UNIVERZITET U BEOGRADU

ELEKTROTEHNIČKI FAKULTET

Master rad

ATOMSKA I ELEKTRONSKA STRUKTURA GRANICA IZMEĐU KRISTALNIH DOMENA U NAFTALENU

Mentor:
prof. dr Jelena Radovanović

Student:
Marko Mladenović
Broj indeksa: 2011/3149

Beograd, septembar 2012.

Ovaj master rad je rađen u Laboratoriji za primenu računara u nauci Instituta za fiziku Beograd. Želim da se zahvalim ovom prilikom svojim mentorima dr Igoru Stankoviću i dr Nenadu Vukmiroviću, koji su rukovodili izradom ovog rada, pružajući mi sve vreme nesebičnu pomoć. Takođe, dugujem zahvalnost i ostalim članovima laboratorije na savetima i podršci. Na kraju, želim da se zahvalim svojoj porodici, prijateljima i svim svojim profesorima, koji su doprineli da danas budem tu gde je jesam.

U Beogradu,
septembar 2012.

Sadržaj

1 Uvod

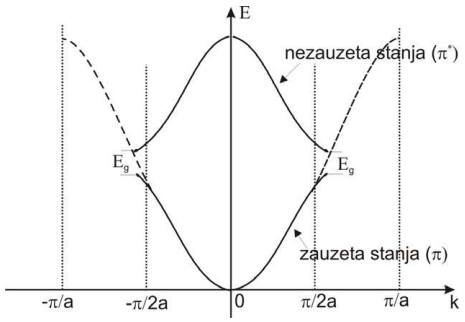
Ideja o korišćenju organskih materijala kao aktivnih elemenata za elektronske uređaje javila se samo nekoliko godina nakon što su napravljeni prvi klasični elektronski uređaji, ali malo toga je realizovano, prvenstveno zbog uspeha i nadmoći elektronske idustrije bazirane na silicijumu [1, 2, 3]. Istraživanje organskih poluprovodnika intenzivirano je devedesetih godina prošlog veka iz dva razloga. Prvo, eksperimenti su pokazali da je moguće napraviti organske svetleće diode (LED) [4, 5, 6, 7] i tranzistore na bazi tankih filmova (TFT) [8]. Takođe, razvijeni su i prototipovi organskih fotovoltaičnih naprava [9]. Drugo, u isto vreme javili su se zahtevi za jeftinim poluprovodničkim uređajima, zbog većeg korišćenja postojećih elektronskih naprava i potrebe za novim fotovoltaičnim ćelijama i energetski efikasnim svetlećim napravama. Organske LED se mogu naći na tržištu, dok je razvoj TFT-a i fotovoltaičnih uređaja još uvek predmet fundamentalnih istraživanja, koja uključuju razvoj novih materijala, proizvodnje i arhitekture uređaja. Ovi materijali su jeftini za proizvodnju i veoma su savitljivi. Mane su: još uvek mala pokretljiovst nosilaca, mala kvantna efikasnost (procenat konverzije upadne sunčeve svetlosti u električnu energiju) i kratak vek trajanja.

Postoje dve klase materijala od kojih se mogu praviti organski poluprovodnci: polimerni i na bazi malih molekula. U organske poluprovodnike na bazi malih molekula, koji mogu biti amorfni i kristalni, spadaju takozvani aromatični ugljovodonici kao što su naftalen, antracen, tetracen, pentacen, rubren, i njihovi derivati, dok u polimerne spadaju polivinil, poliacetilen i drugi. Kristalni materijali imaju veću pokretljivost nosilaca od polimernih. Maksimalna pokretljivost nosilaca (šupljina konkretno) je eksperimentalno utvrđena u rubrenu i iznosi 40 $cm^2 s^{-1} V^{-1}$, što je ipak manje od pokretljivosti nosilaca u silicijumu. Pentacen je materijal kome se poklanja najveća pažnja zbog malog energetskog procepa ($0.874eV$), što ga čini dobrim kandidatom za primenu u elektronskim napravama.

