq)*exp(-time(j)*Vm);
fy = fy_eq + (fy_init - fy_eq)*exp(-time(j)*Vm);
fz = fz_eq + (fz_init - fz_eq)*exp(-time(j)*Vm);
M(j) = getframe(gcf); % pripoji snimek k promenne M
end
movie2avi(M, 'out.avi', 'fps', 7); % ulozi animaci

```

Cvičení 5.4 Spusťte program výše, prohlédněte si výstupní animaci a odpovězte na otázky.

- Jak můžeme fyzikálně interpretovat počáteční podmínku pro rozdělovací funkci?
- Jaký typ částic by mohla rozdělovací funkce popisovat?
- Srážkovou frekvenci jsme nastavili na smysluplnou hodnotu $5 \cdot 10^8$ Hz. Jaký je řádově čas potřebný k dosažení rovnováhy?
- Zkuste v programu 6 zvýšit nebo snížit srážkovou frekvenci. Jak se mění doba potřebná k dosažení rovnováhy?

Obrázky níže ukazují možný výstup programu 6. Je třeba zdůraznit, že obrázky mají proměnlivé měřítko na ose y.



5.3 Rychlostní koeficienty

Na závěr si vyzkoušíme, jak různé rozdělovací funkce rychlosti ovlivňují makroskopické proměnné, v našem případě rychlostní koeficient reakce. Rychlostní koeficient reakce spočítáme z jejího celkového účinného průřezu σ_r jako

$$k_r = \langle \sigma_r(v) v \rangle = \int_0^\infty \sigma_r(v) v F(v) dv. \quad (5.8)$$

Ve vztahu výše již předpokládáme, že účinný průřez závisí pouze na rychlosti dopadající částice a nikoliv na úhlu srážky.

Při počítání integrálu (5.8) narazíme na jeden technický problém, konkrétně na interpolaci účinných srážkových průřezů $\sigma_r(v)$, protože **účinné průřezy většiny reakcí nelze vyjádřit analyticky a jsou k dispozici pouze jako tabelovaná data**. Vzhledem k tomu, že účinný průřez $\sigma_r(v)$ není většinou tabelován pro mnoho hodnot v , je potřeba tabelovaná data interpolovat.

V Matlabu lze interpolovat 1D data pomocí funkce `interp1()`. Řekněme, že máte funkci zadanou vektory `xdata` (nezávislá proměnná) a `ydata` (funkční hodnoty) a chcete získat funkční hodnoty pro jemnější vektor nezávislé proměnné `xinterp`. Syntaxe potom bude následující

```

xdata = [1, 5, 10, 15, 20];
ydata = [1, 35, 110, 235, 410];

xinterp = linspace(8,17,0.5);
yinterp = interp1(xdata, ydata, xinterp, 'pchip', 0);

```

Jak vidíte, funkce `interp1()` má 5 vstupních parametrů. První dva z nich udávají data, která budete interpolovat, tedy vektor nezávislých proměnných (`xdata`) a stejně dlouhý vektor funkčních hodnot (`ydata`). Vektor (`xinterp`) je potom jiný vektor nezávislých proměnných (typicky s jemnějším krokem), pro které chcete funkci interpolovat. Čtvrtý vstupní parametr je interpolační metoda a pátý (nula) udává, že se neprovádí extrapolace a **všechny body, které jsou mimo interval daný vektorem `xdata` budou považovány za nulové**.



Při použití funkce `interp1()` můžete zvolit z několika interpolačních metod. Nejpopulárnějšími volbami jsou `'nearest'`, u které jsou data interpolována na základě hodnoty nejbližšího souseda, `'linear'`, kdy je použita lineární interpolace a `'pchip'`, u které se používá polynomiální interpolace.

Zkusme si nyní spočítat rychlostní koeficient ionizace argonového atomu dopadem elektronu.



Výpočet provedeme pro dvě různé rozdělovací funkce, pro Maxwell-Stefanovu rozdělovací funkci a pro skoro mono-energetický svazek částic, které tím pádem **mají skoro stejnou rychlosť**. Program by neměl obsahovat nic, co jsme se do této chvíle nenaučili. Jeho výstupem jsou dvě čísla, rychlostní koeficient pro Maxwell-Boltzmannovo rozdělení a rychlostní koeficient pro zmíněný mono-energetický svazek.

Program 7: Ionizace dopadem elektronu

```

m = 9.109e-31; % hmotnost elektronu
kB = 1.380e-23; % Boltzmannova konst. v m^2 kg s^-2 K^-1

```

```

TeV = 15; % teplota elektronu v eV
T = TeV*11604; % teplota elektronu v K

% nacteni ucinneho prurezu
sigdata = load('sigmaion.dat');
xsig = sigdata(:,1);
ysig = sigdata(:,2);

v = linspace(0,2e7,1e6); % vektor nezavislych promennych
- velikosti rychlosti
sigma = interp1(xsig, ysig, v, 'pchip', 0); % zjisteni
funkcnich hodnot sigma(v)

