FAKULTA MATEMATIKY, FYZIKY
A INFORMATIKY
UNIVERZITY KOMENSKÉHO BRATISLAVA
Katedra fyziky plazmy

Diplomová práca

Metóda Monte Carlo vo fyzike nízkoteplotnej plazmy

Ľubomír Host

Prof. RNDr. Viktor Martišovitš, DrSc.
(vedúci diplomovej práce)

(build 28. marca 2005)

BRATISLAVA 2003

Čestne vyhlasujem, že predkladanú diplomovú prácu som vypracoval samostatne s využitím uvedenej literatúry.

Ľubomír Host

Ďakujem môjmu diplomovému vedúcemu Prof. RNDr. Viktorovi Martišovitšovi, DrSc. za cenné rady a odbornú pomoc pri písaní diplomovej práce.

Ďalej chcem poďakovať mojim rodičom a priateľom za ich veľmi silnú podporu, trpezlivosť a toleranciu.

Obsah

Kapitola 1

Úvod

Fyzikálne bádanie využíva viacero metód skúmania fyzikálnych dejov alebo procesov. V poslednej dobe k teoretickým a experimentálnym metódam pribudla ďalšia metóda – počítačová alebo simulačná. Prudký rozvoj počítačovej techniky, ktorý zaznamenávame v dnešnej dobe, umožnil začať skúmať fyzikálne modely a procesy aj touto cestou. Nie je tomu inak ani vo fyzike plazmy.

Nie je možné povedať, ktorá z metód je najvýhodnejšia alebo lepšia, taktiež ani to, ktorá z metód je častejšie používaná. Každá má svoje výhody, ale aj svoje nevýhody alebo nedostatky. Všetky metódy však tvoria dokopy jeden ucelený systém a navzájom sa dopĺňajú. Bez ktorejkoľvek metódy by bola fyzika omnoho chudobnejšia a menej kompletná. Preto všetky prístupy k riešeniu problémov zastávajú svoju neoceniteľnú rolu.

Bola vypracovaná široká paleta simulačných metód. Jednotlivé metódy možno rozdeliť do viacerých skupín. Ako hlavné možno spomenúť fluidné metódy riešenia a metódy PIC (Particles In Cells).

Fluidné metódy riešenia sú založené na numerickom riešení makroskopických rovníc. Tieto rovnice popisujú plazmu ako kontinuum podliehajúce pôsobeniu vonkajších a vnútorných síl. Z toho vyplýva makroskopický charakter popisu plazmy.

Programy, ktoré simulujú plazmu na základe Vlasovovej rovnice, sa nazývajú Vlasovove kódy. Táto rovnica je použiteľná v bezzrážkovej plazme. Vlasovova rovnica spolu s Maxwellovými rovnicami úplne popisuje chovanie bezzrážkovej plazmy. V praxi však potrebujeme jednoduchší – pružnejší popis. To možno dosiahnuť použitím rôznych rýchlostných momentov Vlasovovej rovnice. Dostaneme rovnice pre vývoj v priestore a čase pre koncentráciu, strednú rýchlosť a tlak pre každý typ častíc.

Detailnejší popis plazmy možno dosiahnuť počítačovými simuláciami založenými na časticových kódoch (Particle Codes). Tieto metódy nechápu plazmu ako kontinuum, ale ako súbor veľkého počtu nabitých a neutrálnych častíc. Každá z častíc môže mať vlastnú štruktúru a jej parametre sú ovplyvnené interakciou s ostatnými časticami. Tým vzniká isté rozdelenie rýchlostí častíc a taktiež ostatných parametrov.

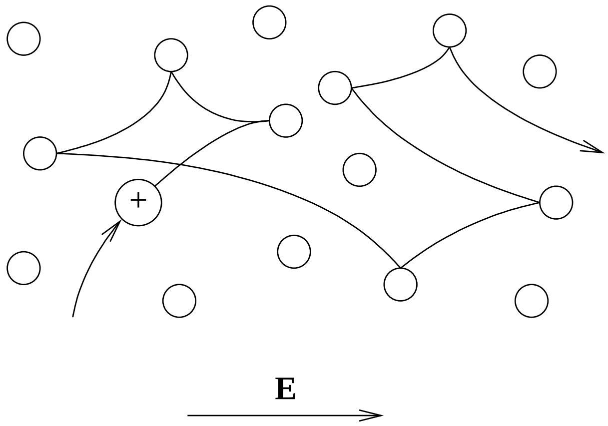
Takáto detailná simulácia však môže naraziť na niekoľko technických problémov. Vysoký počet častíc a komplexnosť ich vzájomných vzťahov (každá častica interaguje s ostatnými časticami) vyžaduje použitie vysokovýkonných superpočítačov. Napriek tomu, že výkon bežne dostupných počítačov každým rokom rýchlo rastie, je reálna simulácia fyzikálnych procesov napr. v plazme stále nedostupná. Preto je pri pokuse o počítačovú simuláciu fyzikálnych procesov nutné používať isté zjednodušenia, ktoré znižujú časové nároky a nároky na počítačové vybavenie na prípustnú hranicu. Naráža sa taktiež na principiálny problém. Aby sme mohli začať fyzikálny dej simulovať, potrebujeme poznať počiatočný stav všetkých uvažovaných častíc, teda mikrostav plazmy. To však nie je možné.

Jedným zo zjednodušujúcich predpokladov môže byť využitie Debye–Hückelovej teórie tienenia náboja. Na základe toho sa predpokladá, že častica neinteraguje so všetkými ostatnými časticami, ale iba s časticami v jej blízkom okolí (v guli s Debyeovým polomerom). Toto zjednodušenie výrazne znižuje výpočtové nároky. Tento model sa nazýva aj PIC (Particles In Cells) model.

Ďalším podstatným zjednodušením je simulovanie fyzikálneho procesu jedno- alebo dvojrozmerným modelom. Prechod z trojrozmerného modelu k dvojrozmernému nie je zložitý a možno ho urobiť, ak uvážime rotačnú symetriu jednej z priestorových súradníc. Takéto zjednodušenie výrazne zníži náročnosť numerických výpočtov, fyzikálny charakter procesu sa však takmer nezmenil. Prechod k jednorozmernému modelu prináša ďalšie výrazné zjednodušenie výpočtov, no tu už však nie je možné dostať plnú charakteristiku fyzikálnych parametrov. Ako príklad možno uviesť anizotropiu rýchlostnej distribučnej funkcie nabitých častíc v elektrickom poli naloženom kolmo na rovinné elektródy.

Táto práca sa zaoberá počítačovou simulačnou metódou Monte Carlo. Snahou je simulovanie procesu rezonančnej výmeny náboja jednak v aproximácii studeného plynu, ale aj s uvážením tepelného pohybu. Tento proces je pozorovaný najmä pri vzácnych plynoch: $He - He^+$, $Ne - Ne^+$, $Ar - Ar^+$ [5]. Z dôvodu výpočtovej zložitosti v trojrozmernom prípade bol použitý iba jednorozmerný model. Simulačná metóda, ktorú sme použili, považuje jednotlivé častice za ideálne tvrdé gule bez vnútornej štruktúry. Toto obmedzenie sa zdá byť opodstatnené, keďže vzácne plyny He, Ne, Ar sa vyznačujú vysokou stabilitou (sú malo reaktívne). Ich nízka reaktivita je spôsobená plným obsadením vonkajších elektrónových orbitálov. Makroskopické parametre, ako je driftová rýchlosť, distribučná funkcia a entropia, sú získavané detailnou simuláciou dostatočného počtu (rádovo $\approx 10^6$) častíc – iónov.

Hlavným zámerom tejto práce je štúdium prechodových javov pri náhlej zmene intenzity vonkajšieho elektrického poľa. Prechodové javy v aproximácii studeného plynu boli už v minulosti študované. Táto diplomová práca je snahou nadviazať na predchádzajúcu prácu [1] a rozpracovať teóriu rezonančnej výmeny náboja aj pri uvážení tepelného pohybu molekúl neutrálneho plynu.



Obrázok 1.1: Schématické znázornenie dráhy kladného iónu v plyne.

1.1 Základné pojmy

Uvažujme systém zmesi neutrálneho plynu a malej prímesi nabitých častíc – iónov. Na tento systém naložme homogénne vonkajšie elektrické pole. Predpokladajme, že nami naložené elektrické pole je teraz ešte stacionárne a homogénne. Vzhľadom na nízku koncentráciu nabitých častíc možno nimi vytvárané dodatočné pole zanedbať, a tak výsledné pole v systéme je úplne určené naloženým vonkajším poľom.

Sledujme teraz, čo sa deje s jednou nabitou časticou. Nech je to kladne nabitý ión. Elektrické pole ho urýchľuje v smere vektora intenzity elektrického poľa $\mathbf{E}$, no ión sa nepohybuje bez prekážok, ale zráža sa s molekulami neutrálneho plynu. Vzájomné zrážky iónov medzi sebou možno zanedbať vzhľadom na ich nízku koncetráciu. Zrážkami s neutrálnymi molekulami postupne ión stráca časť svojej energie a taktiež mizne usmernenosť jeho pohybu. Jeho pohyb sa stáva viacej náhodným. Výsledkom je zložitá cesta pomedzi molekuly v smere poľa $\mathbf{E}$ (viď obrázok 1.1).

Charakter pohybu iónu ovplyvňujú dva faktory: energia dodávaná elektrickým poľom a disipovaná energia pri zrážkach. Ak je energia, dodaná iónu elektrickým poľom za jednotku času, rovná energii, disipovanej za rovnaký čas pri zrážkach, tak vravíme o *hydrodynamickom* transportnom režime. Jedná sa o akúsi rovnovážnu situáciu, nie však v termodynamickom zmysle. Majme na pamäti tento zmysel aj naďalej.

Hydrodynamický režim vyžaduje ustálenosť energetických a rýchlostných pomerov. Ión sa teda pri svojom transporte plynom nachádza v stále rovnakých podmienkach: homogénne pole, neutrálny plyn. Náš systém je časovo aj priestorovo symetrický. Tento režim nastáva po dostatočne dlhej dobe od vstreku iónov do plynu. Doba, kým dôjde k takémuto ustáleniu pomerov, závisí od charakteru vstreku, intenzity elektrického poľa a typu častíc. Možno však očakávať súvis so strednou dobou medzi zrážkami τ.

V prípade, že je nami študovaný systém priestorovo alebo časovo nesymetrický, jedná sa o *nehydrodynamický* transportný režim. Nesymetria študovaného systému môže byť spôsobená gradientom elektrického poľa alebo časovou závislosťou problému.

1.2 Zoznam použitých symbolov

$\mathbb{R}$	množina reálnych čísel
$\mathsf{R}(0,1)$	rovnomerné rozloženie náhodných čísel na intervale $(0,1)$
e	náboj elektrónu, $e = 1.602176462 \times 10^{-19}$ C
k	Boltzmannova konštanta, $k = 1.3806503 \times 10^{-23}$ J.K^{-1}
m_u	atómová hmotnostná konštanta, $m_u = 1.66053873 \times 10^{-27}$ kg
m, m_g	hmotnosť molekuly neutrálneho plynu
E	veľkosť intenzity elektrického poľa
$\mathbf{E}$	vektor intenzity elektrického poľa
$a, \mathbf{a}$	zrýchlenie iónu, $\mathbf{a} = \frac{e\mathbf{E}}{m}$
$\mathbf{r}, x, y, z$	polohový vektor a jeho zložky v karteziánskom súradnicovom systéme
λ	voľná dráha
$\langle\lambda\rangle$	stredná voľná dráha
T, T_g	teplota neutrálneho plynu
v_T	najpravdepodobnejšia tepelná rýchlosť molekuly plynu s teplotou T
$f(v)$	rozdeľovacia funkcia rýchlosti
S	entropia
τ, u, ξ	bezrozmerný čas, rýchlosť a poloha častice
α	pomer medzi energiou iónu dodanou elektrickým poľom na strednej voľnej dráhe a tepelnou energiou iónu
$\mathrm{sgn}(x)$	znamienko premennej x
$\Theta(x)$	Heavisideova skoková funkcia

$$\Theta(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases} \tag{1.1}$$

$\delta(x)$ Diracova delta funkcia

$$\mathrm{sgn}(x) = \begin{cases} -1, & x < 0 \\ 1, & x > 0 \end{cases} \tag{1.2}$$

$\mathrm{erf}(x)$ chybová funkcia

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int\limits_0^x e^{-t^2} dt \tag{1.3}$$

$\mathrm{erfc}(x)$ doplnková funkcia chýb

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int\limits_x^{+\infty} e^{-t^2} dt \tag{1.4}$$

Kapitola 2

Fyzikálna formulácia problému

2.1 Rezonančná výmena náboja

Proces rezonančnej výmeny náboja je interakcia medzi atómom neutrálneho plynu a jeho iónom. Túto reakciu možno zapísať rovnicou

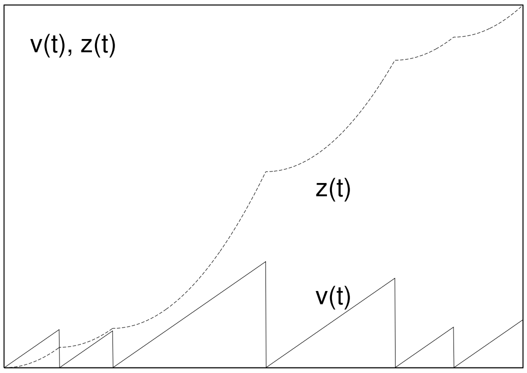
$$A^{\pm} + A(nl) \quad \longrightarrow \quad A(n'l') + A^{\pm}. \tag{2.1}$$

Reakcia je spojená s procesom prechodu valenčného elektrónu cez potenciálovú bariéru v príťažlivom poli susedných častíc. V ideálnom prípade nedochádza k prenosu hybnosti medzi iónom a neutrálnym atómom. Dôvodom je približne rovnaká hmotnosť iónu a neutrálneho atómu, kedže hmotnosť elektrónu možno vzhľadom na hmotnosť atómu zanedbať. Takto prenos náboja z iónu na neutrálny atóm produkuje ión s tepelnou rýchlosťou. V aproximácii studeného plynu možno túto energiu zanedbať. Prípadne možno použiť limitný prechod k silným poliam. Použitím tejto aproximácie možno predpokladať, že po každej zrážke (procese rezonančnej výmeny) ión štartuje z pokoja. Časový priebeh rýchlosti iónu je znázornený na obrázku 2.1.

2.2 Generovanie distribučných funkcií

Výpočet metódou Monte Carlo predpokladá modelovanie náhodného procesu pomocou operácií s náhodnými číslami. Podmienkou úspechu je možnosť náhodný proces mnohonásobne opakovať. K tomu je potrebné mať dostatočný počet náhodných čísel s najrozličnejšími pravdepodobnosťami $F_\xi(x)$ (distribučnými funkciami $f_\xi(x)$). Ako z ďalšieho vyplynie, nie je potrebné zostrojovať špeciálne generátory náhodných čísel pre každý typ distribučnej funkcie. Vystačíme si s generátorom náhodných čísel $\mathsf{R}(0,1)$ s rovnomerným rozdelením na intervale $(0,1)$. Náhodné veličiny s iným typom rozdelenia potom možno získať z $\mathsf{R}(0,1)$ niektorou z uvedených metód.

Náhodné čísla možno získať rozličným spôsobom. Najjednoduchším z generátorov je obyčajná hracia kocka, ktorá poskytuje náhodné číslice $1, 2, \ldots, 6$. Tento generátor je



Obrázok 2.1: Schématické znázornenie priebehu rýchlosti iónu a ubehnutej dráhy v závislosti od času v aproximácii studeného plynu (bezrozmerné jednotky).

však pri numerických výpočtoch robených počítačom nepoužiteľný. Ako použiteľný by mohol byť fyzikálny generátor pripojený k počítaču, ktorý by v okamihu potreby generoval náhodné číslo. Tento typ generátora je založený na analýze náhodných fyzikálnych procesov, ako je radioaktívny rozpad, šum elektronických prvkov a pod. Tento typ generátora predstavuje externé zariadenie pripojené k počítaču a nie je preto bežne používaný.

Dominujúcim typom generátora náhodných čísel je v dnešnej dobe typ založený na aritmetických procedúrach. Náhodné čísla sa vytvárajú pomocou špeciálnych rekurentných vzorcov. V pravom zmysle slova sa preto nejedná o náhodné čísla, lebo sú generované prísne deterministickým spôsobom. Takto generované čísla sa nazývajú pseudonáhodné čísla. Tieto náhodné čísla možno získavať priamo počítačom dostatočne rýchlo a v dostatočnom počte. Nie je podstatné, akým spôsobom náhodné (pseudonáhodné) čísla získame, ale nesmierne dôležité je, aby čísla vyhovovali štatistickým testom.