U radu je prikazan razvijeni metod koji daje atomsku i elektronsku strukturu kristalnih organskih poluprovodnika. Ovi materijali su u realnosti polikristali, što znači da u njima postoji veliki broj kontaktnih površina između monokristala različite kristalne rešetke. Poznato je da upravo te kontaktne površine ograničavaju transportne osobine materijala, ali nije poznato kako. Razvijeni model je primenjen na proračunu atomske i elektronske strukture na granici dva kristalna domena u polikristalu naftalena. Naftalen je izabran zato što je najjednostaniji molekul iz klase kristalnih organskih poluprovodnika na bazi malih molekula, a identičan postupak se može primeniti i na druge materijale. Atomska struktura se dobija uz pomoću Monte Karlo algoritma, dok se elektronska struktura računa pomoću metoda krpljenja naelektrisanja, koji je baziran na teoriji funkcionala gustine. U radu su najpre date teorijske osnove ovih metoda, a potom i rezultati dobijeni uz pomoć njih.

2 Kristalni organski poluprovodnici

Za osobine organskih poluprovodnika odgovorna je π zona, koja je podeljena na dve zone - popunjenu π (valentnu) i nepopunjenu π^* (provodnu) zonu. Između ove dve zone nastaje energijski procep, kao što je prikazano na slici 1. Najviše popunjeno mesto u valentnoj zoni i najniže prazno mesto u provodnoj zoni se označavaju sa HOMO i LUMO, respektivno.

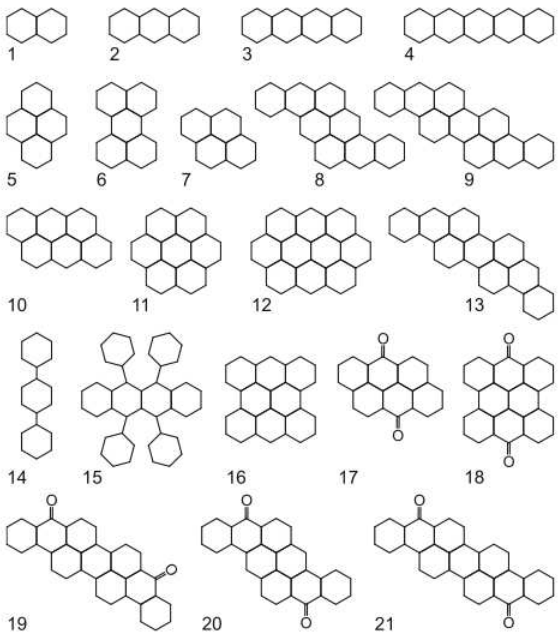


Slika 1: Zonska struktura organskih poluprovodnika

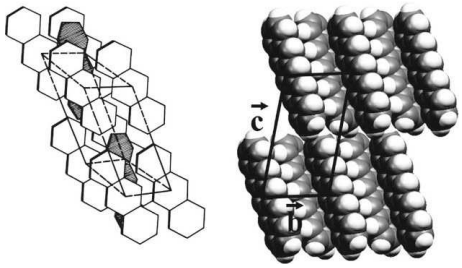
Molekuli nekih kristalnih organskih poluprovodnika prikazani su na slici 2. Njihova osnovna jedinica je benzenov prsten. Jedinična ćelija kristalnih organskih poluprovodnika je uglavnom monoklinička i triklinička [10]. Primer monokliničkog materijala je antracen (slika 3, levo), a primer trikliničkog materijala je pentacen (slika 3, desno). Parametri jedinične ćelije su dužine a, b i c, koje predstavljaju rastojanja između centara mase molekula u okviru jedne ćelije (tj. dužine stranica paralelopipeda jedinične ćelije) i uglovi α, β i γ koji predstavljaju uglove paralelopipeda. U monokliničkoj ćeliji dva ugla su jednaka pravom uglu, a kod trikliničke mogu imati proizvoljne vrednosti. Sa slike 3 vidimo da postoje dve grupe molekula u jednom kristalu. Dva molekula iz različitih grupa su zarotirani za neki ugao (npr. za pentacen taj ugao iznosi 52°), dok su molekuli iz istih grupa međusobno paralelni. U tabeli 1 dati su parametri kristalne rešetke naftalena, čija je atomska i elektronska struktura predmet ovog rada.