% Maxwell-Boltzmannovo rozdeleni
Fv = 4*pi*v.^2.*((m/(2*pi*kB*T))^(1.5).*exp(-m*v.^2/(2*kB*T
)); % rozdelovaci funkce
kr_MB = trapz(v, v.*Fv.*sigma) % rychlostni koeficient
vmean = trapz(v, Fv.*v);

% Rozdelovaci funkce pro skoro monoenergeticky svazek
v0 = vmean;
width = vmean/100;
Fv2 = 1/(sqrt(2*pi)* width) * exp(-(v-v0).^2/(2*width^2));
kr_beam = trapz(v, v .* Fv2 .* sigma) % rychlostni
koeficient

```

Cviení 5.5 Spusťte program a porovnejte výsledné rychlostní koeficienty. Všimněte si, že nejsou shodné přestože použité rozdělovací funkce mají stejnou střední rychlosť. Vysvětlete, proč tomu tak je.

Cviení 5.6 Upravte program tak, že v průběhu vykreslí $\sigma_r(v)$ a odpovězte na následující otázky

- Jaký je ionizační práh v eV?
- Na jaké hodnotě dosahuje účinný průřez maxima?

Cviení 5.7 Spusťte program pro několik hodnot elektronové teploty (definována v eV na řádku 3).

- Jak se změní rychlostní koeficienty, když zvyšujete teplotu?
- Existuje taková teplota, pro kterou rychlostní koeficient mono-energetického svazku překročí rychlostní koeficient Maxwell-Boltzmannova rozdělení? Poskytněte vysvětlení, proč to je/není možné.

Pokročilé cvičení 5.2 Přepište Program 7 tak, aby místo velikosti rychlosti pracoval s energií elektronu. Používání energie je ve fyzice plazmatu mnohem častější.

- (R) Existuje několik veřejně přístupných databází, ze kterých můžete získat účinné průřezy.
Nejvíce vyčerpávající z nich je pravděpodobně LxCat, dostupná na lxcat.net. Další poměrně rozsáhlou databázi je databáze ALADDIN, dostupná na www-amdis.iaea.org/ALADDIN/.



6. Rovnováha částic v plazmatu

Jak víte, typické plazma obsahuje mnoho typů částic. Kromě neutrálních atomů a molekul v základním stavu se mohou vyskytovat excitované částice (popsáno v *Úvod do fyziky plazmatu interaktivně: část I*), kladné nebo záporné ionty a samozřejmě i elektrony, díky kterým je plazma udrženo. Kromě toho mohou mít různé typu částic různé teploty.

Pro různé aplikace plazmatu je třeba znát teploty a koncentrace různých typů částic v plazmatu. Například pro bioaplikace je důležité vytvořit plazma, které bude mít vysokou elektronovou teplotu zatímco teplota těžkých částic (iontů, neutrálů, excitovaných neutrálů) bude nízká, aby nedošlo k tepelnému poškození tkáně.

6.1 Formulace problému

V této poslední kapitole tohoto textu implementujeme program, který počítá koncentrace různých typů částic v argonovém výboji za atmosferického tlaku při zadané elektronové teplotě T_e a teplotě těžkých částic T_g . Model bude opět implementován v Matlabu a bude nula-rozměrný. Toto prakticky znamená, že koncentrace každého typu částic bude pouze funkcí času, nikoliv polohy

$$n_\alpha = n_\alpha(t), \quad (6.1)$$

což fyzikálně odpovídá homogennímu plazmatu.

6.1.1 Srážkový člen

V approximaci homogenního plazmatu, kterou jsme učinili se rovnice kontinuity pro typ částic α redukuje na následující obyčejnou diferenciální rovnici.

$$\frac{\partial n_\alpha}{\partial t} = S_\alpha. \quad (6.2)$$

Rovnice vypadá formálně poměrně jednoduše, musíme ale vzít v potaz, že srážkový člen S_α je složitou funkcí teplot (které jsou naštěstí v našem modelu konstantní), a koncentrací různých typů částic.

Obecně vyjádříme srážkový člen pro typ částic α jako sumu příspěvků od všech reakcí, kde částice typu α figurují jako produkty či reaktanty.

$$S_\alpha = \sum_r S_{\alpha,r}. \quad (6.3)$$

Příspěvek $S_{\alpha,r}$ udává, kolik částic typu α vznikne nebo zanikne díky reakci r za jednotku času. Konkrétní tvar $S_{\alpha,r}$ samozřejmě závisí na řádu reakce a v našem modelu použijeme reakce druhého řádu (**dvojčásticové reakce**) a reakce třetího řádu (**trojčásticové reakce**). Jak už jejich jména naznačují, v prvním případě se pro uskutečnění reakce musí srazit dvě částice, v druhém případě se musí srazit částice tří. Pamatujte, že řád reakce vám řekne jenom to, **kolik reaktantů do reakce vstupuje**, neříká nic o tom, kolik má reakce produktů.