V tejto časti popíšeme základné spôsoby generovania náhodných čísel, ktoré boli použité pri implementácii metódy Monte Carlo. Jedná sa o metódu inverznej funkcie (kapitola 2.2.1) a von Neumannovu metódu (kapitola 2.2.2).

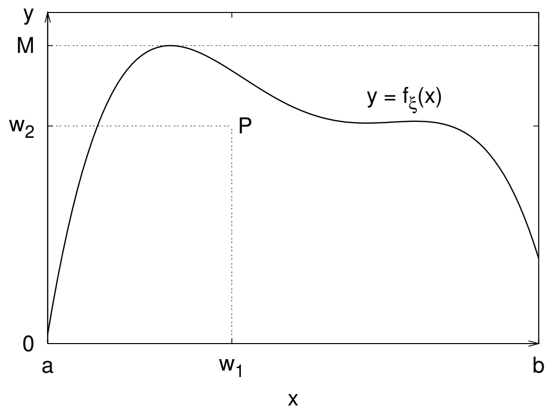
2.2.1 Metóda inverznej funkcie

Táto metóda generovania náhodných čísel je založená na nasledujúcej vete:

Ak náhodná veličina ξ má distribučnú funkciu $f_\xi(x)$, potom rozdelenie náhodnej veličiny

$$\eta = \int_{-\infty}^{\xi} f_\xi(x)dx \tag{2.2}$$

je rovnomerné na intervale $(0, 1)$.



Obrázok 2.2: Generovanie náhodných čísel von Neumannovou metódou výberu.

Transformácia náhodných čísel R_i rovnomerne rozložených na intervale $(0,1)$ na náhodné čísla x_i s daným rozložením $f_\xi(x)$ sa potom dostáva riešením rovnice

$$\int\limits_{-\infty}^{x_i} f_\xi(x)dx = R_i \tag{2.3}$$

Táto metóda sa používa na generovania náhodných čísel s pravdepodobnosťou $F_\xi(x)$ v tých prípadoch, keď vieme $F_\xi^{-1}(y)$ jednoducho vypočítať.

2.2.2 Neumannova metóda

Táto metóda pochádza od J. von Neumanna a niekedy sa táto metóda nazýva aj metódou výberu. Používa sa najmä v prípadoch, kedy nemožno analyticky vyjadriť inverznú funkciu $F_\xi^{-1}(y)$.

Nech distribučná funkcia $f_\xi(x)$ náhodnej veličiny ξ je ohraničená na intervale (a,b). Označme $M = \sup\{f_\xi(x), a < x < b\}$. Vygenerovanie náhodnej hodnoty veličiny ξ potom prebieha následovne (viď. obrázok 2.2):

1. Vygenerujeme dve hodnoty R_1 a R_2 náhodnej veličiny η z $\mathsf{R}(0,1)$
2. Vytvoríme čísla $w_1 = a + R_1(b-a)$, tzn. w_1 z $\mathsf{R}(a,b)$
   $w_2 = MR_2$, tzn. w_2 z $\mathsf{R}(0,M)$
3. Ak bod P so súradnicami (w_1, w_2) bude ležať pod krivkou $y = f_\xi(x)$, teda $w_2 \leq f_\xi(w_1)$, potom za náhodné číslo vezmeme $\xi = w_1$. Ak podmienka nebude splnená, generujeme novú dvojicu (R_1, R_2) a prejdeme k bodu 1.

2.3 Jednorozmerný model driftu nabitých častíc v plyne

Najjednoduchším prípadom, v ktorom je možné použiť metódu Monte Carlo, je štúdium oscilácií driftovej rýchlosti iónov v aproximácii studeného plynu s uvážením rezonančnej výmeny náboja a konštantnej strednej voľnej dráhy (kapitola 2.3.1). V tomto prípade možno teoreticky odvodiť časovú závislosť driftovej rýchlosti, čo umožňuje porovnať nasimulované výsledky s teoretickým priebehom.

V prípade, že začneme brať do úvahy tepelný pohyb molekúl neutrálneho plynu, situácia sa mierne skomplikuje. Tento prípad je teoreticky študovaný v kapitole 2.3.2. Zatiaľ čo v modeli bez tepelného pohybu možno analyticky odvodiť časový priebeh driftovej rýchlosti a zaviesť bezrozmerné parametre, v prípade s tepelným pohybom to už možné nie je. Zavedenie bezrozmerných parametrov, ktoré umožňuje získať zákony podobnosti, je v prípade tepelného pohybu zložitejšie.

2.3.1 Rezonančná výmena náboja v aproximácii studeného plynu

Teoretický model rezonančnej výmeny náboja v aproximácii studeného plynu je založený na nasledujúcich predpokladoch:

- atómy (molekuly) neutrálneho plynu sa nepohybujú
- koncentrácia iónov v neutrálnom plyne je malá
- po zrážke iónu s neutrálnou molekulou má ión nulovú rýchlosť (symetrická nábojová výmena)
- nábojová výmena nastáva okamžite
- elektrické pole, koncentrácia molekúl neutrálneho plynu a iónov je nezávislá od polohy

Podľa predpokladov sú ióny v neutrálnom plyne iba prímes. To umožňuje vzhľadom na ich nízku koncetráciu zanedbať zrážky medzi iónmi a taktiež predpokladať, že distribučná funkcia iónov spätne neovplyvňuje distribučnú funkciu molekúl neutrálneho plynu. Ďalej je podľa predpokladov elektrické pole a koncentrácia plynu a iónov nezávislá od polohy, teda aj distribučná funkcia taktiež od polohy nezávisí. Boltzmannova rovnica sa potom redukuje na tvar

$$\frac{\partial f(v,t)}{\partial t} + a(t)\frac{\partial f(v,t)}{\partial v} = -n|v|\sigma f(v,t) + \frac{n}{V}\delta\left(\frac{v}{V}\right)\int_{-\infty}^{+\infty} |v'|\sigma f(v',t)dv', \tag{2.4}$$

kde n je koncentrácia neutrálneho plynu, v rýchlosť iónu, σ účinný prierez (predpokladáme, že nezávisí od rýchlosti) a $f(v,t)$ je iónová distribučná funkcia. Pravá strana rovnice popisuje rezonančnú výmenu náboja. Aby bol zachovaný bezrozmerný charakter Diracovej delta funkcie a jej argumentu, musíme zaviesť konštantu V, ktorá má rozmer rýchlosti.

Rovnicu (2.4) možno normalizovať zavedením týchto substitúcií:

$$\tau = t\sqrt{n\sigma a}, \qquad u = v/V, \qquad V = \sqrt{a/n\sigma}. \tag{2.5}$$

Po normalizácii dostaneme Boltzmannovu rovnicu v nasledujúcom tvare:

$$\frac{\partial\varphi(u,\tau)}{\partial\tau} + \frac{\partial\varphi(u,\tau)}{\partial u} = -|u|\varphi(u,\tau) + \delta(u)\int\limits_{-\infty}^{+\infty}|u'|\varphi(u',\tau)du', \tag{2.6}$$

kde $\varphi(u,\tau) = \frac{f(v,t)}{n_+}\frac{dv}{du}$ je bezrozmerná distribučná funkcia iónov normalizovaná na jednotku:

$$\int\limits_{-\infty}^{+\infty}\varphi(u,\tau)du = 1. \tag{2.7}$$

Analytické riešenie Boltzmannovej rovnice

Zavedením funkcie

$$\Psi(\tau) = \int\limits_{-\infty}^{+\infty}|u|\varphi(u,\tau)du = \int\limits_{0}^{+\infty}u[\varphi(u,\tau)+\varphi(-u,\tau)]du \tag{2.8}$$

možno rovnicu (2.6) previesť na tvar

$$\frac{\partial\varphi}{\partial\tau} + \frac{\partial\varphi}{\partial u} = -|u|\varphi + \delta(u)\Psi(\tau). \tag{2.9}$$

Časová závislosť driftovej rýchlosti iónov je daná vzťahom

$$u_d(\tau) = \int\limits_{-\infty}^{+\infty}u\varphi(u,\tau)du \tag{2.10}$$

a rovnica (2.8) nadobudne tvar

$$\Psi(\tau) = u_d(\tau) + 2\int\limits_{0}^{+\infty}u\varphi(-u,\tau)du \tag{2.11}$$

Parciálnu diferenciálnu rovnicu (2.9) možno previesť na systém obyčajných diferenciálnych rovníc [6]:

$$\frac{d\tau}{1} = \frac{du}{1} = \frac{d\varphi}{-|u|\varphi + \delta(u)\Psi(\tau)} \tag{2.12}$$

a po úpravách dostaneme

$$u = \tau + C_1 \tag{2.13}$$

$$\frac{d\varphi}{du} = -|u|\varphi + \delta(u)\Psi(u - C_1), \tag{2.14}$$

pričom sme už do vzťahu (2.14) za τ dosadili predchádzajúci vzťah (2.13). Riešenie nehomogénnej diferenciálnej rovnice (2.14) hľadáme v tvare $\varphi(u) = F(u)G(u)$. To vedie k sústave dvoch obyčajných diferenciálnych rovníc

$$\frac{dF}{du} + |u|F = 0 \tag{2.15}$$

$$F\frac{dG}{du} = \delta(u)\Psi(u - C_1). \tag{2.16}$$

Riešenie rovníc (2.15) a (2.16) je

$$F(u) = Ce^{\int_0^u -|x|dx} = Ce^{-\frac{1}{2}u^2 \mathrm{sgn}\, u} \tag{2.17}$$

$$dG = \frac{1}{F(u)}\delta(u)\Psi(u - C_1)du$$

$$G(u) = \frac{1}{C}\int_0^u e^{\frac{1}{2}y^2 \mathrm{sgn}\, y}\delta(y)\Psi(y - C_1)dy + K =$$

$$= \begin{cases} \frac{1}{C}\Psi(-C_1) + K, & \mathrm{u} > 0 \\ 0 + K, & \mathrm{u} < 0 \end{cases} \tag{2.18}$$

$$G(u) = \frac{1}{C}\Theta(u)\Psi(-C_1) + K \tag{2.19}$$

Teda riešenie rovnice (2.14) vyzerá

$$\varphi(u) = F(u)G(u) = [\Theta(u)\Psi(-C_1) + C_2]e^{-\frac{1}{2}u^2 \mathrm{sgn}\, u}. \tag{2.20}$$

$\Theta(x)$ je Heavisideova skoková funkcia definovaná vzťahom (1.1). Konštantu C_1 vyjadríme zo vzťahu (2.13). Z vety o implicitnej funkcii však vyplýva, že konštanta C_2 musí byť nejakou funkciou konštanty C_1. Označme túto funkciu ako $\Phi, C_2 = \Phi(C_1) = \Phi(u - \tau)$. Po dosadení máme

$$\varphi(u, \tau) = [\Theta(u)\Psi(\tau - u) + \Phi(u - \tau)]e^{-\frac{1}{2}u^2 \mathrm{sgn}\, u}. \tag{2.21}$$

Funkciu $\Phi(x)$ určíme z počiatočnej distribučnej funkcie $\varphi(u, 0)$. Z poslednej rovnice ju môžeme vyjadriť ako

$$\Phi(x) = \varphi(x, 0)e^{\frac{1}{2}x^2 \mathrm{sgn}\, x} - \Theta(x)\Psi(-x) \tag{2.22}$$

Spätné dosadenie funkcie $\Phi(x)$ do všeobecného riešenia nám dáva riešenie iba pre $\tau \geq 0$. Dôvodom je nevratnosť skúmaného deja a nami stanovená počiatočná podmienka $\varphi(u, 0)$. Naše riešenie sa preto môže odvíjať iba od tejto počiatočnej podmienky.

$$\varphi(u, \tau) = [\Theta(u) - \Theta(u - \tau)]\Psi(\tau - u)e^{-\frac{1}{2}u^2 \mathrm{sgn}\, u} + \\ + \varphi(u - \tau, 0)e^{\frac{1}{2}(u-\tau)^2 \mathrm{sgn}\,(u-\tau) - \frac{1}{2}u^2 \mathrm{sgn}\, u}. \tag{2.23}$$

Predpokladajme teraz, že všetky ióny sa v čase $\tau = 0$ pohybujú v kladnom smere alebo sú v pokoji. Potom pre distribučnú funkciu $\varphi(u, \tau)$ platí

$$\varphi(u, 0) = 0, \qquad u < 0 \tag{2.24}$$

a vzťah (2.23) možno upraviť na tvar

$$\varphi(u,\tau) = \begin{cases} [1-\Theta(u-\tau)]\Psi(\tau-u)e^{-\frac{1}{2}u^2} + \varphi(u-\tau,0)e^{\tau(\frac{1}{2}\tau-u)}, & u \geq 0 \\ 0, & u < 0 \end{cases} \tag{2.25}$$

Podľa vzťahu (2.11) potom pre funkciu $\Psi(x)$ platí $\Psi(x) = u_d(x)$. Dosadením distribučnej funkcie (2.25) do integrálnej rovnice pre driftovú rýchlosť (2.10) dostaneme vyjadrenie

$$u_d(\tau) = \int_0^\tau u u_d(\tau-u)e^{-\frac{1}{2}u^2}du + \int_0^\infty u\varphi(u-\tau,0)e^{\tau(\frac{1}{2}\tau-u)}du. \tag{2.26}$$

Ustálená hodnota driftovej rýchlosti a rovnovážna distribučná funkcia

Z normovacej podmienky (2.7) a rovnice pre distribučnú funkciu (2.25) možno s využitím rovnice $\Psi(x) = u_d(x)$ vypočítať ustálenú hodnotu driftovej rýchlosti v čase $\tau = \infty$. Integráciou rovnice (2.25) a na základe vlastností Heavisideovej funkcie (1.1) dostaneme

$$1 = \int_0^\tau u_d(\tau-u)e^{-\frac{1}{2}u^2}du + \int_\tau^\infty \varphi(u-\tau,0)e^{\tau(\frac{1}{2}\tau-u)}du \tag{2.27}$$

V prípade, že v čase $\tau = 0$ sú ióny v pokoji (majú nulové rýchlosti), je distribučná funkcia $\varphi(u,0)$ delta funkciou. Integráciou druhého člena a prechodom k limite $\lim_{\tau\to\infty}$ dostaneme vzťah

$$1 = u_d(\infty)\int_0^\infty e^{-\frac{1}{2}u^2}du + \lim_{\tau\to\infty} e^{-\frac{1}{2}\tau^2}, \tag{2.28}$$

na základe ktorého pre ustálenú hodnotu driftovej rýchlosti vychádza rovnica

$$u_d(\infty) = \sqrt{\frac{2}{\pi}}. \tag{2.29}$$

Spätným dosadením do vzťahu (2.25) možno dostať rovnovážnu distribučnú funkciu

$$\varphi(u,\infty) = \sqrt{\frac{2}{\pi}}\Theta(u)e^{-\frac{1}{2}u^2}. \tag{2.30}$$

Relaxácia monoenergetického zväzku

Vzťah (2.25) umožňuje taktiež určiť časový priebeh driftovej rýchlosti v prípade, že sa všetky ióny hýbu v smere elekrického poľa spoločnou rýchlosťou u_0 a v čase $\tau = 0$ je elektrické pole zapnuté. V tomto prípade je potom počiatočná distribučná funkcia daná vzťahom

$$\varphi(u,0) = \delta(u-u_0), \tag{2.31}$$

teda rovnicu (2.25) možno pre $u \geq 0, \tau \geq 0$ zapísať ako

$$\varphi(u,\tau) = [1 - \Theta(u-\tau)]u_d(\tau - u)e^{-\frac{1}{2}u^2} + \delta(u - \tau - u_0)e^{\frac{1}{2}u_0^2}e^{-\frac{1}{2}(u_0+\tau)^2}. \tag{2.32}$$

Na základe vzťahu (2.26) dostaneme pre časový priebeh driftovej rýchlosti vzťah

$$u_d(\tau) = \int_0^\tau u u_d(\tau - u)e^{-\frac{1}{2}u^2}du + (\tau + u_0)e^{\frac{1}{2}u_0^2}e^{-\frac{1}{2}(u_0+\tau)^2}. \tag{2.33}$$

Výpočet času medzi dvomi zrážkami

V aproximácii studeného plynu sa molekuly neutrálneho plynu nepohybujú. Ak na ión silovo pôsobí iba urýchľujúce homogénne pole, ión sa pohybuje rovnomerne zrýchlene s nulovou počiatočnou rýchlosťou (uvažujeme rezonančnú výmenu náboja a tak v dôsledku rovnakej hmotnosti iónu a molekuly neutrálneho plynu dôjde k výmene rýchlostí medzi iónom a neutrálnou molekulou). Potom možno dobu medzi dvomi zrážkami rátať priamo podľa vzťahu

$$t = \sqrt{\frac{2\lambda}{a}} \tag{2.34}$$

kde a je zrýchlenie pôsobiace na nabitú časticu a λ je voľná dráha.