Materijal	Naftalen
temperatura topljenja (°C)	80
tip ćelije	monoklinička
a (nm)	0.824
b (nm)	0.600
c (nm)	0.866
α°	90
β°	122.9
γ°	90

Tabela 1: Parametri jednične ćelije kristalne rešetke naftalena



Slika 2: Kristalni organski poluprovodnici: 1 naftalen, 2 antracen, 3 tetracen, 4 pentacen, 5 piren, 6 perilen, 7 krislen, 8 pirantren, 9 izoviolantren, 10 antraten, 11 koronen, 12 ovalin, 13 violantren, 14 p-terfenil, 15 rubren, 16 m-dinaftantren, 17 antatron, 18 m-dinaftantron, 19 violantron, 20 pirantron, 21 izoviolantron



Slika 3: Jedinična ćelija kristalne rešetke antracena(levo) i pentacena(desno)

Organski poluprovodnici doživeli su najveću komercijalizaciju u organskim svetlećim diodama u televizorima. OLED televizori ne zahtevaju pozadinsko osvetljenje pa su značajno tanji od LCD ili LED televizora. Ukupna vrednost OLED ekrana na tržištu 2015. godine je projektovana na 4 440 miliona dolara. Što se tiče primene u tranzistorima, najdalje su otišli tranzistori sa efektom polja na bazi rubrena, ostvarivši pokretljivost od 40 $cm^2 s^{-1} V^{-1}$. Razvoj solarnih ćelija na bazi organskih poluprovodnika traje poslednjih 10-tak godina. Za razliku od drugih tehnika, ove solarne ćelije stalno ostvaruju sve veću efikasnost. Poslednji rekord postavila je nemačka kompanija Heliatek. On iznosi 10.7% i to upravo za organske poluprovodnike na bazi malih molekula. U tabeli 2 date su talasne dužine maksimalne apsorpcije za neke kristalne organske materijale.

materijal	talasna dužina
naftalen	315 nm
antracen	380 nm
tetracen	480 nm
pentacen	580 nm

Tabela 2: Talasne dužine maksimalne apsorpcije nekih organskih materijala

3 Monte Karlo simulacije

3.1 Uvod

Monte Karlo simulacije su često korišćena tehnika simulacija u matematici i fizici. U svojoj osnovi imaju ponavljanje generisanja slučajne konfiguracije, sve dok se ne uđe u ravnotežno stanje. Na taj način se mogu izračunati mnoge veličine koje se ne mogu odrediti egzaktno. Monte Karlo metod su razvili von Nojman, Ulam i Metropolis za vreme Drugog svetskog rata proučavajući difuziju neutrona u fisionom materijalu. Ime je dobio po kazinu u Monte Karlu u kome je Ulamov stric često gubio svoj novac. Koristi se najčešće za opis ponašanja sistema sa mnogo stepeni slobode (određivanje temperature topljenja kristala, određivanje strukture polimera,...) Takođe, često se koristi i za procenu kretanja na finansijskom tržistu.

3.2 Teorijske osnove

Neka je H hamiltonijan sistema i A fizička veličina koji zelimo da izračunamo. Srednja vrednost veličine A se u statističkoj fizici definiše na sledeći način:

$$\langle A \rangle = \frac{1}{Z} \int \exp(-\beta H) A(\Gamma) d\Gamma \tag{1}$$

gde je $\beta = \frac{1}{k_B T}$, a Z je statistička suma definisana sa: $Z = \int exp(-\beta H) d\Gamma$. Neka je $p_i(t)$ verovatnoća da se sistem u trenutku t nalazi u stanju i . Takođe, neka je $\pi(i \to f)$ verovatnoća da se sistem iz stanja i pomeri u stanje f u intervalu vremena dt. Broj prelazaka iz stanja i u f u jedinici vremena mora biti jednak broju prelazaka sistema iz stanja f u stanje i. Ovaj uslov se naziva uslovom detaljnog balansa:

$$p_f \pi(f \to i) = p_i \pi(i \to f) \tag{2}$$

U statističkoj fizici je verovatnoća da se sistem nađe u nekom stanju data Bolcmanovom težinom:

$$p_f = \frac{1}{Z} \exp[-\beta H(f)] \tag{3}$$

Uvrštavanjem u uslov detaljnog balansa dobijamo:

$$\exp[-\beta H(f)]\pi(f \to i) = \exp[-\beta H(i)]\pi(i \to f) \tag{4}$$

Verovatnoća prelaska iz jednog u drugo stanje se može razdvojiti na dva dela:

$$\pi(i \to f) = T(i \to f)A(i \to f) \tag{5}$$

gde je T matrica verovatnoće izbora finalnog stanja, dok je A matrica verovatnoće da se promena stanja prihvati. Matrica T je simetrična, pa uslov detaljnog balansa dobija sledeći oblik:

$$p_i A(i \to f) = p_f A(f \to i) \tag{6}$$

$$\frac{A(f \to i)}{A(i \to f)} = \frac{p_i}{p_f} = \exp\{-\beta[H(i) - H(f)]\} \tag{7}$$

Postoji više načina za definisanje matrice A. Najpopularniji način je predložio Metropolis, po kome je matrica A definisana na sledeći način:

$$A(i \to f) = \begin{cases} \exp\{-\beta[H(f) - H(i)]\} & , H(f) > H(i) \\ 1 & , H(f) \leq H(i) \end{cases} \tag{8}$$

Ukoliko se energija sistema smanjuje promenom konfiguracije, promena se prihvata, ukoliko ne, prihvata se sa verovatnoćom jednakom odnosu Bolcmanovih težina finalnog i inicijalnog stanja.