R Ve skutečnosti **trojčásticové reakce neprobíhají**, ale ve fyzice plazmatu je často používáme jako approximaci mnoha dějů, zejména za vyšších tlaků. Trojčásticová reakce ve skutečnosti probíhá jako dvě bezprostředně navazující dvojčásticové reakce



Pokud je druhá z těchto reakcí výrazně rychlejší a pravděpodobnější než ta první, můžeme tento proces popsat zjednodušeně, trojčásticovou reakcí



Příkladem dvojčásticové reakce je ionizace argonového atomu dopadem elektronu (později budeme tuto reakci označovat R4).



V této reakci se srazí elektron s excitovaným atomem argonu a dodá mu energii potřebnou pro jeho ionizaci. Tato reakce se potom promítne do srážkových členů všech jejích reaktantů a produktů,

$$S_{e,R4} = +k_1(T_e, T_g) \cdot n_e \cdot n_{Ar^*}, \quad (6.8)$$

$$S_{Ar^+,R4} = +k_1(T_e, T_g) \cdot n_e \cdot n_{Ar^*}, \quad (6.9)$$

$$S_{Ar^*,R4} = -k_1(T_e, T_g) \cdot n_e \cdot n_{Ar^*}. \quad (6.10)$$

Vidíte, že příspěvek k srážkovému členu obsahuje kromě koncentrací částic i rychlostní koeficient reakce, který může záviset na teplotě elektronů i teplotě těžkých částic. Jiné teploty v tuto chvíli uvažovat nebudeme, protože za našich podmínek je relativně rozumné pracovat s **dvojteplotním modelem**, tedy modelem, ve kterém mají elektrony svou vlastní teplotu, ale všechny ostatní typy částic (neutrálky, ionty, excitované neutrálky) jsou v termodynamické rovnováze a mají teplotu T_g . Můžete si také všimnout, že jednotlivé příspěvky $S_{\alpha,r}$ výše se liší znaménkem, jehož volba znaménka je velmi intuitivní. Vzhledem k tomu, že reakce (R4) **vytvoří jeden nový elektron a jeden nový iont**, jsou znaménka u odpovídajících příspěvků ke srážkovým členům kladná. Na druhou stranu reakce **spotřebuje jeden excitovaný argonový atom** a odpovídající příspěvek ke srážkovému členu $S_{Ar^*,R4}$ je tedy záporný.

Příkladem **trojčásticové srážky** je například vznik molekulárního argonového iontu. Existence této částice vás možná překvapí vzhledem k tomu, že argon je inertní plyn, ale ve vysokotlakém plazmatu hraje tato částice poměrně důležitou roli. Reakce, které vede na vznik tohoto iontu je



Analogicky k dvojčásticovým reakcím přispívá tato trojčásticová reakce ke srážkovým členům následovně

$$S_{Ar_2^+,R8} = +k_8(T_g) \cdot n_{Ar} \cdot n_{Ar} \cdot n_{Ar^+}, \quad (6.12)$$

$$S_{Ar^+,R8} = +k_8(T_g) \cdot n_{Ar} \cdot n_{Ar} \cdot n_{Ar^+}, \quad (6.13)$$

$$S_{Ar,R8} = -k_8(T_g) \cdot n_{Ar} \cdot n_{Ar} \cdot n_{Ar^+}. \quad (6.14)$$

Tentokrát může rychlostní koeficient záviset jenom na teplotě těžkých částic T_g , protože do reakce nevstupují elektrony. Navíc se musí dát pozor na to, že rychlostní koeficient bude mít jinou jednotku (viz cvičení níže).

Cvičení 6.1 V naší formulaci má srážkový člen vždy jednotku $1/(m^3 \cdot s)$. Jaká je odpovídající fyzikální interpretace? Jaká je jednotka rychlostního koeficientu k_r pro dvojčásticové a trojčásticové reakce? ■

6.1.2 Reakční schéma

Reakční schéma pro náš 0D model bylo převzato z článku od Baevy a kol., který se zabývá simulacemi mikrovlnného pochodňového výboje v argonu [Bae+12]. Typy částic, které model obsahuje jsou uvedeny v tabulce 6.1.

Table 6.1: Seznam typů částic zahrnutých do modelu argonového plazmatu

Ar	atom argonu v zákl. stavu (koncentraci získáme ze stavové rovnice)
Ar*	excitovaný atom argonu (sdrůžuje resonanční a metastabilní 4s stavy)
Ar ⁺	atomový argonový iont
Ar ₂ ⁺	molekulární argonový iont
e	elektron

Mělo by být zdůrazněno, že rovnici kontinuity (6.2) budeme řešit jenom pro 4 typy částic, Ar*, Ar⁺, Ar₂⁺ and e. Koncentrace argonového atomu v základním stavu totiž může být spočtena ze stavové rovnice ideálního plynu, která pro jednoatomové plyny platí velmi dobře. Pokud by argon nebyl vůbec ionizován, byla by koncentrace argonových atomů jednoduše

$$n_{\text{Ar}} = \frac{p}{k_B T_g} \quad (6.15)$$

kde p je tlak, k_B je Boltzmannova konstanta a T_g je teplota těžkých částic. Nicméně, rovnice (6.15) neplatí, pokud je část argonu ionizována nebo excitována.