2.3.2 Rezonančná výmena náboja s uvážením tepelného pohybu molekúl neutrálneho plynu

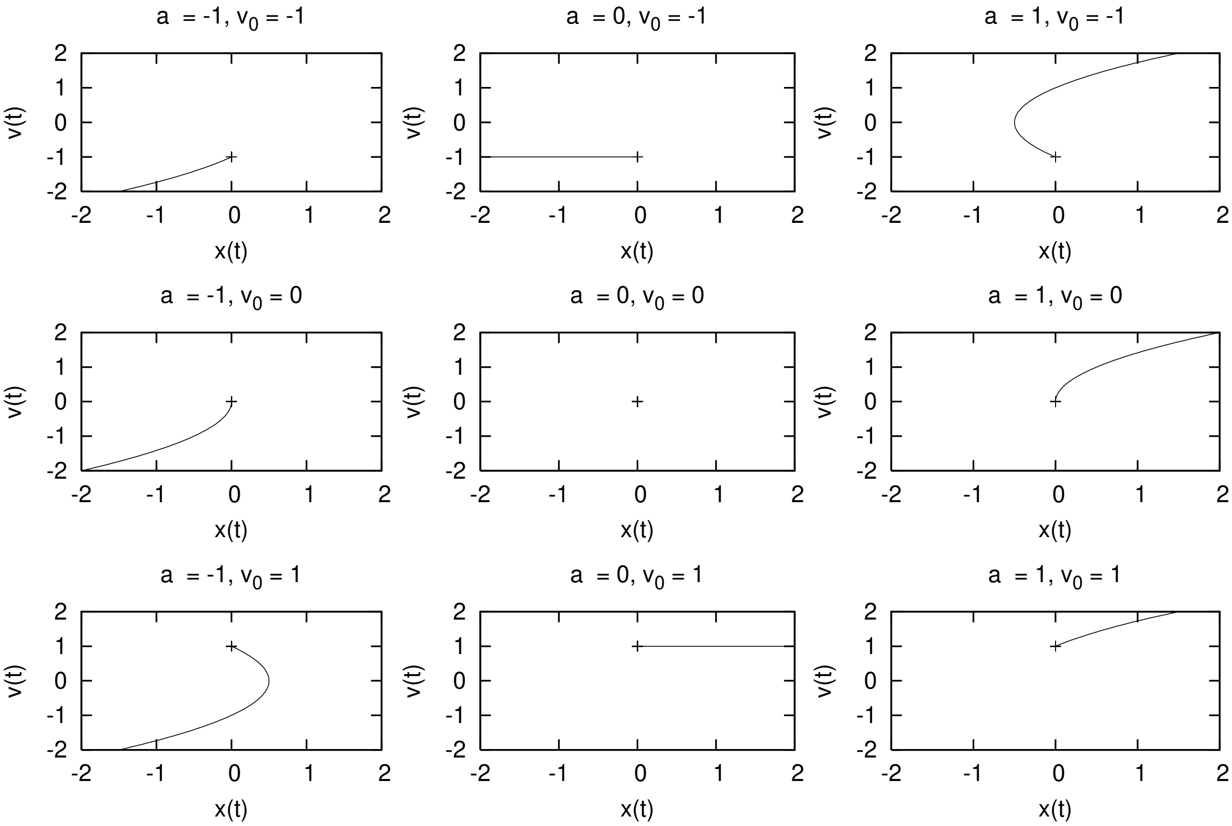
Štúdium tohto javu vychádza z predošlých úvah a platia predpoklady uvedené v kapitole 2.3.1. Tentoraz sa však nepredpokladá, že molekuly neutrálneho plynu sú v pokoji (nepohybujú sa), ale že sa pohybujú tepelnými rýchlosťami. Preto po zrážke iónu s molekulou neutrálneho plynu ión získa tepelnú rýchlosť danej molekuly. Pre plyn s teplotou T_g je rýchlostné rozdelenie molekúl neutrálneho plynu dané Maxwellovým rozdelením (jednorozmerný prípad)

$$f(v_x)dv_x = \frac{1}{\sqrt{\pi}v_T}\exp\left(-\frac{v_x^2}{v_T^2}\right)dv_x, \tag{2.35}$$

kde $v_T = \sqrt{\frac{2kT_g}{m_g}}$ je najpravdepodobnejšia tepelná rýchlosť molekuly neutrálneho plynu, k Boltzmannova konštanta, T_g teplota neutrálneho plynu a m_g hmotnosť molekuly neutrálneho plynu.

Pri uvážení tepelného pohybu musíme čas medzi dvomi zrážkami t rátať z kvadratickej rovnice pre rovnomerne zrýchlený pohyb. Fyzikálna predstava je nasledovná: predpokladáme, že na ión pôsobí konštantné zrýchlenie a. Ión štartuje s počiatočnou náhodnou tepelnou rýchlosťou v_0 (generovanou z rozdelenia podľa vzťahu (2.35)). Voľnú dráhu λ prejde ión rovnomerne zrýchlene (spomalene) za čas t. Poloha, v ktorej sa ión bude nachádzať v čase t, je daná vzťahom

$$x = \frac{1}{2}at^2 + v_0 t. \tag{2.36}$$



Obrázok 2.3: Rýchlosť iónu v závislosti od jeho polohy pre rôzne hodnoty počiatočnej rýchlosti a zrýchlenia

Tento vzťah je nutné analyzovať podrobnejšie. Ako si možno všimnúť na obrázku 2.3, pri vhodne „zvolených“[1] parametroch λ, a a v_0 môže nastať prípad, že ión zmení smer letu skôr, ako urazí dráhu λ – voľnú dráhu. V literatúre sa tento prípad nerieši a autori sa snažia vzniknutú situáciu obísť. Keďže našou snahou je navrhnúť program, ktorý bude datailne simulovať jednotlivé častice, musíme tento problém vyriešiť. Ako riešenie sa ponúka interpretovať voľnú dráhu λ ako absolútnu hodnotu dráhy, ktorú musí ión prejsť, aby došlo k zrážke. Detailné riešenie problému je zhrnuté v nasledujúcich bodoch:

$\mathbf{a = 0 \ \wedge\ v_0 = 0}$ – na ióny nepôsobí urýchľujúce pole a počiatočná rýchlosť iónu je nulová. Čas medzi dvomi zrážkami je $t = \infty$, teda ión nebude driftovať.

$(\mathbf{sgn\, a = sgn\, v_0}) \ \wedge\ (\mathbf{a \neq 0 \ \vee\ v_0 \neq 0})$ – rýchlosť a zrýchlenie iónu majú súhlasné znamienka a aspoň jeden z parametrov je nenulový. V tomto prípade možno čas medzi dvomi zrážkami vyrátať ako

$$t = \frac{2\lambda}{|v_0| + \sqrt{v_0^2 + 2|a|\lambda}}. \tag{2.37}$$

[1] pri numerických výpočtoch sa voľná dráha λ a počiatočná rýchlosť iónu v_0 generujú náhodne podľa príslušných distribučných funkcií

sgn a $\neq$ sgn v$_0$ – znamienka rýchlosti a zrýchlenia sú opačné. Ak $v_0 \leq \sqrt{2\lambda|a|}$, na vzdialenosti menšej ako λ nadobudne ión nulovú rýchlosť a vplyvom zrýchlenia sa začne pohybovať opačne. Ión nadobudne nulovú rýchlosť za čas $t_1 = \frac{|v_0|}{|a|}$ a za tento časový úsek preletí dráhu $\lambda_1 = \frac{v_0^2}{2|a|}$. Aby nastala ďalšia zrážka, musí ión preletieť ešte dráhu $\lambda_2 = \lambda - \lambda_1 = \lambda - \frac{v_0^2}{2|a|}$. Tento úsek preletí ión za čas

$$t_2 = \frac{\sqrt{2\lambda|a| - v_0^2}}{|a|}.$$

Celkový čas, ktorý potreboval ión na ubehnutie voľnej dráhy teda je

$$t = t_1 + t_2 = \frac{|v_0| + \sqrt{2|a|\lambda - v_0^2}}{|a|}. \tag{2.38}$$

Naopak, ak platí $|v_0| \geq \sqrt{2\lambda|a|}$, smer pohybu iónu sa na vzdialenosti menšej ako λ nezmení a pre čas medzi dvomi zrážkami platí

$$t = \frac{2\lambda}{|v_0| + \sqrt{v_0^2 - 2|a|\lambda}}. \tag{2.39}$$

Vzťahy (2.37) a (2.39) možno nahradiť spoločným vyjadrením, ktoré bude platiť v obidvoch prípadoch. Pre tieto účely definujme[2] sgn $0 \equiv 1$, čím sa zahrnú aj prípady $a = 0 \ \vee \ v_0 = 0$:

$$t = \frac{2\lambda}{|v_0| + \sqrt{v_0^2 + 2a\lambda \operatorname{sgn} v_0}}. \tag{2.40}$$

Zavedenie bezrozmerných premenných

Ak uvážime tepelný pohyb molekúl neutrálneho plynu, ión po zrážke už nebude štartovať s nulovou počiatočnou rýchlosťou, ale s rýchlosťou v_0, ktorú mala neutrálna molekula tesne pred zrážkou (ión a neutrálna molekula si vzhľadom na rovnaké hmotnosti vymenia rýchlosti). Táto počiatočná rýchlosť je daná Maxwellovským rozdelením. V homogénnom elektrickom poli bude poloha a rýchlosť iónu daná rovnicami

$$x = \frac{1}{2}at^2 + v_0 t \tag{2.41}$$
$$v = at + v_0 \tag{2.42}$$

Na základe týchto rovníc možno zaviesť bezrozmerné parametre. Nech

$$x = K_x \xi \tag{2.43}$$
$$v = K_v u \tag{2.44}$$
$$t = K_t \tau \tag{2.45}$$

[2]korektná matematická definícia je sgn $0 \equiv 0$

kde K_x, K_v, K_t sú normalizačné konštanty pre polohu, rýchlosť a čas a ξ, u, τ sú príslušné bezrozmerné premenné. Potom rovnicu (2.42) možno zapísať ako

$$\xi = \frac{1}{2}\frac{aK_t^2}{K_x}\tau^2 + \frac{K_v K_t}{K_x}u_0\tau \tag{2.46}$$

$$u = \frac{aK_t}{K_v}\tau + u_0 \tag{2.47}$$

Nech normalizačné koeficienty rýchlosti a polohy sú

$$K_v = v_T = \sqrt{\frac{2kT_g}{m_g}} \qquad K_x = \langle\lambda\rangle, \tag{2.48}$$

kde k je Boltzmannova konštanta, T_g teplota neutrálneho plynu, m_g hmotnosť neutrálnej molekuly a $\langle\lambda\rangle$ stredná voľná dráha. Položme

$$\frac{K_v K_t}{K_x} = 1 \tag{2.49}$$

a označme ako parameter α súbor konštánt

$$\alpha = \frac{aK_t^2}{K_x} = \frac{a\langle\lambda\rangle}{v_T^2}. \tag{2.50}$$

Ľahko možno nahliadnuť, že parameter α udáva pomer medzi energiou iónu dodanou elektrickým poľom na strednej voľnej dráhe a tepelnou energiou iónu. Koeficient α teda určuje, či sa jedná o slabé ($\alpha < 1$) alebo o silné ($\alpha > 1$) pole.

Rovnice (2.46) a (2.47) po úprave vyzerajú nasledovne:

$$\xi = \frac{1}{2}\alpha\tau^2 + u_0\tau \tag{2.51}$$

$$u = \alpha\tau + u_0 \tag{2.52}$$

Kapitola 3

Počítačová simulácia driftu

3.1 Hardverové a softvérové vybavenie

Keďže hardvérové a softvérové vybavenie vzhľadom na zameranie tejto práce predstavuje experimentálnu aparatúru, považujem za dôležité spomenúť aspoň hlavné prvky.

Celá diplomová práca bola vytvorená za pomoci otvoreného softvéru šíreného pod licenciou GNU GPL [10]. Použitý operačný systém bol GNU/Linux Debian 3.0 [11]. Zdrojové kódy programov a iných textov boli napísané v textovom editore Vim [12]. Programy v jazyku C boli kompilované kompilátorom gcc-2.95 [13] a odladené debuggerom gdb-5.3 [14]. Výpočty prebiehali na počítači s procesorom Athlon 1.2 GHz a operačnou pamäťou RAM veľkosti 256 MB.

Časová náročnosť výpočtu pre jedno nastavenie vstupných parametrov (časový priebeh driftovej rýchlosti, distribučnej funkcie a pod.) sa pohybovala z rozmedzí 1 minúta až niekoľko hodín v závislosti od nastavenia vstupných parametrov. Veľkosť operačnej pamäte RAM nemala v tomto prípade vplyv na rýchlosť výpočtu, lebo simulačný program používal iba minimálnu časť pamäte (< 1MB). Simulácia najrýchlejšie prebiehala v prípade $T = 0$ K. Spomalenie výpočtu pri vyššej teplote bolo spôsobené potrebou viackrát volať generátor náhodných čísel na vygenerovanie náhodnej tepelnej rýchlosti a taktiež skracovaním doby medzi dvomi zrážkami.

3.2 Implementácia metódy Monte Carlo

3.2.1 Štruktúra programu

Keďže som sa snažil spraviť simulačný program čo najviac flexibilným, odrazilo sa to na relatívnej zložitosti programu. Základné vlastnosti teraz spomeniem.

Aby bolo možné zadávať vstupné parametre do výpočtu a spúšťať program dávkovo (tzn. bez ďalšej potrebnej obsluhy), použil som externú knižnicu `libcfg+` [15], [16], ktorá umožňuje načítať vstupné parametre z príkazového riadka (alebo konfiguračného súboru) a naplniť nimi príslušné premenné. Na vývoji tejto knižnice som sa čiastočne podieľal. Spôsob použitia programu a zoznam akceptovaných parametrov možno získať spustením programu s parametrom `./hotGas --help` alebo `./hotGas help`.

Po úspešnom odovzdaní parametrov do programu a výpočte hodnôt základných premenných sa kvôli informácii užívateľa vypíše zoznam všetkých dôležitých parametrov. To umožňuje kontrolu a neskoršiu orientáciu v dátových súboroch. Všetky údaje sú vypisované na obrazovku. Dátové súbory pri neinteraktívnom spúšťaní výpočtu sa vytvárali presmerovaním výstupu do súboru s príslušným názvom.

Odladenie algoritmu výpočtu si vyžiadalo pomerne veľké množstvo riadkov zdrojového kódu. Aby bola možná aj neskoršia kontrola algoritmu, rozhodol som sa ponechať tieto časti kódu. Všetky časti týkajúce sa ladenia podliehajú podmienenej kompilácii. Preto pri obyčajnej kompilácii tieto inštrukcie do výsledného strojového kódu zahrnuté nie sú, takže nedochádza k spomaľovaniu behu programu a program beží, ako keby sa tieto riadky ani nevyskytovali v zdrojovom kóde. Naopak zapnutie režimu ladenia sa udeje zadefinovaním symbolickej konštatny `DEBUG` v kompilátore.

Keďže sa ukázalo, že výpočet driftovej rýchlosti a distribučných funkcií možno robiť v podstate tým istým algoritmom, rozhodol som sa použiť podmienený preklad aj v tomto prípade. Výhodu to má opäť v tom, že ak je požadovaný iba výpočet časového priebehu driftovej rýchlosti, nie je spotrebovávaný výpočtový výkon na výpočet distribučnej funkcie a naopak. Preto ak nie je definovaná konštanta `DISTRIB`, výsledný skompilovaný program ráta časový priebeh driftovej rýchlosti. Ak symbolická konštanta `DISTRIB` definovaná je, skompilovaný program ráta časový priebeh distribučnej funkcie.

3.2.2 Popis algoritmov

Drift skupiny iónov v plyne, teoreticky študovaný v kapitole 2.3.1 vyzerá nasledovne: skupina iónov odštartuje z presne udanej polohy. V každom časovom okamihu sa časť iónov pôvodného zväzku zrazí. Pri zrážke sa mení rýchlosť iónu. Tým dochádza k rozptylu pôvodného zväzku a pohyb iónov sa mení z usmerneného na chaotický. Distribučná funkcia iónov sa postupne vyvíja v čase, až nadobudne ustálený tvar.

Simulácia deja, kde by sa naraz simulovala dostatočne veľká skupina iónov (presnosť metódy Monte Carlo od toho závisí) by vyžadovala mimoriadne veľké nároky na operačnú pamäť, pretože by bolo potrebné uchovávať informáciu o polohe a rýchlosti každého iónu z predchádzajúceho kroku.