3.3 Monte Karlo algoritam

Neka je T broj Monte Karlo koraka simulacije, a N broj čestica u sistemu. Algoritam Monte Karlo simulacije je prikazan na slici 4.

Najpre se generiše potpuno slučajna konfiguracija sistema. Potom se ulazi u ciklus od 1 do T, a u okviru njega u ciklus od 1 do N. U okviru drugog ciklusa najpre se računa energija sistema. Potom se na slučajan način izabere jedna čestica koja se pomeri za neko slučajno odabrano rastojanje. Nakon pomeranja čestice računa se energija nove konfiguracije sistema. Nova konfiguracija se prihvata ukoliko je ispunjen Metropolisov uslov [11, 12].

Simulaciju vršimo tako sto neke veličine držimo konstantnim, posmatramo kako se ponašaju ostale veličine i računamo njihovu srednju vrednost i fluktuacije oko te srednje vrednosti. Razlikujemo četiri tipa Monte Karlo simulacija (ansambla):

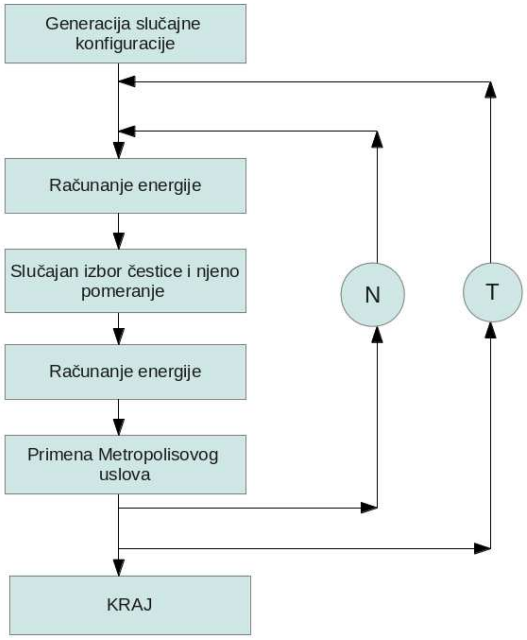
1) mikrokanonskna NVE simulacija (konstantni broj čestica, zapremina i energija)

2) NVT simulacija (konstantni broj čestica, zapremina i temperatura)

3) NPT simulcija (konstantni broj čestica, pritisak i temperatura)

4) μVT simulacija velikog kanonskog ansambla (konstantni hemijski potencijal, zapremina i temperatura).

Srednje vrednosti izračunate kao rezultat 2 različite simulacije su jednake.



Slika 4: Algoritam Monte Karlo simulacije

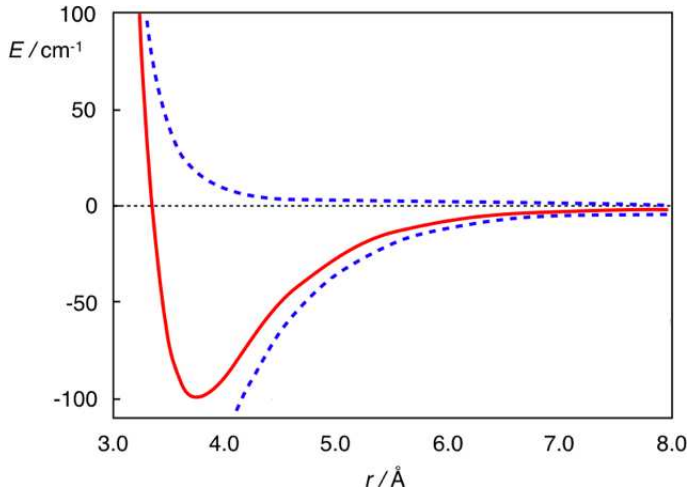
3.4 Potencijal Lenard Džonsa

Da bismo opisali ponašanje sistema koji sadrži mnogo čestica, moramo na adekvatan način opisati interakciju između čestica. To se radi izborom potencijala. Potencijal koji se najčešće koristi u Monte Karlo simulacijama je potencijal Lenard Džonsa:

$$v^{LJ}(r) = 4\epsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}] \tag{9}$$

Oblik ovog potencijala dat je na slici 5.