Cvičení 6.2 Jaká je skutečná koncentrace argonových atomů v základním stavu pokud koncentrace excitovaných atomů je n_{Ar^*} , koncentrace atomových iontů je n_{Ar^+} a koncentrace molekulárních iontů je $n_{\text{Ar}_2^+}$? ■

Typy částic uvedené v tabulce 6.1 mohou vstupovat do celkem 11 reakcí. Úplnou sadu těchto reakcí potom najeznete v tabulce 6.2. Toto reakční schéma předpokládá, že rozdělovací funkce elektronů i těžkých částic jsou Maxwellovské. Díky tomuto předpokladu a pokročilému fitování můžeme vyjádřit rychlostní koeficienty reakcí jako analytické funkce T_e a T_g .

Cvičení 6.3 Reakce (R1), (R3) a (R4) v tabulce 6.2 obsahují exponenciální část $\exp(-C/T_e)$, která je nulová při nízké elektronové teplotě a blíží se jedné při vyšších elektronových teplotách.

- Jaká je jednotka a fyzikální význam konstant 11.65, 15.76 a 4.11?
- Podle rychlostních koeficientů zkuste odhadnout, který z ionizačních kanálů bude pravděpodobně nejdůležitější. ■

Table 6.2: Reakční schéma pro vysokotlaké argonové plazma (převzato z [Bae+12]). Elektronová teplota je vždy vyjádřena v eV a teplota plynu v K.

reakce	rychlostní koeficient	jednotka
(R1) $e + Ar \longrightarrow e + Ar^*$	$4.9 \cdot 10^{-15} \cdot \sqrt{T_e} \cdot \exp(-11.65/T_e)$	m^3/s
(R2) $e + Ar^* \longrightarrow e + Ar$	$4.8 \cdot 10^{-16} \cdot \sqrt{T_e}$	m^3/s
(R3) $e + Ar \longrightarrow 2e + Ar^+$	$1.27 \cdot 10^{-14} \cdot \sqrt{T_e} \cdot \exp(-15.76/T_e)$	m^3/s
(R4) $e + Ar^* \longrightarrow 2e + Ar^+$	$1.37 \cdot 10^{-13} \cdot \sqrt{T_e} \cdot \exp(-4.11/T_e)$	m^3/s
(R5) $e + e + Ar^+ \longrightarrow e + Ar$	$8.75 \cdot 10^{-39} \cdot T_e^{-4.5}$	m^6/s
(R6) $e + Ar_2^+ \longrightarrow Ar + Ar^*$	$1.04 \cdot 10^{-12} \cdot (T_e/0.026)^{-0.67} \cdot \frac{1-\exp(-418/T_g)}{1-0.31\exp(-418/T_g)}$	m^3/s
(R7) $e + Ar_2^+ \longrightarrow e + Ar + Ar^+$	$1.11 \cdot 10^{-12} \cdot \exp\left(-2.94 - 3\frac{T_g-300}{T_e \cdot 11604}\right)$	m^3/s
(R8) $Ar^+ + 2Ar \longrightarrow Ar + Ar_2^+$	$2.25 \cdot 10^{-43} (T_g/300)^{-0.4}$	m^6/s
(R9) $Ar + Ar_2^+ \longrightarrow Ar^+ + 2Ar$	$0.522 \cdot 10^{-15} T_g^{-1} \exp(-15131/T_g)$	m^6/s
(R10) $Ar^* + Ar^* \longrightarrow e + Ar^+ + Ar$	$6.2 \cdot 10^{-16}$	m^3/s
(R11) $Ar^* + Ar \longrightarrow Ar + Ar$	$3.0 \cdot 10^{-21}$	m^3/s

Cvičení 6.4 Vyjádřete srážkové členy pro následující typy částic: e , Ar^* , Ar^+ , Ar_2^+ pomocí rychlostních koeficientů k_1 až k_{11} a koncentrací jednotlivých typů částic, n_{Ar} , n_e , n_{Ar^*} , n_{Ar^+} , $n_{Ar_2^+}$.

■

Jak vidíte, i pro tak jednoduchý plyn jakým je argon je počet reakcí poměrně vysoký. V plazmatu vzroste počet přípustných reakcí velmi dramaticky, pokud pracujeme v molekulárním plynu nebo ve směsi plynů. U molekulárních plynů musíme vzít v potaz rotační a vibrační excitace, disociaci a reasociaci molekul. Jen pro představu o složitosti, pro minimální popis plazmatu ve směsi O₂/N₂ byste potřebovali nejméně několik desítek reakcí. Pro popis plazmatu **v argonu a vlhkém vzduchu roste počet reakcí na cca 4000** [GB13].

6.2 Implementace

Nyní již známe všechno pro to, abychom mohli vyřešit systém obyčejných diferenciálních rovnic v následujícím tvaru

$$\frac{\partial}{\partial t} \begin{pmatrix} n_e \\ n_{Ar^*} \\ n_{Ar^+} \\ n_{Ar_2^+} \end{pmatrix} = \begin{pmatrix} S_e \\ S_{Ar^*} \\ S_{Ar^+} \\ S_{Ar_2^+} \end{pmatrix} \quad (6.16)$$

Formálně podobnou soustavu ODR jsme řešili v kapitole 3. Pokud si tedy nejste jistí ohledně čehokoliv, co se týká kódu a zejména řešiče `ode45()`, zkuste se znova podívat na cvičení ve zmíněné kapitole.

Vzhledem k tomu, že už se nejedná o Váš první program v Matlabu, poskytne vám tento studijní text pouze šablonu pro celý program a chybějící části budete muset doplnit. Spustitelný Matlabový skript, který obsahuje počáteční podmínky a volání řešiče vypadá následovně:

Program 8: solve.m

% nastaveni globalnych promennych	1
global p;	2
global kB;	3
global Tg;	4