Možno však využiť predpoklad 2 z kapitoly 2.3.1. Keďže ióny navzájom medzi sebou neinteragujú (vzhľadom na ich nízku koncentráciu v plyne), možno simuláciu robiť postupne. To znamená, že nebudeme simulovať naraz celú skupinu iónov, ale postupne necháme driftovať jeden ión, pričom iba zaznamenávame jeho príspevok k distribučnej funkcii alebo driftovej rýchlosti. Tým odpadne potreba pamätať si parametre každého iónu z predchádzajúceho kroku, čím sa podstatne znížia pamäťové nároky. Uchovávať sa bude iba časový priebeh požadovaných výstupných parametrov. Takto sa výpočet s jedným iónom opakuje veľakrát, až získame dostatočne veľkú skupinu iónov.

Keďže numerický výpočet prebieha v diskrétnom priestore, musíme príspevok každého iónu započítavať v presne určených okamihoch – bodoch. Uvažujeme homogénne pole, preto sa ión v čase medzi dvomi zrážkami pohybuje rovnomerne zrýchleným (spomaleným) pohybom. Doba medzi dvomi zrážkami však nie je konštantná. Závisí od toho, akú dráhu ión prekoná bez toho, aby sa zrazil, čo je náhodná veličina. Voľná dráha, ktorú urazí ión do ďalšej zrážky, je generovaná metódou inverznej funkcie (kapitola 2.2.1) s rozdelením

$$f(x) = \frac{1}{\langle\lambda\rangle} e^{-\frac{x}{\langle\lambda\rangle}}. \tag{3.1}$$

Príslušná rozdeľovacia funkcia $F(x)$ je

$$F(x) = \int_0^x \frac{1}{\langle\lambda\rangle} e^{-\frac{\xi}{\langle\lambda\rangle}} d\xi = 1 - e^{-\frac{x}{\langle\lambda\rangle}} \tag{3.2}$$

a inverzná funkcia

$$F^{-1}(y) = -\langle\lambda\rangle \ln(1-y) \tag{3.3}$$

Tento vzťah možno pre potreby výpočtu upraviť využitím vlastnosti, že číslo y bude mať rovnaké rozdelenie ako číslo $(1-y)$, pretože $y \in \mathsf{R}(0,1)$. Tým sa dosiahne maličké zrýchlenie programu. Na základe tejto vlastnosti možno písať

$$F^{-1}(y) = -\langle\lambda\rangle \ln(y). \tag{3.4}$$

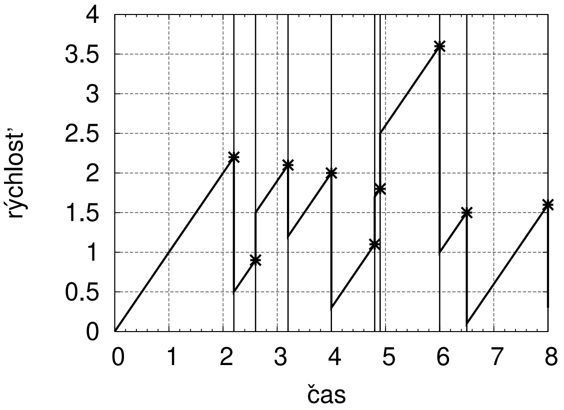
V závislosti od toho, aká voľná dráha sa vygeneruje a aký časový krok `d_time` zvolíme (zadáme ako parameter do programu), môže ión prebehnúť niekoľko časových buniek bez zrážky. Alebo naopak, ak sú časy medzi dvomi zrážkami menšie ako časový krok, nastane zrážok niekoľko v jednej časovej bunke. S týmto náš program ráta. Preto je hlavný výpočtový algoritmus rozdelený do dvoch častí.

Schéma použitá pri odladení algoritmu nachádzajúceho sa v prílohe A na riadkoch 505–[LINE a] – 606–[LINE b] (strana 43) je na obrázku 3.1. Navrhol som časy, kedy dôjde k zrážke tak, aby boli pokryté všetky prípady:

- čas medzi dvomi zrážkami je väčší ako časový krok
- čas medzi dvomi zrážkami je menší ako časový krok
- zrážka nastane presne v bode, kedy je nutné zaznamenať parametre iónu.

Body, v ktorých som navrhol, že dôjde k zrážke sú označené hviezdičkou. Pri ladení algoritmu som predpokladal, že časový krok `d_time` je 1.0 a zrážky nastanú postupne v časoch $2.2, 2.6, 3.2, 4.0, 4.8, 4.9, 6.0, 6.5, 8.0$ (jednotky momentálne nie sú dôležité). V závislosti od toho, koľko času zostáva do nasledujúcej zrážky, beží prvá alebo druhá časť algoritmu (riadky 526–[LINE c] až 579–[LINE d] a 586–[LINE e] až 599–[LINE f]).

Prvá časť algoritmu pracuje vtedy, ak čas zostávajúci do zrážky `rand_time` je väčší ako časový krok výpočtu `d_time`. Vtedy sa vypočítajú nové parametre iónu a zaznamenajú



Obrázok 3.1: Schéma použitá pri implementácii teoretického modelu. Schématické znázornenie priebehu rýchlosti v závislosti od času. Body, keď nastala zrážka, sú označené hviezdičkou.

sa. Tým je zabezpečené, že príspevok každého z iónov je zarátaný v správnom čase. Pritom pri každom prebehnutí slučky `while()` sa dekrementuje čas do zostávajúcej zrážky a inkrementuje reálny čas `real_time`. Takže ladenie algoritmu vyzeralo tak, že sa postupne zaznamenali príspevky v časoch $0.0, 1.0, 2.0$. Do druhej časti algoritmu sa vstúpi zo zostávajúcim časom do zrážky `time_left` 0.2.

Druhá časť algoritmu pracuje vtedy, ak čas zostávajúci do nasledujúcej zrážky `time_left` je menší ako časový krok `d_time`. Ión sa nechá ešte urýchliť zostávajúci čas do zrážky. V tomto bode dôjde k zrážke. Preto sa vygeneruje náhodná tepelná rýchlosť zavolaním funkcie `random_thermal_speed()` (riadok 241–[LINE i]). Táto funkcia v prípade nulovej teploty vráti okamžite nulovú rýchlosť, v prípade nenulovej teploty vráti náhodnú rýchlosť s distribučnou funkciou danou vzťahom (2.35). Na vygenerovanie náhodnej rýchlosti sa použije von Neumannova metóda výberu (kapitola 2.2.2). Z tejto počiatočnej rýchlosti je ión opäť urýchľovaný na vzdialenosti `rand_x`. Podľa vzťahu (2.40) sa vypočíta čas do nastávajúcej zrážky. Ak má nasledujúca zrážka nastať pred ďalším časovým krokom, postup sa opakuje. Pri ladení programu to potom vyzeralo tak, že sa z prvej časti algoritmu použijte `time_left` $= 0.2$. Vygenerovala sa „náhodná“ tepelná rýchlosť `v` $= 0.5$ a čas do nasledujúcej zrážky `rand_time` $= 0.4$. Keďže zrážka nastane v tej istej časovej bunke, opakuje sa druhá časť algoritmu. Ďalším „náhodným“ časom bude `rand_time` $= 0.6$. Vtedy už podmienka na riadku 599–[LINE f] splnená nebude. Preto sa pokračuje riadkom 604–[LINE g], kde sa upraví hodnota `time_left` na hodnotu `time_left` $= 3.0 - 2.6 = 0.4$, pričom sa hodnota `rand_time` nezmení. Takže v prvej časti algoritmu sa na základe `time_left` správne vypočíta poloha a rýchlosť iónu a neskôr na riadku 573–[LINE h] správne upraví hodnota `rand_time`.

3.3 Porovnanie výsledkov

Monte Carlo simulácia driftu iónov v plyne pod účinkom elektrického poľa je veľmi detailná. V princípe možno zaznamenať polohu a rýchlosť každého iónu v ľubovoľnom časovom okamihu. Avšak takáto detailná informácia nie je väčšinou potrebná. Pri veľkom súbore častíc nedáva globálny prehľad. Preto je výhodnejšie sledovať makroskopické parametre ako sú driftová rýchlosť, teplota a pod., ktoré možno aj experimentálne namerať.

Počítačovou simuláciou deja bola sledovaná driftová rýchlosť iónov a rýchlostná distribučná funkcia. Driftovú rýchlosť možno určovať dvomi spôsobmi. Prvý spôsob je určovanie driftovej rýchlosti z ubehnutej dráhy. Tento spôsob nie je natoľko detailný a časový priebeh driftovej rýchlosti nevykazuje oscilácie. Druhý spôsob je detailnejší, pretože určuje driftovú rýchlosť iónov priamo z aktuálnych rýchlosti iónov. Ako možno vidieť neskôr, takáto driftová rýchlosť vykazuje oscilácie.

3.3.1 Driftová rýchlosť a distribučná funkcia v aproximácii studeného plynu

V prípade aproximácie studeného plynu možno porovnať analytické riešenie s výsledkami dosiahnutými metódou Monte Carlo. Drift iónov nastáva vzhľadom na nulovú teplotu neutrálneho plynu v silnom elektrickom poli. Za takýchto podmienok vykazuje driftová rýchlosť oscilácie.

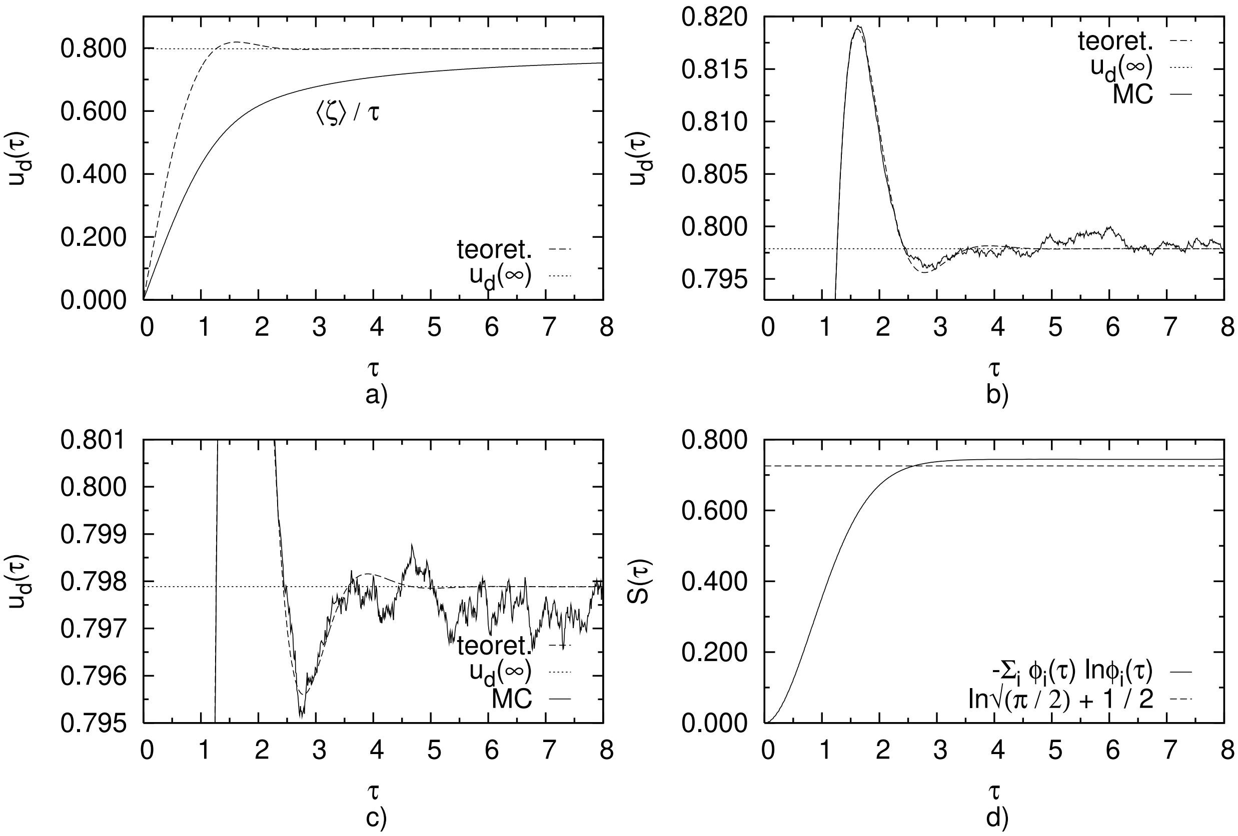
Predpokladajme, že všetky ióny sú v čase $t = 0$ v bode $x = 0$ a majú nulové rýchlosti. Potom vo vzťahu (2.26) pre $\varphi(u-\tau, 0)$ platí

$$\varphi(u, 0) = \delta(u) \tag{3.5}$$

Na základe tohto predpokladu bol vzťah (2.26) numericky zintegrovaný. Použitá bola obyčajná obdĺžnikova metóda. Časový priebeh tohto analytického riešenia je porovnaný s výsledkami dosiahnutými metódou Monte Carlo. Výsledky sú zobrazené na obrázku 3.2-a až 3.2-c. Aby bolo možné výsledky lepšie porovnávať a pozorovať zákony podobnosti, sú škály grafov v bezrozmerných jednotkách. Reálne hodnoty parametrov sa dostanú prenormovaním podľa vzťahu (2.5). V grafe 3.2-a-c je vynesený analytický priebeh a driftová rýchlosť určovaná z prebehnutej dráhy a priamo z rýchlosti iónov. Simulácia bola robená pre $N = 1.0 \times 10^6$ iónov.

Driftová rýchlosť určovaná z prebehnutej dráhy konverguje k ustálenej hodnote driftovej rýchlosti pomalšie. Dôvodom je práve spôsob výpočtu. Driftová rýchlosť je určovaná zo strednej ubehnutej dráhy iónu, čo je integrál rýchlosti cez čas. Keďže počiatočná rýchlosť a poloha iónu sú v aproximácii studeného plynu nulové, musí sa to nutne prejaviť na výslednom priebehu. V čase $\tau > 23.0$ je však už odchýlka takto určenej driftovej rýchlosti menšia ako 2.0% a v čase $\tau > 40.0$ je už menšia ako 1.15%.

Na obrázku 3.2-d je znázornený časový priebeh entropie iónového zväzku. Entropii je v tomto prípade úmerná hodnota veličiny $-\sum_i \varphi_i \ln \varphi_i$, kde φ_i je pravdepodobnosť, že



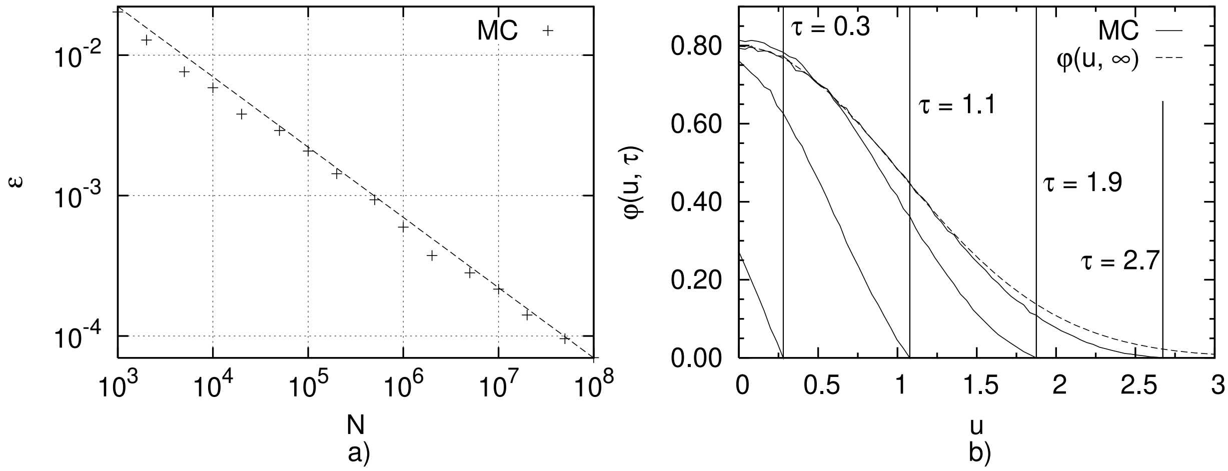
Obrázok 3.2: a,b,c) Časový priebeh driftovej rýchlosti. Krivka označená $\langle\zeta\rangle/\tau$ je driftová rýchlosť určovaná z prebehnutej dráhy, krivka MC je driftová rýchlosť určovaná priamo z rýchlosti iónov. d) Časový priebeh entropie iónového zväzku. Počet simulovaných iónov $N = 1.0 \times 10^6$.

rýchlosť iónu je v intervale $(u_i, u_i + \mathrm{d}u_i)$. Keďže v tomto prípade všetky ióny štartujú s nulovou počiatočnou rýchlosťou a teda sa jedná o dokonale usporiadaný systém, je entropia[1] nulová. Postupne, ako dochádza k zrážkam, entropia rastie. V čase $\tau = 4.0$ nadobúda ustálenú hodnotu $\ln\sqrt{\pi/2} + 1/2$ a ďalej sa už výraznejšie nemení. To naznačuje, že neusporiadanosť systému dosiahla maximum, a teda rýchlosti iónov sú chaotické. Monotónnosť priebehu entropie a jej rast hovorí o nevratnosti fyzikálneho deja [7]. Ustálená hodnota entropie bola určená vzťahom

$$S(\infty) = \lim_{\tau\to\infty} S(\tau) = -\int\limits_{-\infty}^{\infty} \varphi(u,\infty)\ln\varphi(u,\infty)\,\mathrm{d}u = \ln\sqrt{\frac{\pi}{2}} + \frac{1}{2}, \qquad (3.6)$$

kde $\varphi(u,\infty)$ je rovnovážna distribučná funkcia daná vzťahom (2.30). Odchýlka vypočítanej hodnoty entropie je spôsobená numerickou chybou (numericky zistená hodnota bola 0.744 oproti teoretickej hodnote 0.726). Hodnota entropie bola určovaná z výpisu hod-

[1] entropia okrem iného charakterizuje stav neusporiadanosti systému



Obrázok 3.3: a) Závislosť relatívnej chyby ε metódy Monte Carlo od počtu simulovaných iónov. b) Distribučná funkcia rýchlosti iónov pri teplote $T = 0$ K v niekoľkých časových okamihoch. Počet simulovaných iónov $N = 1.0 \times 10^6$.

nôt distribučnej funkcie integráciou jednoduchou obdĺžnikovou metódou. Z fyzikálneho hľadiska je však dôležitá jej monotónnosť a ustálený charakter. Veľkosť odchýlky možno ovplyvniť dostatočným počtom simulácii, veľkosťou rýchlostného intervalu (z technických dôvodov nemožno uvažovať celý rýchlostný priestor) a veľkosťou delenia rýchlostného intervalu.