Parametar ϵ predstavlja apsolutnu vrednost potencijala u tački minimuma i naziva se dubina jame, dok parametar σ predstavlja rastojanje između čestica na kome je potencijal interakcije između njih jednak 0. Često se u simulacijama koristi skalirani potencijal Lenard Džonsa u kome ne figurišu parametri ϵ i σ. Sa slike vidimo da je za rastojanja manja od σ potencijal pozitivan i može biti jako veliki. Zato se uvodi parametar r_{min} koji obično uzima vrednosti od 0.7σ do 0.9σ i predstavlja minimalno rastojanje na kome se računa potencijal Lenard Džonsa. Za manja rastojanja uzima se da potencijal ima konstantnu vrednost. Takođe, vidimo da za vrednosti rastojanja veće od 2σ potencijal postaje jako mali (reda veličine 10^{-3}) u odnosu na potencijal u tački minimuma. Zato je zgodno uvesti parametar r_C koji predstavlja rastojanje na kome je potencijal jednak 0 i za sva veća rastojanja takođe ima vrednost 0. Time ubrzavamo simulaciju, a ne utičemo na tačnost rezultata. Ukoliko koristimo skalirani potencijal Lenard Džonsa ($\epsilon = \sigma = 1$) onda je potrebno i ostale veličine koje koristimo u simulaciji ili ih računamo skalirati u



Slika 5: Potencijal Lenard Džonsa

Veličina	Formula
gustina	$\rho\sigma^3$
temperatura	k_BT/ϵ
pritisak	$P\sigma^3/\epsilon$
vreme	$t\sqrt{\epsilon/(m\sigma^2)}$
sila	$f\sigma/\epsilon$

Tabela 3: Veličine u Lenard Džons jedinicama. Množenjem fizičke veličine odgovarajućom formulom iz tabele dobija se veličina u jedinicama Lenard Džonsa.

takozvane Lenard Džons jedinice. U tabeli 3 prikazane su formule pomoću kojih se veličine koje se najčesće koriste u simulacijama skaliraju u jedinice Lenard Džonsa [11].

Parametri σ i ϵ se razlikuju za različite čestice. Ukoliko radimo simulaciju molekula koji u sebi sadrži različite atome ili pseudoatome parametre σ i ϵ tada računamo na sledeći način:

$$\sigma_{AB} = \frac{1}{2}(\sigma_A + \sigma_B) \tag{10}$$

$$\epsilon_{AB} = \sqrt{\epsilon_A \epsilon_B} \tag{11}$$

gde su A i B različiti atomi ili pseudoatomi. U tabeli 4 date su vrednosti parametara σ i $\frac{\epsilon}{k_B}$ za neke atome i grupe atoma [13, 14].

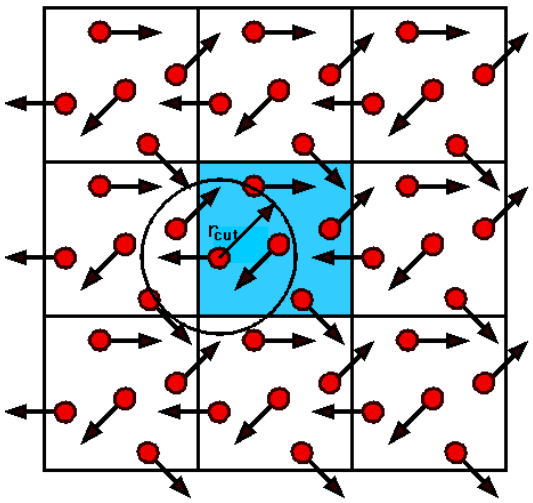
3.5 Tehnike za poboljšanje simulacije

Monte Karlo simulacije se izvode na malom broju čestica, izmedju 10 i 10000. Maksimalna veličina sistema koga simuliramo određena je performansama računara na kome

Atom	σ(nm)	ϵ/k_B(K)
H	0.281	8.6
C	0.335	51.2
O	0.295	61.6
Ar	0.341	119.8
Kr	0.383	164.0
CH	0.37	50