```

global Te;
p = 1e5; % tlak v Pa
kB = 1.38e-23; % Boltzmannova konst. v m^2 kg s^-2 K^-1
Tg = 400; % Teplota plynу v Kelvin
Te = 2; % Teplota elektronu v eV

tsteps = 1000; % pocet casovych kroku
tspan = logspace(-11, -6, tsteps); % pro casove kroky
    nepouzijeme lineарne rostouci hodnoty, ale
    logaritmicky rostouci
% Pocatecni koncentrace v m^-3
nArs_init = 1e12; % Ar*
nArp_init = 1e12; % Ar+
nAr2p_init = 1e12; % Ar_2+
ne_init = nArp_init+nAr2p_init; % poc. podminka pro
    elektrony dana kvazineutralitou
% Vektor pocatecnich koncentraci
initial = [nArs_init, nArp_init, nAr2p_init, ne_init];

% reseni soustavy ODR
[t, sol] = ode45('odefun', tspan, initial);

% Vykresleni dat
close all
figure;
hold on;
nAr = p./(kB*Tg)-sol(:,2)-0.5*sol(:,3)-sol(:,1);
plot(log10(t), log10(nAr), 'c');
plot(log10(t), log10(sol(:,1)), 'r');
plot(log10(t), log10(sol(:,2)), 'b');
plot(log10(t), log10(sol(:,3)), 'm');
plot(log10(t), log10(sol(:,4)), 'k');
ylim([12, 26])
legend('Ar', 'Ar^*', 'Ar^+', 'Ar_2^+', 'electrons', 'Location', 'northwest')

xlabel('Time, log_{10} [s]');
ylabel('Number density, log_{10} [m^{-3}]');
title(['p=' num2str(p), ' Pa, T_g=' num2str(Tg), ' K,
    T_e=' num2str(Te), ' eV']);
set(gca, 'fontsize', 16);

```

Ve zdrojovém kódu výše by vás nic nemělo překvapit. Největší rozdíl oproti našim předchozím kódům řešícím soustavu ODR je na řádku 12. Konkrétně zde nepoužíváme lineárně rostoucí vektor pro nezávislou proměnnou (čas), ale používáme vektor, který roste logaritmicky. Je to proto, že většina veličin v plazmatu se v čase mění velmi rychle.

Stejně jako v kapitole 3, volá program `solve.m` řešič `ode45()`, abychom získali časový vývoj soustavy ODR. Funkce, která vrací pravé strany soustavy diferenciálních rovnic (6.16) se opět nazývá `odefun()` a je uložena v matlabovém souboru s odpovídajícím jménem.

Program 8: odefun.m