Ako vyplýva z grafu 3.3-a, relatívna chyba metódy Monte Carlo je pre $N = 1.0 \times 10^6$ iónov[2] menšia ako 1.0×10^{-3}. Závislosť chyby metódy Monte Carlo od počtu simulácii bola urobená zrátaním absolútnej odchýlky driftovej rýchlosti od ustálenej hodnoty $u_d(\infty)$ v časoch $10.0 < \tau < 40.0$, teda kedy je už dej ustálený. Matematicky možno výpočet chyby ε z grafu 3.3-a zapísať ako

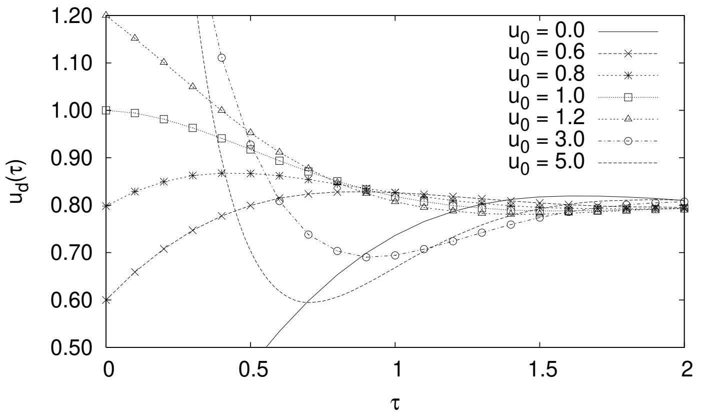
$$\varepsilon = \lim_{\tau \to \infty} \frac{\int\limits_{\tau}^{\tau+\Delta\tau} |u_d(t) - u_d(\infty)| dt}{\Delta\tau \; u_d(\infty)} \tag{3.7}$$

Cez body získané numerickým výpočtom bola preložená fitovacia krivka $\varepsilon = AN^{-\frac{1}{2}}$, kde A je konštanta úmernosti. Ako vidieť z grafu 3.3-a, zistená presnosť metódy Monte Carlo presne súhlasí s teoretickým odvodením. Keďže za jednotku času sa odsimuluje stále rovnaký počet iónov, presnosť metódy Monte Carlo závisí od času výpočtu ako $\sim t^{-\frac{1}{2}}$.

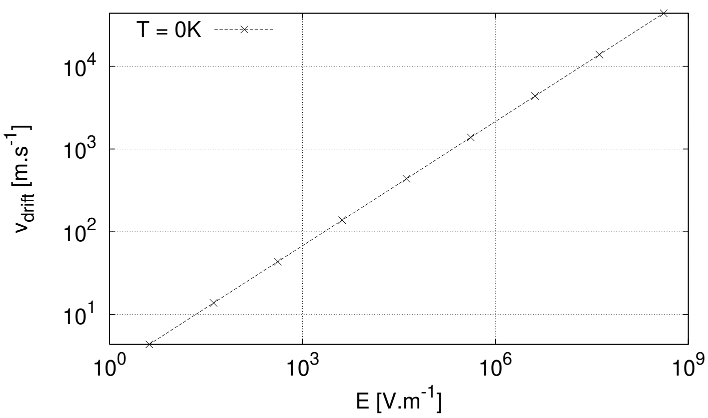
Distribučná funkcia iónov je v niekoľkých časových okamihoch znázornená na obrázku 3.3-b. Možno pozorovať, že prvotný delta pík[3] pomaly zaniká (v grafe však pokles jeho

[2] všeobecnejšie povedané od počtu simulácii

[3] Distribučná funkcia je tuto „zložená“ z dvoch častí. Jednou je spomínana delta funkcia (zvislé čiary v grafe). Druhou časťou je spočiatku lineárny pokles, z ktorého sa postupne formuje ustálený tvar rovnovážnej distribučnej funkcie daný vzťahom (2.30).



Obrázok 3.4: Závislosť driftovej rýchlosti od času pre rôzne hodnoty počiatočnej rýchlosti iónov pri teplote $T = 0$ K. Počet simulovaných iónov $N = 1.0 \times 10^6$.



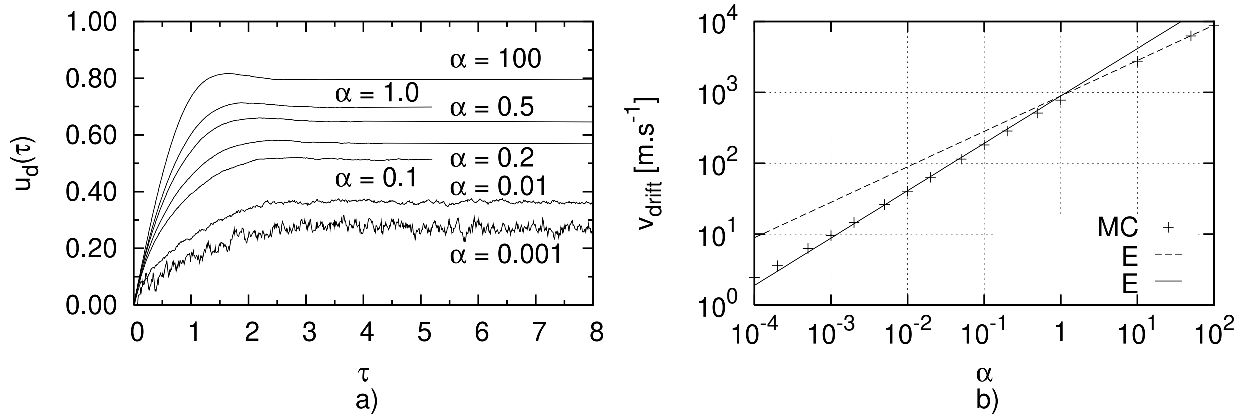
Obrázok 3.5: Závislosť driftovej rýchlosti od intenzity elektrického poľa.

výšky nie je dobre vidieť) a postupne sa formuje ustálený tvar rovnovážnej distribučnej funkcie. Pre porovnanie je v grafe zobrazený teoretický priebeh distribučnej funkcie v čase $\tau = \infty$ (vzťah (2.30)). Pre časy $\tau > 4.0$ už nemožno v grafe pozorovať odchýlky od rovnovážneho stavu, a preto nie sú zobrazené.

Zaujímavý je časový vývoj driftovej rýchlosti, ak počiatočná rýchlosť iónov je nenulová. Na obrázku 3.4 je táto závislosť zobrazená pre niekoľko hodnôt počiatočných rýchlostí. Fyzikálne najzaujímavejší je časový vývoj pre počiatočnú rýchlosť iónov $u_0 = \sqrt{2/\pi} \doteq 0.8$, teda hodnotu ustálenej driftovej rýchlosti v čase $\tau = \infty$. Očakávali by sme, že ak už ióny majú „požadovanú“ rýchlosť, tá sa meniť nebude. Opak je však pravdou. Driftová rýchlosť chvíľu narastá a aj v tomto prípade pozorujeme prekmit cez rovnovážnu hodnotu. Vnútorne sa totiž systém ešte nenachádza v rovnovážnom stave, čo dokazuje priebeh entropie.

3.3.2 Driftová rýchlosť s uvážením tepelného pohybu molekúl

Podobne ako v aproximácii studeného plynu, aj pri uvážení tepelného pohybu plynu boli sledované viaceré charakteristiky. Sledovali sa časové závislosti driftovej rýchlosti, distribučnej funkcie a entropie pri rôznom pomere tepelnej energie a energie od elektrického poľa, ktorý udáva koeficient α – rovnica (2.50). Cieľom bolo sledovať taktiež závislosť driftovej rýchlosti od intenzity elektrického poľa.



Obrázok 3.6: a) Časový priebeh driftovej rýchlosti pre rôzne intenzity elektrického poľa pri teplote $T = 300$ K. b) Závislosť driftovej rýchlosti od intenzity elektrického poľa. Naznačená smernica $v_{drift} = 891.05\ \alpha^{1/2}$ m. s^{-1} a fitovaná smernica $v_{drift} = 891.05\ \alpha^{0.668}$ m. s^{-1}. Počet simulovaných iónov hélia $N = 5.0 \times 10^5$.

Ako možno vidieť na obrázku 3.6-a, časový vývoj driftovej rýchlosti pre veľmi silné polia ($\alpha \sim 10^2$) je identický s prípadom, keď sme tepelný pohyb neutrálnych častíc plynu neuvažovali. V prípade, že je energia od elektrického poľa získavaná na strednej voľnej dráhe rovnaká ako tepelná energia ($\alpha \sim 1$), dochádza k miernemu znižovaniu hodnoty ustálenej driftovej rýchlosti.

Z teórie rezonančnej výmeny náboja [5] vyplýva, že v oblasti silných polí je závislosť driftovej rýchlosti od intenzity elektrického poľa daná vzťahom

$$v_{drift} = \sqrt{\frac{2}{\pi}\frac{eE\langle\lambda\rangle}{m}}, \tag{3.8}$$

čo je aj odvodené v kapitole 2.3.1. V prípade slabých polí je driftová rýchlosť daná vzťahom

$$v_{drift} = \frac{A}{\sqrt{mkT}} eE\langle\lambda\rangle, \tag{3.9}$$

kde konštanta $A = 0.330 \div 0.341$ [5]. Pohľadom na graf 3.6-b však vidíme, že oblasť slabých polí vykazuje inú závislosť ako je teoreticky predpovedaná závislosť $v_{drift} \sim E$. Fitovaním získaných numerických dát bola zistená závislosť

$$v_{drift} = v(\alpha = 1.0)\ \alpha^a, \tag{3.10}$$

kde exponent a zistený numericky je $a = 0.668264 \pm 0.002686$ a driftová rýchlosť $v(\alpha = 1.0)$ pre $\alpha = 1.0$ je

$$v(\alpha = 1.0) = \sqrt{\frac{2}{\pi}\alpha v_T^2} = \sqrt{\frac{2}{\pi}1.0\frac{2kT}{m}} = 891.05 \text{ m. s}^{-1}$$

Celkovo je teda hodnota driftovej rýchlosti v oblasti slabých polí[4] vyššia, ako by sa žiadalo. Inak by sme výsledok mohli interpretovať aj tak, že pri takomto fyzikálnom modeli, kedy sa náhodná tepelná rýchlosť generuje podľa rozdeľovacej funkcie danej vzťahom (2.35), sa principiálne nemôžeme dostať do oblasti slabých polí a stále zotrvávame v prechodnej oblasti[5]. Dôvodom je priebeh rozdeľovacej funkcie tepelných rýchlostí a ako sa neskôr pokusne ukázalo, je exponent závislosti driftovej rýchlosti od intenzity elektrického poľa citlivý na tvar rozdeľovacej funkcie.

Ak chceme zisťovať driftovú rýchlosť v oblasti slabých polí, teda $\alpha \ll 1.0$, musí byť aj energia dodaná iónu elektrickým poľom na strednej voľnej dráhe menšia ako tepelná energia iónu. Pri rozdeľovacej funkcii (2.35) však pravdepodobnosť, že ma ión rýchlosť $v = 0$, a teda že je jeho rýchlosť [6] nižšia ako v_T, relatívne veľmi vysoká. To znamená, že veľká skupina iónov má malú tepelnú energiu, a teda energia dodávaná poľom nie je nižšia ako tepelná energia. Dôsledok: pole nie je slabé.

Pri trojrozmernom modeli by bola situácia iná. Porovnaním rozdeľovacích funkcií pre jednorozmerný a trojrozmerný model (obrázok 3.7) vidíme, že zatiaľ čo pre $u = 0$ nadobúda funkcia $f(u)$ maximum, rozdeľovacia funkcia $f_{3D}(u)$ nadobúda minimum. V trojrozmernom prípade je teda skupina iónov s rýchlosťou $v = 0$ málo početná a energia dodaná elektrickým poľom ma väčšiu „šancu“ byť menšia ako tepelná energia iónu. To umožňuje ísť s intenzitou elektrického poľa dolu a skúmať závislosť driftovej rýchlosti aj v týchto slabých poliach.

Môžeme teda spraviť teoretický experiment. Uvažujme nasledujúce rozdeľovacie funkcie:

$$f(v_x)dv_x = \frac{1}{\sqrt{\pi}v_T}\exp\left(-\frac{v_x^2}{v_T^2}\right)dv_x \tag{3.11}$$

$$f_\beta(v_x)dv_x = \begin{cases} 0, & |v_x| < \beta v_T \\ \dfrac{1}{\mathrm{erfc}(\beta)}\dfrac{1}{\sqrt{\pi}v_T}\exp\left(-\dfrac{v_x^2}{v_T^2}\right)dv_x, & |v_x| > \beta v_T \end{cases} \tag{3.12}$$

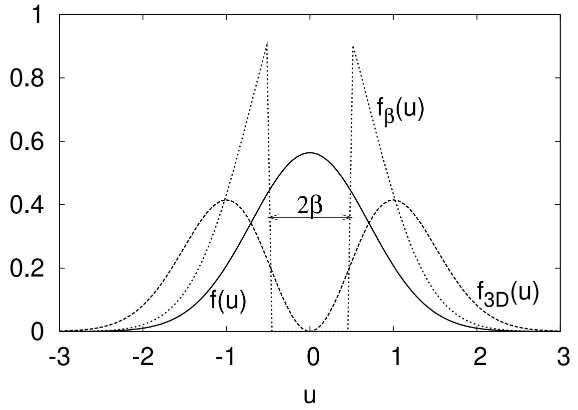
$$f(v)dv = \frac{4\pi}{(\sqrt{\pi}v_T)^3}\exp\left(-\frac{v^2}{v_T^2}\right)v^2dv \tag{3.13}$$

Rovnica (3.12) predstavuje jeden z jednoduchých spôsobov, ako „vyrezať“ pomalé ióny a priblížiť sa tak k trojrozmernému modelu. Hodnotou koeficientu β možno riadiť šírku vyrezaného pásma, a tak sledovať vplyv pomalých iónov na závislosť driftovej rýchlosti. Použitím rovnice (3.13) možno vybudovať akýsi hybridný jeden a pol rozmerný model.

[4]V héliu $He - He^+$ pri teplote $T = 300$ K a strednej voľnej dráhe $\langle\lambda\rangle = 3.0\times10^{-7}$ m je hranica silných a slabých polí pri intenzite elektrického poľa asi $E = 170.0$ kV. m^{-1}

[5]V prechodovej oblasti $\alpha = 1.0$ dôjde pri prechode od silných polí k slabým k zmene smernice (logaritmická škála) z hodnoty 0.5 na hodnotu 0.668, ďalej sa však už nemení.