Tabela 4: Parametri Lenard Džons potencijala za neke atome i grupe atoma

vršimo simulaciju, dominantno memorijom i brzinom procesora. Sistem se obično stavlja u kutiju. S obzirom da u simulaciji vršimo pomeranje čestica, može se desiti da čestica izađe iz kutije. Takođe, na ivicama kutije dolaze do izražaja efekti površine. Da bi se neželjene posledice ograđivanja sistema izbegle, primenjuju se periodični granični uslovi. Oko kutije se dodaju njene kopije tako da sistem postaje beskonačan. Svaka čestica iz originalne kutije ima svoju kopiju u svakoj kutiji. Ukoliko čestica izađe iz originalne kutije, zahvaljujući periodičnim graničnim uslovima, njena kopija iz susedne kutije ce ući u originalnu kutiju i na taj način sprečavamo gubitak čestice. Ilustracija periodičnih graničnih uslova je data na slici 6.



Slika 6: Ilustracija periodičnih graničnih uslova

Sa slike vidimo da čestica može da interaguje i sa česticama iz drugih kutija ukoliko se one nalaze u krugu (za 2D slučaj kao na slici) poluprečnika r_C . Stavljanjem uslova da dimenzija kutije mora biti minimalno jednaka $2r_C$, obezbeđujemo da čestica interaguje sa drugim česticama samo jednom, bilo sa originalnom česticom, bilo sa nekom njenom kopijom, u zavisnosti od toga koja je najbliža.

Ako je broj čestica u sistemu koji simuliramo veći od 1000, računanje interakcije svaka dva para čestica postaje složeno i vremenski zahtevno. Jedan od najboljih načina za ubrzanje simulacije sistema velikog broja čestica je podela simulacione kutije na ćelije jednakih dimenzija. Dimenzija ćelija mora biti veća od r_C. Kada se čestica pomeri, ispituje se promena energije u ćeliji u kojoj se čestica nalazi i u okolnim ćelijama. Čestica tokom trajanja simulacije može izaći iz svoje ćelije, zato je potrebno vršiti raspored čestica po ćelijama u svakom koraku simulacije ili periodično.

3.6 Termodinamičke veličine

Energiju čestice definišemo klasično, kao zbir kinetičkog i potencijalnog člana:

$$E = \langle K \rangle + \langle \Phi \rangle \tag{12}$$

Srednju vrednost kinetičke energije definišemo preko temperature:

$$\langle K \rangle = \frac{3}{2} N k_B T, \tag{13}$$

dok je potencijalna energija jednostavno zbir svih potencijala koji potiču od drugih čestica. Pritisak računamo uz pomoć modifikovane jednačine idealnog gasa:

$$PV = N k_B T + \langle W \rangle, \tag{14}$$

gde je W takozvani virial i definiše se na sledeći način:

$$\langle W \rangle = \frac{1}{3} \sum_{i=1}^{N} \overrightarrow{r_i} \overrightarrow{f_i}, \tag{15}$$

$\overrightarrow{f_i}$ je sila koja deluje na i-tu česticu. Pritisak, dakle, računamo po formuli:

$$P = \rho k_B T + \frac{\langle W \rangle}{V} \tag{16}$$

Virijalna komponenta pritiska, ukoliko koristimo potencijal Lenard Džonsa iznosi:

$$W^{LJ}(r) = 16\epsilon[(\frac{\sigma}{r})^{12} - 0.5(\frac{\sigma}{r})^{6}] \tag{17}$$

Fluktuacije neke veličine su nam bitne iz dva razloga. Prvi je taj što posmatranjem vrednosti fluktuacije možemo da utvrdimo da li se sistem nalazi u termodinamičkoj ravnoteži. Drugi razlog je sto pomoću fluktuacija računamo neke veličine kao što je na primer toplotni kapacitet. Generalno, fluktuacija neke veličine A se definiše kao:

$$o(A) = \langle \delta A^2 \rangle = \sqrt{\langle A^2 \rangle - (\langle A \rangle)^2} \tag{18}$$

Toplotni kapacitet pri konstantnoj zapremini računamo na sledeći način:

$$C_v = \frac{3}{2}N + \frac{\langle \Phi^2 \rangle_{NVT}}{T} \tag{19}$$

Koeficijent termalnog pritiska defnišemo kao:

$$\gamma_V = \frac{N + \frac{\langle \delta\Phi\delta P \rangle_{NVT}}{T^2}}{V} \tag{20}$$