```

function dqdt = odefun(t, q),
    % Pouzijeme globalni promenne definovane v solve.m
    global p;
    global kB;
    global Tg;
    global Te;
    % vektor q obsahuje koncentrace Ar*, Ar+, Ar_2+ a
        elektronu
    nArs = q(1);
    nArp = q(2);
    nAr2p = q(3);
    ne = q(4);
    % koncentrace argonu je spocitana pomocí stavove
        rovnice a koncentraci zbyvajicich typu castic
    nAr = p./(kB*Tg)-nArs-0.5*nAr2p-nArp;
    % Rychlostni koeficienty jsou ulozeny v oddelenych
        souborech s funkcemi
    k1 = f_k1(Te, Tg);
    k2 = f_k2(Te, Tg);
    k3 = f_k3(Te, Tg);
    k4 = f_k4(Te, Tg);
    k5 = f_k5(Te, Tg);
    k6 = f_k6(Te, Tg);
    k7 = f_k7(Te, Tg);
    k8 = f_k8(Te, Tg);
    k9 = f_k9(Te, Tg);
    k10 = f_k10(Te, Tg);
    k11 = f_k11(Te, Tg);
    % Nyni musime definovat srazkove clenky
    SArs = +k1*ne*nAr ...
    -k2*ne*nArs ...
    -k4*ne*nArs ...
    +k6*ne*nAr2p ...
    -2*k10*nArs*nArs ...
    +k11*nArs*nAr;
    SArs = ...;
    SArs = ...;
    Se = ...;
    % a na zaver je vratime jakozto derivace vstupniho
        vektoru
    dqdt = [SArs; SArs; SArs; Se];
end

```

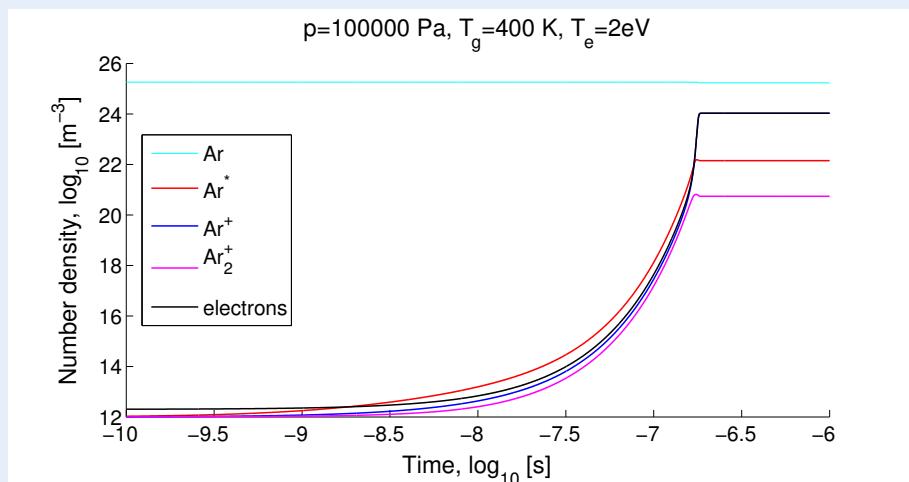
Ve funkci odefun() je třeba doplnit srážkové členy pro jednotlivé typy částic. Srážkový člen pro Ar^* (proměnná SArs) už byl předvyplněn, ale srážkové členy pro Ar^+ (proměnná SArs), Ar_2^+ (proměnná SArs2p) a elektrony (proměnná Se) **musíte doplnit na základě cvičení 6.4.** Navíc skript používá **rychlostní koeficienty** k1 až k11 **definované pomocí externích funkcí** f_k1() až f_k11(), které si berou **na vstupu teplotu těžkých částic T_g v Kelvin a teplotu elektronů T_e v elektronvoltech**. Samozřejmě, ne všechny rychlostní koeficienty budou funkcemi

jak T_g tak T_e ale pro větší konzistentnost jsou funkčím $f_k1()$ až $f_k11()$ předány obě teploty. Musíte tedy vytvořit 11 jednoduchých funkcí $f_k1()$ až $f_k11()$ na základě reakcí v tabulce 6.2.

Cvičení 6.5

Dokončete program výše, konkrétně

1. Do souboru `odefun.m` doplňte výrazy pro srážkové členy.
2. Vytvořte 11 souborů s funkcemi, `f_k1.m` až `f_k11.m`. Každý z těchto souborů bude počítat rychlostní koeficient odpovídající reakci. **Vezměte v potaz, že funkce musí být schopny práce s vektorovými argumenty a používejte s T_g a T_e pouze prvkové operace (`*`, `/`, `atd...`).**
3. Spusťte kód s předdefinovanými hodnotami tlaku p , T_g and T_e . Ověřte, zda výstup vypadá podobně jako následující obrázek.



Prosím vezměte v potaz, že jste první studenti pracující s tímto textem a není vyloženo, že autoři neudělali chybu při formulaci srážkových členů. Pokud váš výstup vypadá **lehce odlišně**, a jste přesvědčeni o správnosti vašich srážkových členů, neváhejte protokol odevzdat.