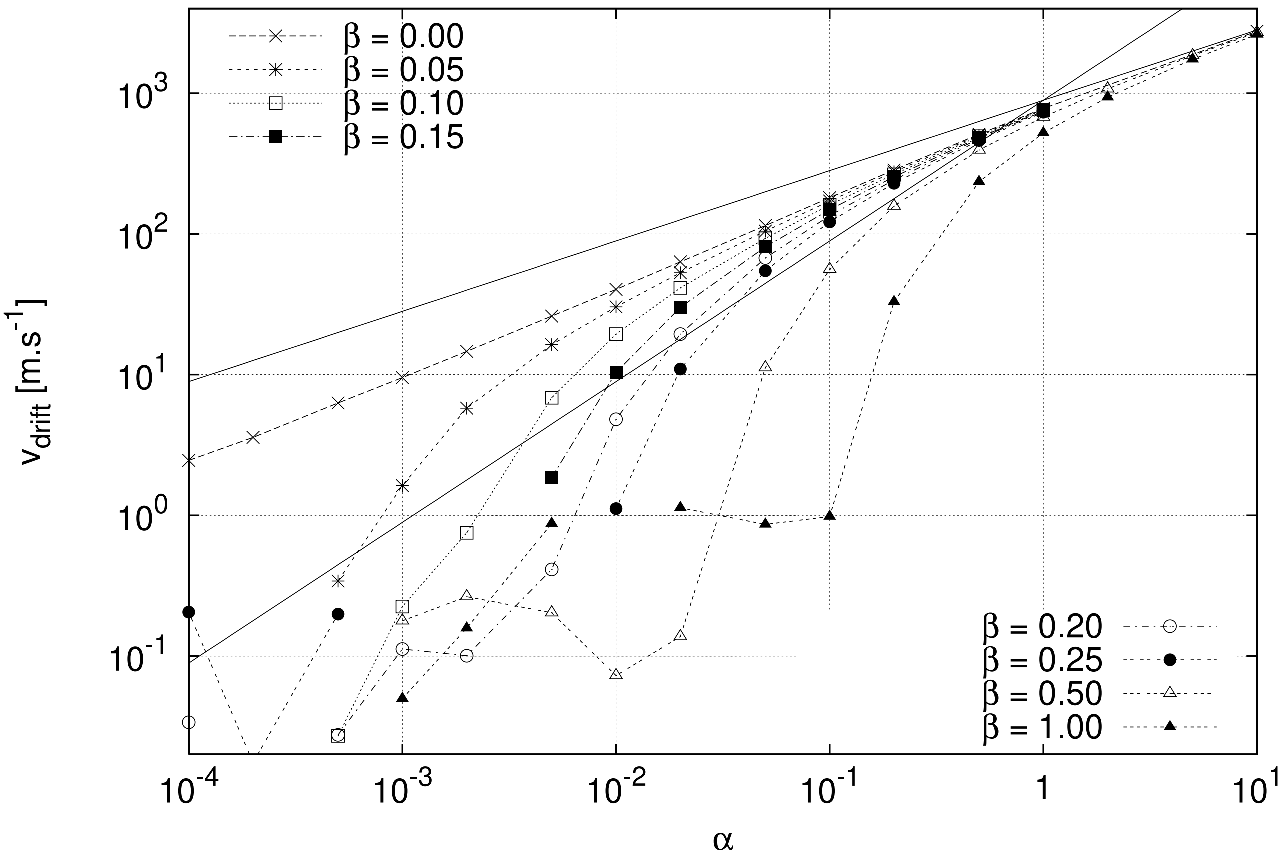
[6]všimnime si, že práve hodnota rýchlosti v_T vystupuje vo vzťahu (2.50) pre koeficient α udávajúci „silu“ poľa



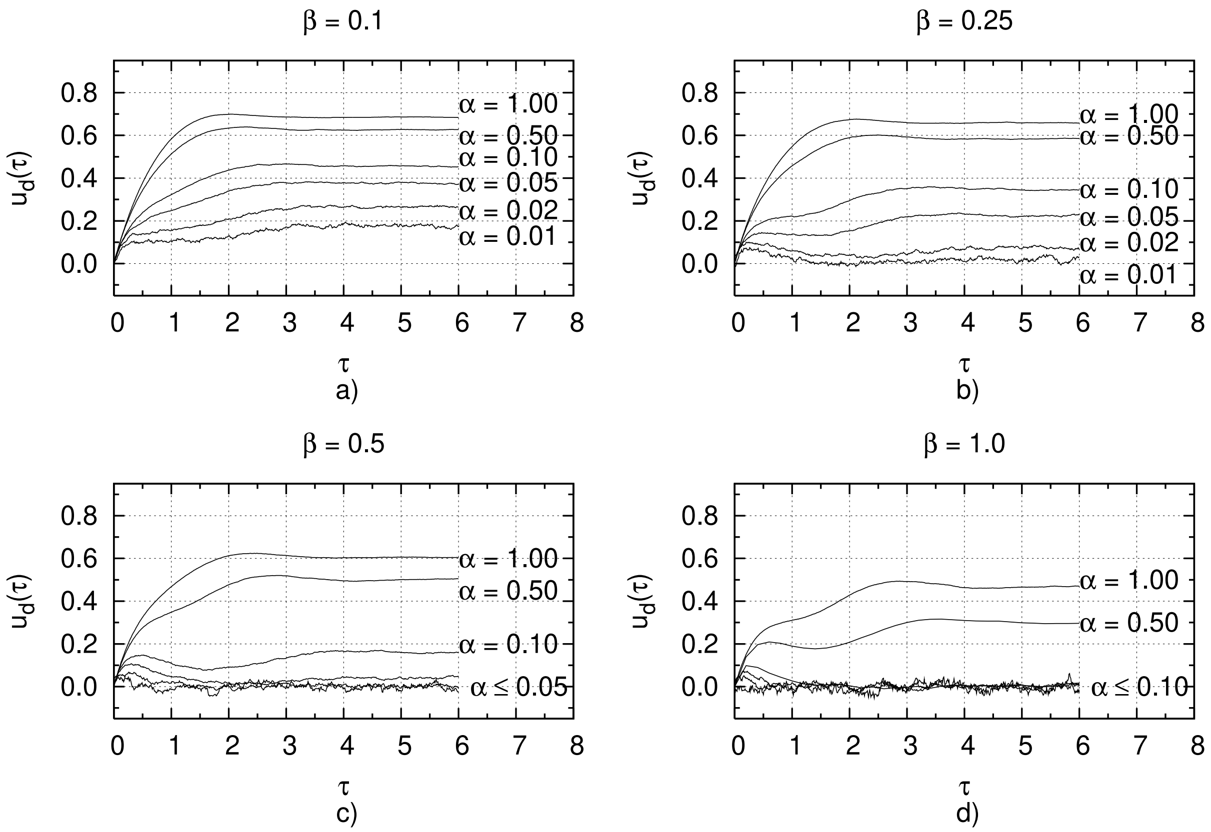
Obrázok 3.7: Porovnanie distribučných funkcií. Zobrazené distribučné funkcie sú normalizované, $u = v/v_T$. $f_\beta(u)$ vykreslené pre $\beta = 0.5$.

Vplyv koeficientu β na priebeh driftovej rýchlosti je na obrázku 3.8. V grafe sú znázornené závislosti driftovej rýchlosti od intenzity elektrického poľa (resp. koeficientu α, ktorý je s ňou spätý) pre rôzne hodnoty koeficientu β. Možno taktiež sledovať vplyv výberu koeficientu β na priebeh krivky. Už pri veľmi malej hodnote koeficientu $\beta \sim 0.05$ dochádza k ovplyvneniu priebehu. V oblasti $\alpha = 1.0 \times 10^{-2}$ pre $\beta = 0.05$ má krivka driftovej rýchlosti požadovanú smernicu, no pri prechode k nižším poliam sa krivka láme viacej dolu a smernica sa zvyšuje. Pri zvyšovaní hodnoty koeficientu β dochádza k tomuto zlomu skôr a pokles je väčší. Pri slabých poliach ($\alpha < 1.0 \times 10^{-2}$) však pre väčšie hodnoty koeficientu β zvyknú byť hodnoty driftovej rýchlosti veľmi blízke nule, resp. mierne záporné, a teda v logaritmickej škále ich nemožno zobraziť. Výsledky, ktoré dostávame, sú akýsi tepelný šum. Príčina príliš veľkého poklesu hodnoty driftovej rýchlosti pri vyšších hodnotách koeficientu β jasná nie je. V oblasti silných polí sú krivky pre všetky hodnoty koeficientu β vo veľmi dobrej zhode s teoretickým priebehom.

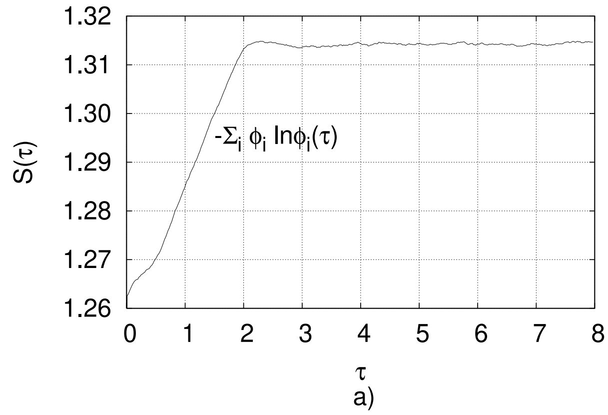
Časové priebehy driftovej rýchlosti pre rôzne hodnoty koeficientu β sú znázornené na obrázku 3.9 (strana 29). V grafe si možno všimnúť, že ustálenie driftovej rýchlosti nastáva neskôr ako v prípade bez tepelného pohybu. Ak sa zaujímame o entropiu, môžeme vychádzať napr. z grafu 3.10. Ak porovnáme priebeh entropie s priebehom entropie pre $T = 0$ K, vidíme viacero rozdielov. V tomto prípade už entropia v čase $\tau = 0$ je nenulová. Dôvodom je to, že počiatočná rýchlosť iónu bola generovaná podľa rozdeľovacej funkcie (3.12), teda počiatočná rozdeľovacia funkcia iónov nebola delta funkcia. Ďalej si možno všimnúť počiatočný hrb ($\tau < 0.5$) a väčšie oscilácie v ustálenom stave spôsobené tepelným šumom.



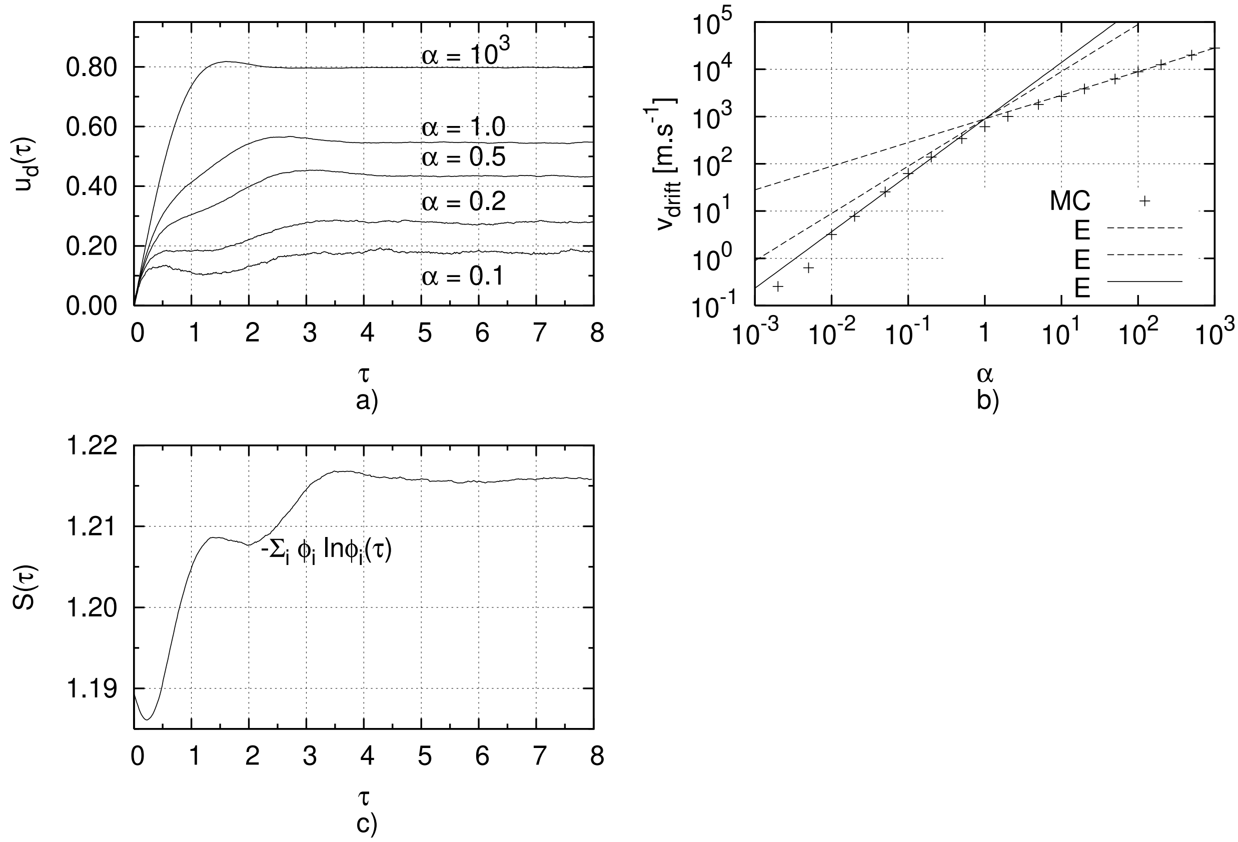
Obrázok 3.8: Závislosť driftovej rýchlosti iónov hélia od intenzity elektrického poľa pri teplote $T = 300$ K pri rôznych hodnotách koeficientu β.



Obrázok 3.9: Časový priebeh driftovej rýchlosti pre rôzne intenzity elektrického poľa pri teplote $T = 300$ K pri rôznych koeficientoch β.

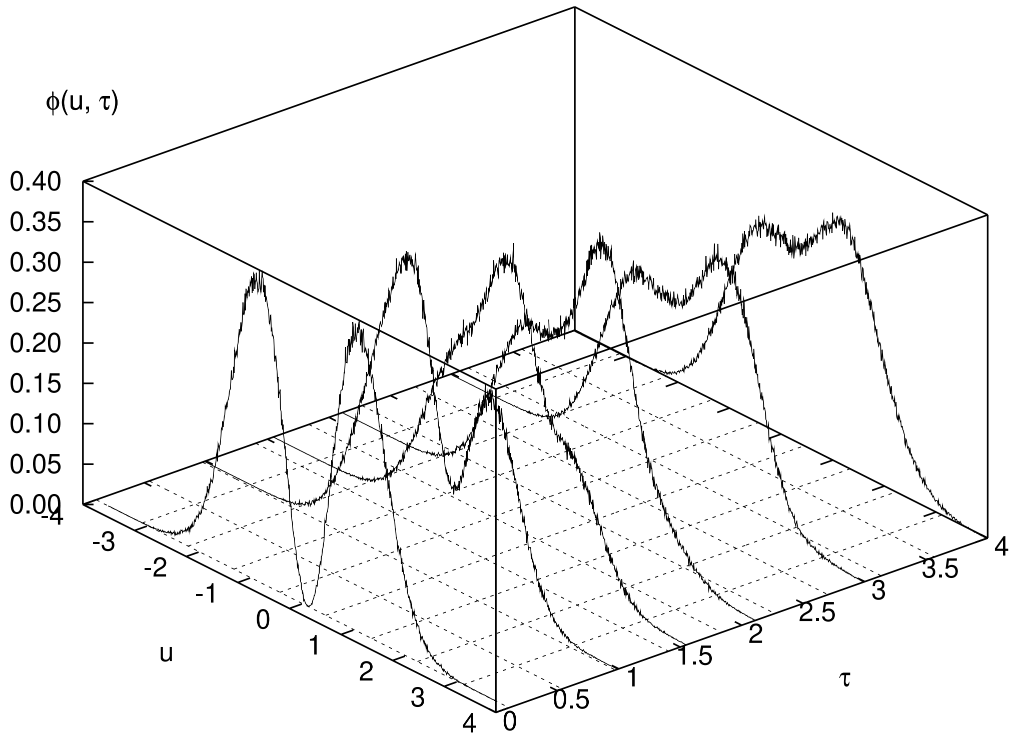


Obrázok 3.10: Časový priebeh entropie pre $\alpha = 0.1, \beta = 0.3$ pri teplote $T = 300$ K.



Obrázok 3.11: a) Časový priebeh driftovej rýchlosti pre rôzne intenzity elektrického poľa pri teplote $T = 300$ K ak sa použije rozdeľovacia funkcia (3.13). b) Závislosť driftovej rýchlosti od intenzity elektrického poľa. c) Časový vývoj entropie pre $\alpha = 1.0$.