Koeficijent izotermalne kompresibilnosti određujemo pomoću formule:

$$\beta_T = \frac{\langle \delta V \rangle_{NPT}}{VT} \tag{21}$$

Konačno, koeficijent termalne ekspanzije možemo računati po formuli:

$$\alpha_P = \beta_T \gamma_V \tag{22}$$

Ukoliko vršimo NVT simulaciju, u Metropolisovom uslovu umesto hamiltonijana sistema možemo koristiti srednju potencijalnu energiju, jer je kinetička energija konstantna. Sa druge strane, kod NPT simulacije Metropolisov uslov je malo drugačiji. Tada osim promene energije, posmatramo i promenu zapremine sistema. Ispitujemo veličinu:

$$\Delta = \beta[H(f) - H(i) + P(V(f) - V(i)) - N\log(\frac{V(f)}{V(i)})] \tag{23}$$

Ako je Δ negativno tada se prihvata promena konfiguracije, ukoliko nije, prihvata se sa verovatnoćom $\exp(-\Delta)$.

3.7 Strukturne veličine

Često želimo da ispitamo strukturu sistema koji simuliramo. Na primer, ukoliko imamo neki materijal, želimo da ispitamo kakvu strukturu čine molekuli tog materijala, kristalnu ili amorfnu. Za opis rasporeda čestica (atoma, molekula, ...) koristimo funkciju raspodele parova $g(r)$. Ova funkcija predstavlja verovatnoću nalaženja para čestica na međusobnom rastojanju r u odnosu na neku slučajnu konfiguraciju (broj čestica koje se nalaze na rastojanju r u odnosu na broj na istom rastojanju pri istoj gustini u idealnom gasu). Računa se uz pomoć formule:

$$g(r) = \frac{V}{N^2}\langle \sum_i \sum_{j \neq i} \delta(r - r_{ij}) \rangle \tag{24}$$

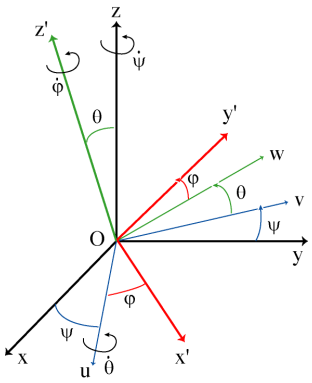
Funkcija raspodele parova nije samo korisna za opis strukture sistema, već se pomoću nje mogu izračunati neke bitne veličine kao što su energija, pritisak, hemijski potencijal... Ukoliko simuliramo molekule, funkcija raspodele parova se može proširiti, tako što bi se posmatrala i orijentacija molekula opisana sa dva Ojlerova ugla ϕ i θ.

Uređenost sistema možemo opisati i parametrom koji se naziva parametar translatornog uređenja, koji u stvari predstavlja Furijeovu transformaciju gustine:

$$\rho(\overrightarrow{k}) = \frac{1}{N}\sum_{i=1}^{N}\cos(\overrightarrow{k}\,\overrightarrow{r_i}), \tag{25}$$

gde je $\overrightarrow{k} = [(2N)^{1/3}\pi/L](1,1,1)$, gde je L dužina kutije. Za kristalnu strukturu ovaj parametar ima vrednost 1.

Ukoliko simuliramo molekule, možemo posmatrati kako je sistem uređen preko parametra orijentacije molekula, takozvanog direktora. Pretpostavimo da je orijenatacija molekula definisana uglovima ϕ i θ kao na slici 7. Definišemo 3 komponente direktora za svaku ko-



Slika 7: Ojlerovi uglovi

ordinantnu osu:

$$\overrightarrow{u_x} = \cos\phi\sin\theta\overrightarrow{e_x} \tag{26}$$

$$\overrightarrow{u_y} = \sin\phi\sin\theta\overrightarrow{e_y} \tag{27}$$

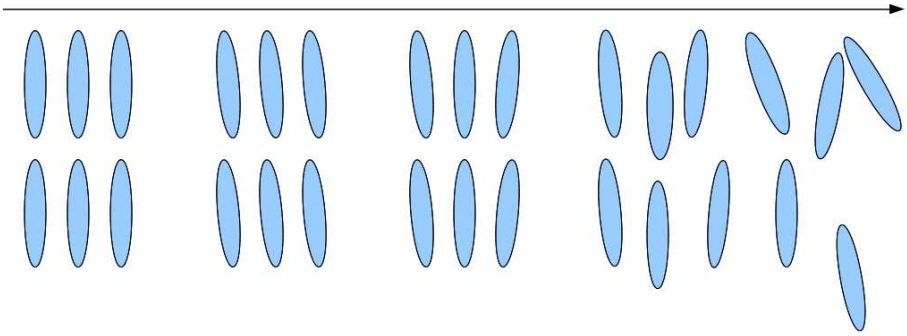
$$\overrightarrow{u_z} = \cos\theta\overrightarrow{e_z} \tag{28}$$