Nyní, když program funguje, můžete zkoušet změnit různé parametry, abyste získali kvalitativní představu o tom, jak argonové plazma funguje.

Cvičení 6.6

Zkuste zvýšit či snížit elektronovou teplotu a odpověďte na následující otázky.

- Jak a proč se mění rovnovážná koncentrace elektronů?
- Jak se mění doba potřebná pro zápal plazmatu?
- Jaký je dominantní iont ve fázi zápalu a který iont je dominantní po stabilizaci plazmatu? Mění se relativní zastoupení iontů s elektronovou teplotou?

Cvičení 6.7

Spusťte program pro vámi zvolenou hodnotu teploty elektronů a pro tlaky 10^3 , 10^4 , 10^5 a 10^6 Pa. Odpovězte na tyto otázky

- Jak a proč se mění rovnovážná koncentrace elektronů?
- Jak se mění doba potřebná pro zápal plazmatu?

6.3 Parametrická studie

Složitější programy podobné tomu, který jsme vyvinuli v tomto oddíle se často ve fyzice plazmatu používají pro zkoumání vlastností plazmatu za různých podmínek, ačkoliv počet reakcí je většinou výrazně vyšší (několik tisíc). Mnohdy je užitečné vědět, jak se bude složení plazmatu měnit pro různé teploty elektronů, teploty těžkých částic a tlaky.

Pokročilé cvičení 6.1 Modifikujte program v předchozím oddíle tak, že bude řešit soustavu diferenciálních rovnic pro více hodnot T_e or p a po ustálení vykreslí hustotu elektronů. Použijte při tom smyčku typu for.

To náročně na tomto cvičení je fakt, že doba zápalu se s teplotou elektronů a tlakem mění. Proto musíte v každém kroku smyčky for upravit délku časového intervalu podle aktuálních hodnot T_e a p . V opačném případě bude řešení trvat velice dlouho.

R Pokud se řeší podobné systémy diferenciálních rovnic pro mnoho typů částic a pro mnoho podmínek, musí se použít speciální numerické postupy, aby řešení trvali přijatelnou dobu. Jedním z jednoduchých triků je transformace rovnice kontinuity do logaritmického tvaru. Nezávislou proměnnou potom není koncentrace částic, ale její logaritmus

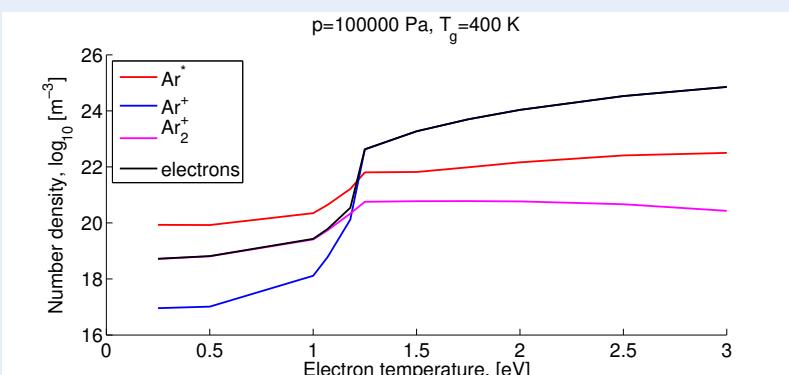
$$l_j = \ln n_j. \quad (6.17)$$

Tento trik může konvergenci velmi urychlit, protože zatímco hodnoty n_j se mění napříč mnoha řády, hodnoty l_j se většinou mění jen v rámci jednoho řádu.

Pokročilé cvičení výše může být zbytečně časově náročné zejména pro studenty, kteří neplánují Matlab dále používat. Nicméně, i v případě, že jste toto cvičení nedokončili, dokončete alespoň to následující.

Cvičení 6.8 Jak již bylo zmíněno, složení plazmatu se často dramaticky mění s elektronovou teplotou. Obrázek níže ukazuje rovnovážnou koncentraci různých typů částic jako funkci elektronové teploty při konstantním tlaku $p = 10^5$ Pa a teplotě $T_g = 400$ K. Promyslete si následující otázky a odpovězte na ně