Na obrázku 3.11 sú vykreslené závislosti driftovej rýchlosti od času a intenzity elektrického poľa a časový priebeh entropie. Distribučná funkcia je znázornená na obrázku 3.12 v trojrozmernom zobrazení, keďže zobrazenie jej časového vývoja v dvoch rozmeroch je neprehľadné. Tieto závislosti boli získané pri použití rozdeľovacej funkcie (3.13), teda rozdeľovacej funkcie rýchlosti iónov platnej v trojrozmernom priestore. Ako sa ukazuje, pri takomto jednoduchom modeli, kedy je zrážka rezonančnej výmeny náboja simulovaná jednorozmerne a iba rozdeľovacia funkcia je braná „trojrozmerne“, možno dosiahnuť pomerne dobrú zhodu s teoretickým modelom, ktorý hovorí, že v oblasti slabých polí závisí driftová rýchlosť od intenzity elektrického poľa lineárne (rovnica (3.9)). V oblasti $10^{-1} < \alpha < 1$ sa zdá byť tento zákon pomerne dobre splnený, smerom k vyšším intenzitám závislosť prechádza k smernici 0.5, čo je v zhode s teóriou pre silné polia (rovnica (3.8)). Pre polia, kedy je $\alpha < 10^{-1}$ sa numericky získané výsledky odchyľujú od teórie a driftová rýchlosť závisí od intenzity elektrického poľa ako $v_{drift} \sim E^a$, kde konštanta a určená fitovaním krivky je $a = 1.194 \pm 0.007$. Táto odchýlka by mohla byť vysvetlená tým, že fyzikálny model nebol plne trojrozmerný.



Obrázok 3.12: Distribučná funkcia iónov hélia (He$-$He$^+$) pri teplote $T = 300$ K. $\alpha = 1.0$, počet simulovaných iónov $N = 5.0 \times 10^5$. Bezrozmerná škála.

Kapitola 4

Záver

V tejto diplomovej práci bola študovaná relaxácia driftovej rýchlosti iónov pri rezonančnej výmene náboja. Práca nadväzuje na [1], kde relaxácia driftovej rýchlosti už bola študovaná. Výsledky boli nezávisle overené a bola dosiahnutá výborná zhoda. Problematika relaxácie driftovej rýchlosti je ďalej rozšírená pre prípad nenulovej teploty, teda proces rezonančnej výmeny náboja je študovaný nielen v aproximácii studeného plynu resp. v oblasti silných polí.

Bol navrhnutý program, ktorým možno proces rezonančnej výmeny náboja simulovať metódou Monte Carlo. Túto simuláciu možno s dostatočnou presnosťou prevádzať na počítačoch bežnej výkonnostnej triedy. Výpočet pre $\sim 10^5$ častíc trvá rádovo minúty a je náročný na výpočtový výkon počítača. Najnáročnejšou operáciou je proces generovania pseudonáhodných čísel. Zrýchlenie programu by sa mohlo dosiahnuť úpravou generátora náhodných čísel, keďže von Neumannova metóda výberu je pomerne málo efektívna. Pri implementácii plne trojrozmerného modelu procesu zrážky by pravdepodobne bolo potrebné túto časť programu dopracovať, lebo potreba väčšieho počtu náhodných čísel a taktiež vyššia náročnosť výpočtov by spôsobila niekoľkonásobné predĺženie času simulácie.

Celkovo je však metóda Monte Carlo pomerne rýchla a presná a výsledky ňou dosiahnuté sú veľmi zaujímavé. Miernou nevýhodou je pomalá konvergencia, keďže presnosť metódy je úmerná času výpočtu ako $\sim t^{-1/2}$. Ak by sme sa uspokojili s nižšou presnosťou výpočtu, rýchlosť výpočtu by sa značne znížila. Výsledky dosiahnuté touto metódou sú (čo sa týka presnosti) porovnateľné s experimentálnymi údajmi, avšak na rozdiel od experimentu môže byť počítačová simulácia omnoho lacnejšia.

Teoretická závislosť driftovej rýchlosti od intenzity elektrického poľa pre silné polia sa potvrdila. Problematickou sa však ukázala oblasť slabých polí. Pri jednoduchom jednorozmernom fyzikálnom modele bolo nemožné dostať sa do oblasti slabých polí. Dôvodom bol už diskutovaný problém pomalých iónov, teda iónov s malou tepelnou energiou.

To, že práve tieto pomalé ióny v jednorozmernom modele ovlyvňujú exponent závislosti driftovej rýchlosti od intenzity elektrického poľa, bolo dokázané zmenou rozdeľovacej funkcie rýchlosti, ktorú ióny nadobúdajú po zrážke (procese rezonančnej výmeny náboja). Vyseknutie už veľmi úzkeho rýchlostného intervalu spôsobuje pozorovateľnú zmenu závislosti driftovej rýchlosti.

Ako sa ukázalo, použitie trojrozmernej rozdeľovacej funkcie rýchlosti iónov v jednorozmernom modeli umožňuje dostať sa veľmi blízko k teoretickému priebehu driftovej rýchlosti aj v oblasti slabých polí. Kedže však trojrozmerný model, pre ktorý je predpoveď o lineárnej závislosti driftovej rýchlosti od intenzity elektrického poľa vypracovaná, nebol plne implementovaný, pre veľmi slabé polia ($\alpha \ll 0.01$) dochádza k odchýlke od tejto lineárnej závislosti. Preto sa predpokladá, že dopracovaním tohto modelu by sa dosiahla výborná zhoda.

Literatúra

[1] Martišovitš V., Šimko T.: *Transient phenomena at non-hydrodynamic ion transport in gases*, Zborník 11th Symposium on Element. Processes and Chem. Reactions in Low Temperature Plasma, Low Tatras 1998, Book of Invited Papers, (Ed. Morvová M., Hensel K.), JSMF, 110 – 135.

[2] Dřímal J., Trunec D.: *Úvod do metódy Monte Carlo*, Přírodovědecká fakulta UJEP, Brno 1989.

[3] Buslenko N. P., Šrejder J. A.: *Stochastické početní metody Monte Carlo*, Státni nakladatelství technické literatury, Praha 1965.

[4] Kruer W. L.: *The physics of laser plasma interactions*, Frontiers in physics 73, Addison-Wesley Publishing Company, Inc., USA and Canada 1988.

[5] McDaniel E. W., Mason E. A.: *The mobility and diffusion of ions in gases*, John Wiley and Sons, New York – London – Sydney – Toronto 1973.

[6] Hronec J.: *Diferenciálne rovnice II. – Parciálne diferenciálne rovnice*, Vydavateľstvo Slovenskej akadémie vied, Bratislava 1958.

[7] Zajac R., Pišút J.: *Štatistická fyzika*, Matematicko-fyzikálna fakulta Univerzity Komenského, Bratislava 1995.

[8] von Engel A.: *Electric plasmas: Their nature and uses*, Taylor & Francis Ltd, London and New York 1983.

[9] Bartsch H. J.: *Matematické vzorce*, Mladá fronta, Praha 1996.