- Pokud byste chtěli plazma použít jako zdroj světla, jakou elektronovou teplotu byste zvolili a proč?
- Poskytněte jednoduché vysvětlení pro pokles koncentrace Ar_2^+ s rostoucí T_e ?



6.4 Závěr

Tím se dostáváme na konec tohoto studijního textu. Gratulujeme, pokud jste došli až sem a podařilo se Vám dokončit všechna cvičení. Srdečně doufáme, že Vám cvičení a příklady v tomto textu alespoň trochu pomohly prohloubit pochopení komplexních procesů, které se v plazmatu odehrávají. Také doufáme, že schopnosti programovat v Matlabu, které jste se zde naučili, upotřebíte tom odvětví fyziky, kterým se rozhodnete zabývat.

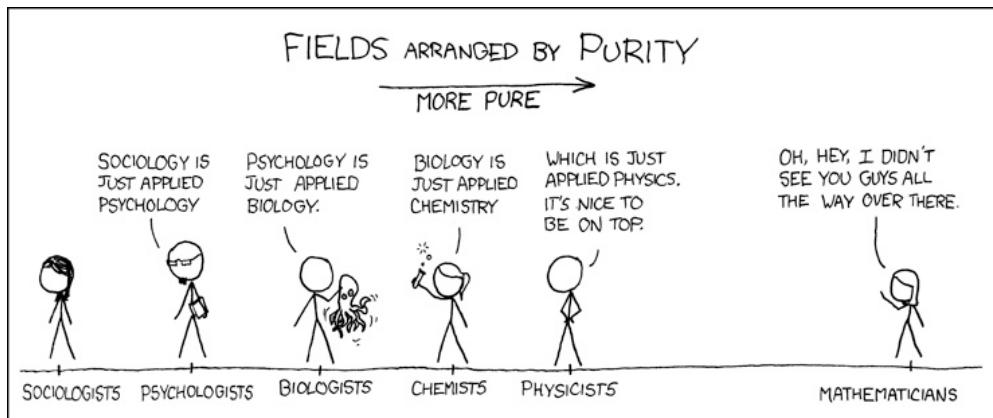
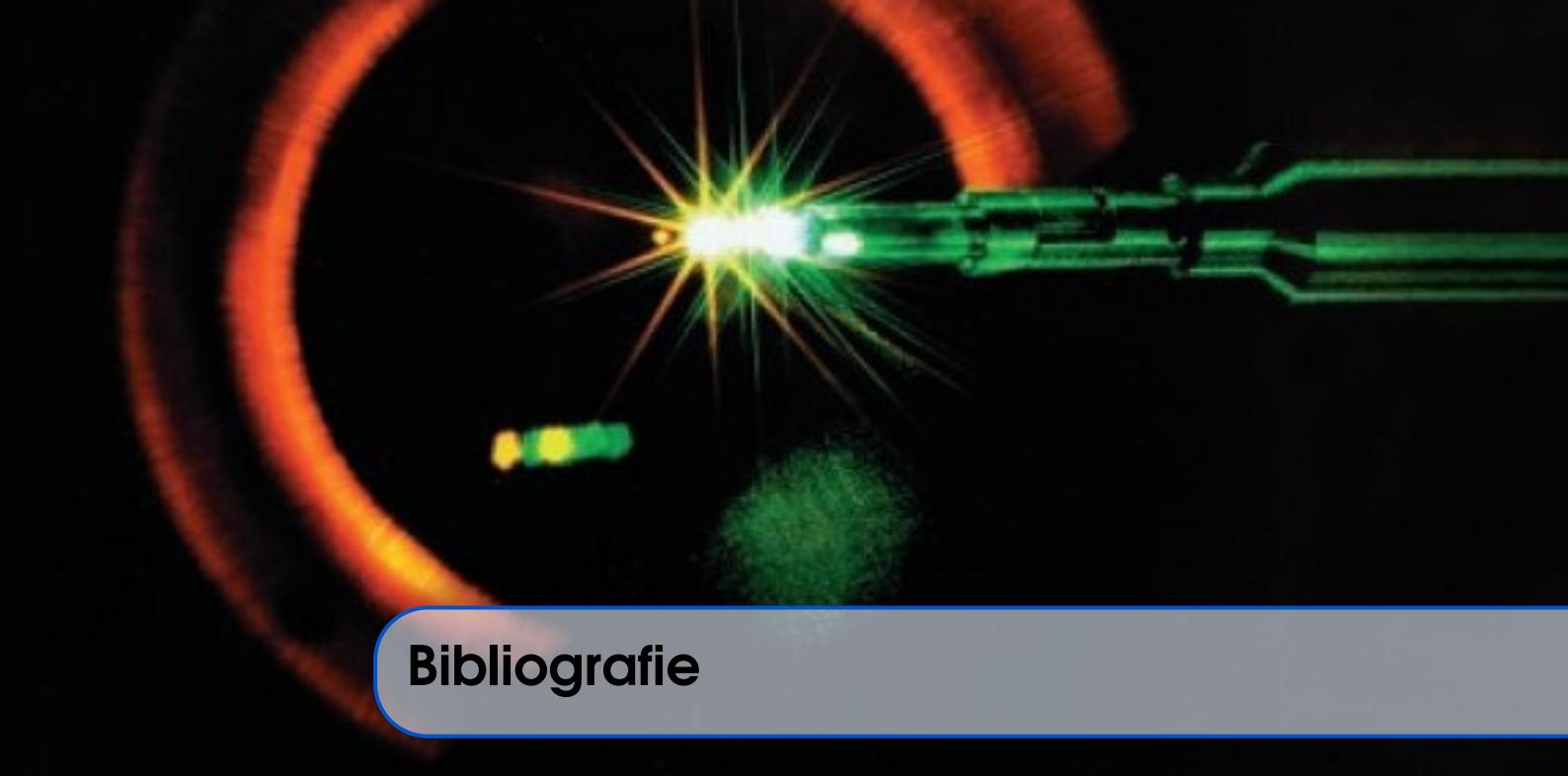


Figure 6.1: Staženo z xkcd.org



Bibliografie

Knihy

- [Bit04] J. A. Bittencourt. *Fundamentals of Plasma Physics*. 3rd edition. New York: Springer, 2004 (cited on page 13).
- [GK08] Dan M. Goebel and Ira Katz. *Fundamentals of Electric Propulsion: Ion and Hall Thrusters*. 1st edition. Pasadena: California Institute of Technology, 2008 (cited on page 16).

Články

- [Bae+12] M. Baeva et al. “Modeling of microwave-induced plasma in argon at atmospheric pressure”. In: *Physical Review E* 85.5 (May 2012), page 056404. ISSN: 1539-3755. DOI: 10.1103/PhysRevE.85.056404. URL: <http://link.aps.org/doi/10.1103/PhysRevE.85.056404> (cited on pages 47, 48).
- [Bak12] Daniel N Baker. “New Twists in Earth’s radiation belts”. In: *American Scientist* 102 (2012) (cited on page 21).
- [GB13] W Van Gaens and A Bogaerts. “Kinetic modelling for an atmospheric pressure argon plasma jet in humid air”. In: *Journal of Physics D: Applied Physics* 46.27 (2013), page 275201. URL: <http://stacks.iop.org/0022-3727/46/i=27/a=275201> (cited on page 48).
- [Shp+13] Yuri Y. Shprits et al. “Unusual stable trapping of the ultrarelativistic electrons in the Van Allen radiation belts”. In: *Nature Physics* 9.11 (2013), pages 699–703. ISSN: 1745-2473. DOI: 10.1038/nphys2760. URL: <http://www.nature.com/doifinder/10.1038/nphys2760> (cited on page 21).

Online zdroje

- [FEI10] FEI. *Introduction to Electron Microscopy*. July 2010. URL: <http://www.fei.com/documents/introduction-to-microscopy-document/> (cited on page 15).

- [Mar13] Stefano Markidis. *Lectures on Computational Plasma Physics*. Sept. 2013. URL: <https://www.pdc.kth.se/education/computational-plasma-physics/computational-plasma-physics> (cited on page 21).
- [Sci13] Scitechdaily. *Scientists Explain the Formation of the Unusual Third Van Allen Radiation Ring*. Sept. 2013. URL: <http://scitechdaily.com/scientists-explain-formation-unusual-third-van-allen-radiation-ring/> (cited on page 21).