Internetové odkazy

[10] `http://www.gnu.org`

[11] `http://www.debian.org`

[12] `http://www.vim.org`

[13] `http://gcc.gnu.org/`

[14] http://www.gnu.org/directory/gdb.html

[15] http://freshmeat.net/projects/libcfgplus

[16] http://platon.sk/projects/libcfg+

Príloha A

Zdrojový kód programu

```
/*
 * Monte Carlo simulation of distribution function and drift speed
 * of ions in charge-exchange collision. One dimensional model.
 * Thermal motion of neutral gas taken into account.
 *
 * Developed by Lubomir Host 'rajo' <rajo AT platon.sk>
 * Licensed under terms of GNU General Public License.
 *
 * Compilation & executing:
 *
 * gcc -DDRIFT   -o hotGas-drift    hotGas.c -lm -lcfg+
 * gcc -DDISTRIB -o hotGas-distrib  hotGas.c -lm -lcfg+
 *
 * ./hotGas-drift N=1e6 time=8.0 d_time=0.01 T=0 lambda=3e-7 u=2.0 a=1e10
 *
 * libcfg+ library needed for compilation can be downloaded from
 *
 * http://freshmeat.net/projects/libcfgplus/
 * http://platon.sk/projects/libcfg+/
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <time.h>

#include <platon/cfg+.h>

/* type used for real numbers */
#define Real_t           double
/* constant used only for generation Mxw. distribution*/
#define MAX_V_T_CONST    4.5

/* DEBUG rutines, skip to line #140 {{{ */
#ifdef DEBUG

int debug          = 1;
int debug_prefixed = 0;

#define DEBUG_PREFIX "# @@ "

#define _DEBUG_MSG(_level, _msg...)                             \
    /* if _level = 0, print without prefix */                   \
    if (!(_level)) {                                            \
        if (debug_prefixed)                                     \
            fprintf(stderr, "\n");                              \
```

```
        fprintf(stderr, _msg);                                  \
        debug_prefixed = 0;                                     \
    }                                                           \
    else if (debug >= (_level)) {                               \
        /* first create whole string */                         \
        char debug_string[4096];                                \
        snprintf(debug_string, 4096, _msg);                     \
        /* if string ends with '\n', prefix it */               \
        if ((debug_string + strlen(debug_string)) !=            \
                (strrchr(debug_string, '\n') + 1)) {            \
            /* if something was already prefixed, terminate */  \
            if (debug_prefixed) {                               \
                fprintf(stderr, "%s", debug_string);            \
            }                                                   \
            else {                                              \
                fprintf(stderr, DEBUG_PREFIX "%.*s%s",          \
                        (_level),   "               ",          \
                        debug_string);                          \
                debug_prefixed = 1;                             \
            }                                                   \
        }                                                       \
        /* if string doesn't ends with '\n', print as is,      */ \
        /* but prefix it with DEBUG_PREFIX if not prefixed yet */ \
        else {                                                  \
            /* if previous output was prefixed, terminate it   */ \
            if (debug_prefixed) {                               \
                fprintf(stderr, "\n");                          \
            }                                                   \
            fprintf(stderr, DEBUG_PREFIX "%.*s%s",              \
                    (_level),   "               ",              \
                    debug_string);                              \
            /* and next output must be prefixed */              \
            debug_prefixed = 0;                                 \
        }                                                       \
    }                                                           \

/* executes code if debug enabled */
#define _DEBUG(_level, _code)                                   \
    if (debug >= _level) {                                      \
        _code;                                                  \
    }

/* executes code if debug enabled */
#define _DEBUG_BEGIN(_level)                                    \
    if (debug >= _level) {
#define _DEBUG_END(_level)                                      \
    }

int dbg_rnd_idx = 0;
/* "random" speeds returned in DEBUG mode */
Real_t dbg_rnd_speed[] = {
    0.5, 1.5, 1.2, 0.3,
    1.7, 2.5, 1.0, 0.1,
    0.3, -1, -2, -3,
    -4, -5
};
Real_t dbg_rnd_time[]  = {
    2.2, 0.4, 0.6, 0.8, /* t = 4.0 */
    0.8, 0.1, 1.1, 0.5, /* t = 6.5 */
    1.5, 1, 1, 1
};

#else /* DEBUG not defined */

#define _DEBUG_MSG(_level, _msg...)    /* empty */
#define _DEBUG(_level, _code)          /* empty */
#define _DEBUG_BEGIN(_level)           \
    if (0) { /* executes never */
```

```
#define _DEBUG_END(_level)             \
    }

#endif /* }}} DEBUG rutines */

#ifndef COMPILE_STRING
#  define COMPILE_STRING "manual"
#endif

char *version = __FILE__ ": " __DATE__ " " __TIME__ /* version {{{ */
#ifdef DEBUG
", DEBUG defined"
#endif
#ifdef DISTRIB
", DISTRIB defined"
#endif
; /* i}}} end of version[] */

/* Variables declaration, default values {{{ */
long int N          = 100000;          /* number of ions*/
Real_t e_charge     = 1.602176462E-19; /* elementary charge [C] */
Real_t k_B          = 1.3806503E-23;   /* Boltzmann constant [J.K^-1] */
Real_t m_u          = 1.66053873E-27;  /* atomic mass constant [kg] */
Real_t T_g          = 0.0;             /* temperature of neutral gas in Kelvins */
Real_t m_g          = 4;               /* molekul mass [atomic mass units] He=4 He_2=8 etc.*/
Real_t lambda       = 0.001;           /* mean free path in meters */
Real_t acceleration = 1.0;             /* acceleration */
Real_t E            = 0.0;             /* intensity of electric field */
Real_t alpha        = 1.0;             /* ratio between energy from field and thermal energy */
Real_t beta         = 0.0;             /* random thermal speeds > beta * v_T */
Real_t start_speed  = 0.0;             /* startup speed of ions */
Real_t abs_acc      = 0.0;             /* absolute value of acceleration */
Real_t max_time     = 10.0;            /* max. time of simulation */
Real_t real_time    = 0.0;             /* real time computed from start of simulation */
Real_t d_time       = 1E-2;            /* time step */
Real_t v_T          = 0.0;             /* thermal speed v_T = sqrt( 2 * k_B * T_g / m_g) */
Real_t v_drift      = 0.0;             /* theoretical drift speed in time = \infty
                                        *  v_drift = sqrt(2 * a * lambda / 3.1415) */

Real_t Dimens_x     = 0.0;             /* normalisation constant of position */
Real_t Dimens_v     = 0.0;             /* normalisation constant of speed */
Real_t Dimens_t     = 0.0;             /* normalisation constant of time */

#ifdef DISTRIB
long int **Fx_plus,  **Fv_plus;        /* distribution funkcions over x and over v */
long int **Fx_minus, **Fv_minus;       /* distribution funkcions for negative x, v */
long int Nt    = 100;                  /* # of points in time */
long int Nx    = 100;                  /* number of points in x/v axis */
long int idx   = 0;                    /* index into arrays with distribution function */
Real_t f_max_x = 5.0;                  /* x-limit for distribution function, dimensionless */
Real_t f_max_v = 5.0;                  /* v-limit for distribution function, dimensionless */
Real_t d_f_max_x;                      /* interval in distribution function */
Real_t d_f_max_v;                      /* interval in distribution function */
int output_step = 1;                   /* time step in output of distribution funkction */
#endif
/* }}} Variables declaration */

#define xmalloc(_type, _x, _count)                    /* {{{ */    \
    if ((_count) == 0)                                             \
        fprintf(stderr,                                            \
            "WARNING: memory allocation "                          \
            "(" #_type ") " #_x " of 0 counts ");                  \
        _x = (_type *) malloc((_count) * sizeof(_type));           \
        if (_x == NULL) {                                          \
            fprintf(stderr, "Can't alloc array " #_x               \
                "[%ld] of type (" #_type "), errno = %d - %s\n",   \
                (_count), errno, strerror(errno));                 \
                exit(errno);                                       \
```

```
        }                                                         \
    memset(_x, 0, (_count) * sizeof(_type)); /* }}} */

/* return signum of number */
Real_t signum(double x)
{ /* {{{ */
    if (x < 0.0)
        return -1.0;
    else
        return  1.0;
} /* }}} */

/* return random thermal speed */
Real_t random_thermal_speed(void)
{ /* {{{ */
    Real_t ret_v;       /* returned speed */
    Real_t ret_v2;      /* speed square */
    Real_t func;        /* distribution function */

#ifdef DEBUG
    _DEBUG_BEGIN(5)
        /* if debug is on, return speed from dbg_rnd_speed[] array */
        return dbg_rnd_speed[dbg_rnd_idx++];
    _DEBUG_END(5)
#endif

    if (v_T == 0.0)
        return 0.0;

    /*
     * Maxwell distribution in 1-D model:
     *
     *               1                    v_x^2
     * f_T(v_x) = --------------- exp ( - --------)
     *            sqrt(\pi) v_T           v_T^2
     *
     * v_T = sqrt( 2 k_B T / m_g)
     */

    /* von Neumann method */
    while (1) {
        ret_v  = -MAX_V_T_CONST + 2.0 * MAX_V_T_CONST * (rand() / (RAND_MAX + 1.0));
        ret_v2 = ret_v * ret_v;
#ifdef DISTRIB_FUNC_3D
        func = 2.7 * ret_v2 * exp(- ret_v2); /* 2.7 = exp(-1) */
#else
        func = exp( - ret_v2);
#endif
        if (fabs(ret_v) > beta
                && ((1.0 * (rand() / (RAND_MAX + 1.0))) <= func)) {
            return v_T * ret_v;
        }
    }
} /* }}} */

/* return time to the next collison */
Real_t random_time(Real_t v)                              /* [LINE i] */
{ /* {{{ */
    Real_t rand_x, rand_time;
    Real_t z;

#ifdef DEBUG
    _DEBUG_BEGIN(5)
        /* if debug is on, return speed from dbg_rnd_speed[] array */
        _DEBUG_MSG(1, "random_time(%g) = rand_time = %g\n",
                v, dbg_rnd_time[dbg_rnd_idx]);
        return dbg_rnd_time[dbg_rnd_idx];
    _DEBUG_END(5)
```

```
#endif
    /* Distribution function:
     *
     * f(x) = (1 / lambda) * exp( - x / lambda)
     *
     * integral f(x):
     *
     * F(x) = 1 - exp( - x / lambda)
     *
     * inverse function:
     *
     * F^-1(y) = - lambda * ln(1 - y)
     *
     */

    /* inversion function method */
    z      = rand() / (RAND_MAX + 1.0);
    /* rand_x    = - lambda * log(1.0 - z); */
    rand_x = -lambda * log(z); /* optimalized */
    /*
     * Analytical solution of quadr. equation:
     * 0.5*a*t^2 + v_0*t - lambda = 0
     * Special case sgn(a) != sgn(v) && v < sqrt(2*lambda*fabs(a))
     */
    if (v == 0.0) {
        return sqrt(2 * rand_x / abs_acc);
    }
    if (signum(v) != signum(acceleration) && fabs(v) < sqrt(2 * rand_x * abs_acc))
        rand_time = (fabs(v) + sqrt(2 * abs_acc * rand_x - v * v)) / abs_acc;
    else
        rand_time = (2.0 * rand_x) /
            (fabs(v) + sqrt(v * v + 2 * acceleration * rand_x * signum(v)));
    _DEBUG_MSG(1, "random_time(%g) = rand_time = %g\n", v, rand_time);

    return rand_time;

} /* }}} */

/* print parameters after start of simulation */
void print_parameters(void)
{    /* parameter dump {{{ */

    fprintf(stdout,
            "## COMPILATION: %s\n"
            "## VERSION: %s\n"
            "## PARAM:  N                 = %ld\n"
            "## PARAM:  time_step         = %f\n"
            "## PARAM:  max_time          = %f\n"
            "## PARAM:  acceleration      = %g\n"
            "## PARAM:  E                 = %g\n"
            "## PARAM:  start_speed       = %g\n"
            "## PARAM:  alpha             = %g\n"
            "## PARAM:  beta              = %g\n"
#ifdef DISTRIB
            "## PARAM:  Nt                = %ld\n"
            "## PARAM:  Nx                = %ld\n"
            "## PARAM:  f_max_x           = %f\n"
            "## PARAM:  f_max_v           = %f\n"
            "## PARAM:  output_step       = %d\n"
#endif
            "## PARAM:  T_g               = %f\n"
            "## PARAM:  m_g               = %.10g\n"
            "## PARAM:  v_T               = %g\n"
            "## PARAM:  lambda            = %.5g\n"
            "## VALUE:  real_max_time     = %.10g\n"
            "##         sqrt(lambda/a)    = %.10g\n"
            "## VALUE:  real_d_time       = %.10g\n"
            "## VALUE:  k_B * T_g         = %.10g\n"
```

```
            "## VALUE:  lambda * q * E   = %.10g\n"    /*   a*m = q*E   */
            "## (%s field)\n",
            COMPILE_STRING, version,
            /* d_time and max_time must be recomputed, because its values are
             * adjusted to Dimens_t */
            N, d_time / Dimens_t, max_time / Dimens_t,
            acceleration, E, start_speed / Dimens_v,
            alpha, beta,
#ifdef DISTRIB
            Nt, Nx, f_max_x, f_max_v, output_step,
#endif
            T_g, m_g, v_T, lambda,
            max_time, Dimens_t,
            d_time, k_B * T_g,
            (lambda * abs_acc * m_g),
            (lambda * abs_acc * m_g) < (2 * k_B * T_g) ? "low" : "STRONG");
    fflush(stdout);
} /* }}} */

int main(int argc, char **argv)
{
    double         i = 0;
    long int       j = 0L, k = 0L;    /* counters */
    Real_t         rand_time;
    Real_t         x, v, t, time_left;
#ifdef DISTRIB
    long int       l = 0L;            /* counters */
#else
    Real_t         *X;                /* x coordinate of ion */
    Real_t         *U;                /* drift speed */
    long int       Nb;
#endif
    int             help = 0;

    /* {{{ cmdline parsing */
    /* libcfg+ parsing context */
    CFG_CONTEXT con;

    /* Parsing return code */
    register int ret;

    /* Option set */
    struct cfg_option options[] = {
        { "help",        'h', NULL, CFG_BOOL,   (void *) &help,         0 },
        { "N",           'N', NULL, CFG_LONG,   (void *) &N,            0 },
#ifdef DISTRIB
        { "Nx",          'X', NULL, CFG_LONG,   (void *) &Nx,           0 },
        { "f_max_x",     'x', NULL, CFG_DOUBLE, (void *) &f_max_x,      0 },
        { "f_max_v",     'v', NULL, CFG_DOUBLE, (void *) &f_max_v,      0 },
        { "output_step", 's', NULL, CFG_INT,    (void *) &output_step,  0 },
#endif
        { "d_time",      'd', NULL, CFG_DOUBLE, (void *) &d_time,       0 },
        { "time",        't', NULL, CFG_DOUBLE, (void *) &max_time,     0 },
        { "a",           'a', NULL, CFG_DOUBLE, (void *) &acceleration, 0 },
        { "alpha",       'A', NULL, CFG_DOUBLE, (void *) &alpha,        0 },
        { "beta",        'B', NULL, CFG_DOUBLE, (void *) &beta,         0 },
        { "u",           'u', NULL, CFG_DOUBLE, (void *) &start_speed,  0 },
        { "T",           'T', NULL, CFG_DOUBLE, (void *) &T_g,          0 },
        { "mass",        'm', NULL, CFG_DOUBLE, (void *) &m_g,          0 },
        { "lambda",      'l', NULL, CFG_DOUBLE, (void *) &lambda,       0 },
#ifdef DEBUG
        { "debug",       'd', NULL, CFG_INT,    (void *) &debug,        0 },
#endif
        CFG_END_OF_LIST
    };

    /* Creating context */
    con = cfg_get_context(options);
```

```
    if (con == NULL) {
        puts("Not enough memory");
        return 1;
    }
    cfg_clear_property(con, CFG_LINE_LONG_OPTION_PREFIX);
    cfg_add_properties_type(con, CFG_LINE_LONG_OPTION_PREFIX, "--", "", NULL);

    /* Parse command line from second argument to end (NULL) */
    cfg_set_cmdline_context(con, 1, -1, argv);

    /* Parsing command line */
    ret = cfg_parse(con);
    if (ret != CFG_OK) {
        fprintf(stderr, "error parsing command line: ");
        cfg_fprint_error(con, stderr);
        fprintf(stderr, "\n");
        return ret < 0 ? -ret : ret;
    }

    /* Help screen {{{ */
    if (help) {
        puts("Usage:");
        printf("  %s [options]\n", argv[0]);
        puts("Options:");
        puts("  -h  help                         print this help screen");
        puts("  -N  N=<number>                   number of simulated ions");
        puts("  -d  d_time=<time_step>        [dimensionless]");
        puts("                                   simulation time step");
        puts("  -t  time=<max_time>           [dimensionless]");
        puts("                                   time of simulation");
        puts("  -a  a=<acceleration>          [meter.sec^-2]");
        puts("                                   acceleration of ions");
        puts("  -A  alpha=<koeficient>        [dimensionless]");
        puts("                                   alpha = acceleration * lambda / (v_T * v_T)");
        puts("  -B  beta=<koeficient>         [dimensionless]");
        puts("                                   random thermal speeds > beta * v_T");
        puts("  -u  u=<speed>                 [dimensionless]");
        puts("                                   startup speed of ions normalized to");
        puts("                                   sqrt(lambda*acceleration)");
        puts("  -T  T=<T_g>                   [Kelvin]");
        puts("                                   temperature of neutral gas");
        puts("  -m  mass=<m_g>                [atomic mass units]");
        puts("                                   mass of neutral gas molecule and ions (He=4)");
        puts("  -l  lambda=<mean_free_path>   [meters]");
        puts("                                   mean free path");
#ifdef DISTRIB
        puts("  -X  Nx=<pixels>               [count]");
        puts("                                   pixel per x axis in f(x,t) and/or f(v,t)");
        puts("  -x  f_max_x=<max_x>              range of x axis");
        puts("  -v  f_max_v=<max_v>              range of v axis");
        puts("  -s  output_step=<step>           don't print distrib. f. in every time step");
#endif
#ifdef DEBUG
        puts("  -d  debug=<level>                enable debug level <level>");
#endif
        puts("");
        printf("%s N=1e6 time=10 d_time=1e-2 T=10 lambda=3e-7 u=2.0 a=1e7\n", argv[0]);
        puts("");
    } /* }}} */
    /* }}} cmdline parsing */

    /* initialize random number generator */
#ifdef DEBUG
    srand(1);
#else
    srand(time(NULL));
#endif
    /* compute parameters from input parameters */
```

```
    m_g     *= m_u;                          /* from atomic mass units to kilograms */
    lambda   = fabs(lambda);                 /* make mean free path absolute */
    v_T      = sqrt( 2 * k_B * T_g / m_g);   /* thermal speed */
    if (T_g != 0.0)                          /* acceleration set from cmdline is overwrited */
        acceleration = alpha * v_T * v_T / lambda;
    abs_acc  = fabs(acceleration);           /* absolute value of acceleration */
    E        = m_g * acceleration / e_charge; /* intensity of electric field */

    /* normalisation constants */
    Dimens_x = lambda;
    Dimens_t = sqrt(lambda / abs_acc);
    if (acceleration == 0.0)
        Dimens_v = 1.0; /* TRIVIAL case */
    else
        Dimens_v = sqrt(abs_acc * lambda);

    /* theoretical drift speed int time = oo */
    v_drift = signum(acceleration) * sqrt(2 / M_PI) * Dimens_v;

    /* recompute parameters from dimensionless to real */
    start_speed *= Dimens_v;
    max_time    *= Dimens_t;
    d_time      *= Dimens_t;

#ifdef DISTRIB
    Nt = ceil(max_time / d_time) + 1;
    d_f_max_x = f_max_x * Dimens_v * max_time / Nx;
    d_f_max_v = f_max_v * Dimens_v / Nx;

    xmalloc(long int *, Fx_plus, Nt);
    xmalloc(long int *, Fv_plus, Nt);
    xmalloc(long int *, Fx_minus, Nt);
    xmalloc(long int *, Fv_minus, Nt);
    for (j = 0; j < Nt; j++) {
        xmalloc(long int, Fx_plus[j], Nx);
        xmalloc(long int, Fv_plus[j], Nx);
        xmalloc(long int, Fx_minus[j], Nx);
        xmalloc(long int, Fv_minus[j], Nx);
    }
#else
    Nb = ceil(max_time / d_time);
    xmalloc(Real_t, X, Nb);
    xmalloc(Real_t, U, Nb);
#endif
    print_parameters();
    if (help)
        return 1;

    /* loop over all ions, one by one ... */             /* [LINE a] */
    for (i = 0; i < N; i++) {
        /* If start_speed is not set, ions must start with thermal speed */
        if (start_speed == 0.0)
            v = random_thermal_speed();
        else
            v = start_speed;
        t = x = 0.0;
        /* index must be reseted, because we are
         * going from start with every ion
         */
        j = 0;

        real_time = d_time;
        rand_time = random_time(v);
        time_left = d_time;

        /* move with ion while real time is < max. time */
        while (real_time < max_time) {
            if ((t + rand_time) > max_time)
```

```
                rand_time = max_time - t;
            /* FIRST part of algoritmus */                 /* [LINE c] */
            /* run while rand_time > time step d_time */
            while ((t + rand_time) > real_time) {
                _DEBUG_MSG(2, "X.1.X");
                x    += ((1.0 / 2.0) * acceleration * time_left + v) * time_left;
                v    += acceleration * time_left;
                j++;
#ifdef DISTRIB /* Distribution function or drift speed {{{ */
                /* compute distribution function if DISTRIB defined ... */
                if (x < 0) {
                    idx = floorl(-x / d_f_max_x);
                    if (idx < Nx) {
                        Fx_minus[j][idx] += 1;
                    }
                }
                else {
                    idx = floorl(x / d_f_max_x);
                    /* fprintf(stderr, "# x_idx = %ld\n", idx); */
                    if (idx < Nx) {
                        /* fprintf(stderr, "x j = %ld\n", j); */
                        Fx_plus[j][idx] += 1;
                    }
                }
                if (v < 0) {
                    idx = floorl(-v / d_f_max_v);
                    if (idx < Nx) {
                        Fv_minus[j][idx] += 1;
                    }
                }
                else {
                    idx = floorl(v / d_f_max_v);
                    /* fprintf(stderr, "# v_idx = %ld\n", idx); */
                    if (idx < Nx) {
                        /* fprintf(stderr, "v j = %ld\n", j); */
                        Fv_plus[j][idx] += 1;
                    }
                }
#else
                /* ... otherwise compute drift sped */
                X[j] += x;
                U[j] += v;
#endif /* }}} */
                _DEBUG_MSG(1, "%g    %g    %g\n", real_time, x, v);
                _DEBUG_MSG(2, "YYYY t = %g real_time = %g rand_time = %g time_left = %g\n",
                        t, real_time, rand_time, time_left);
                real_time += d_time;    /* increase real time */
                t         += time_left;
                rand_time  = rand_time - time_left;        /* [LINE h] */
                /* time to next point "time_left" from SECOND part "X.2.X"
                 * is taken into account, run next loop with step d_time
                 * time_left is recomputed after FIRST part of algoritmus
                 */
                time_left = d_time;
            }                                              /* [LINE d] */

            time_left = t + rand_time - real_time + d_time;

            /* If intervals between 2 collision are < time step, we need many
             * of these intervals for one time step
             */
            /* SECOND part of algoritmus */                /* [LINE e] */
            do {
                _DEBUG_MSG(2, "X.2.X");
                x   += ((1.0 / 2.0) * acceleration * time_left + v) * time_left;
                t   += time_left;
                /* v   += acceleration * time_left; */
                v    = random_thermal_speed();
```

```

                rand_time = random_time(v);
                time_left = rand_time;

                _DEBUG_MSG(2, "ZZZZ t = %g real_time = %g rand_time = %g time_left = %g\n",
                        t, real_time, rand_time, time_left);
            } while ((t + rand_time) < real_time           /* [LINE f] */
                    && real_time < max_time);
            _DEBUG_MSG(2, "t = %g real_time = %g rand_time = %g time_left = %g\n",
                    t, real_time, rand_time, time_left);

            time_left = real_time - t;                     /* [LINE g] */
        } /* end of loop over time */
    } /* end of loop over ions */                          /* [LINE b] */


    /* Data output {{{ */
    /* Print data to stdout with full description,
     * parameters are already dumped
     */
#ifdef DISTRIB
    for (k = 1; k < Nt; k += output_step) {
        fprintf(stdout,
                "\n## Data format    : 1:real_time 2:relat_time  "
                "3:x   4:F(x>0,t)   5:F(x<0,t)   "
                "6:v   7:F(v>0,t)   8:F(v<0,t)\n");
        for (l = 0; l < Nx; l++)
            fprintf(stdout,
                    "%.5g""\t%.5g"  "\t%.5g""\t%.8g""\t%.8g"  "\t%.5g""\t%.8g""\t%.8g\n",
                    k * d_time, k * d_time / Dimens_t,
                    l * d_f_max_x, (1.0 * Fx_plus[k][l]) / N, (1.0 * Fx_minus[k][l]) / N,
                    l * d_f_max_v, (1.0 * Fv_plus[k][l]) / N, (1.0 * Fv_minus[k][l]) / N);
        fprintf(stdout, "\n"); /* line separator for gnuplot */
    }
#else
    fprintf(stdout,
            "\n## Data format    : 1:real_time   2:relat_time   "
            "3:(x / time / N)   4:(v_drift / N)   5:x   6:u_drift\n");
    fprintf(stdout,
            "%.5g" "\t%.5g" "\t%.8g" "\t%.8g" "\t%.8g" "\t%.8g\n",
            0.0, 0.0, start_speed, start_speed, 0.0, start_speed / Dimens_v);
    fflush(stdout);
    for (k = 1; k < Nb; k++) {
        fprintf(stdout,
                "%.5g" "\t%.5g" "\t%.8g" "\t%.8g" "\t%.8g" "\t%.8g\n",
                k * d_time, k * d_time / Dimens_t,
                X[k] / N / k / d_time, U[k] / N, X[k] / N,
                U[k] / N / Dimens_v);
    }
#endif
    /* }}} */

    return 0;
}

/* Modeline for text editor Vim:
 * vim: ts=4
 * vim600: fdm=marker fdc=3
 */
```