Da bismo opisali koliko je sistem uređen računamo parametar uređenja. Najpre formiramo matricu dimenzija 3×3:

$$Q_{\alpha\beta} = \frac{1}{2N}\sum_{i}(3\overrightarrow{u_{\alpha i}}\overrightarrow{u_{\beta i}} - \delta_{\alpha\beta}), \tag{29}$$

gde su α i β sve kombinacije x, y i z, a $\delta_{\alpha\beta}$ Kronikerova delta funkcija. Dijagonalizujemo matricu Q i dobijemo njene sopstvene vrednosti (njih 3). Maksimalna sopstvena vrednost predstavlja parametar uređenja S [15]. Može imati vrednosti izmedju 0 i 1. Ukoliko je $S < 0.1$ tada kažemo da je sistem izotropan. Izotropan sistem je potpuno neuređen po pitanju direktora. To je karakteristika tečne faze. Ukoliko je $S = 1$ tada sistem ima savršenu kristalnu strukturu. Sistem se može smatrati uređenim ukoliko je $S > 0.4$. Na slici 8 prikazan je tranzicija iz čvrste u tečnu fazu. Između ove dve faze nalazi se međufaza

tečnih kristala. Ova faza se sastoji iz tri podfaze: smektičke C, smektičke A i nematičke faze. Struktura sistema u smektičkoj C fazi je gotovo ista kao u kristalnoj, samo što je direktor zaokrenut za neki ugao u odnosu na direktor u kristalu. U smektičkoj A fazi parametar uređenja opada. U nematičkoj fazi kristalna rešetka počinje da se narušava.



Slika 8: Tranzicija iz čvrste u tečnu fazu se sastoji od 5 podfaza(s leva na desno): kristalne, smektičke C, smektičke A, nematičke i izotropne.

Korelacija između dve merene veličine A i B se u statističkoj fizici definiše preko koeficijenta korelacije:

$$c_{AB} = \frac{\langle \delta A \delta B \rangle}{\sigma(A)\sigma(B)} \tag{30}$$

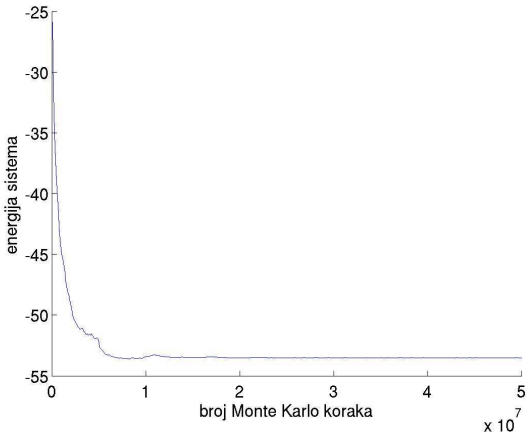
Švarcove nejednačine garantuju da će se vrednost koeficijenta korelacije nalaziti između 0 i 1, pri čemu vrednost bliska 1 znači visoku korelaciju.

3.8 Početno i ravnotežno stanje sistema

Inicijalnu konfiguraciju sistema možemo izabrati na dva načina. Najjednostavniji način je uzeti potpuno slučajni raspored čestica. U tom slučaju potrebno je iskoristiti dobar generator slučajnih brojeva. Problem sa ovim metodom je da se dve čestice mogu preklopiti, što će rezultirati visokim potencijalom. Drugi način je raspoređivanje čestica u rešetku. Mana ovog načina je što će tada biti potrebno više vremena za postizanje termodinamičke ravnoteže (TDR). Ukoliko simuliramo molekule treba odrediti i njihovu početnu orijentaciju. I ovde imamo dva izbora, potpuno slučajna orijentacija ili da molekuli na početku budu usmereni u zadatom pravcu.

Kada se pusti simulacija, potrebno je neko vreme da sistem uđe u stanje termodinamičke ravnoteže. Najbolji način da se utvrdi da li je sistem ušao u TDR, je da se posmatra potencijal ili pritisak. Ukoliko se krene od slučajne konfiguracije, potencijal i pritisak bi trebalo da padaju sve dok ne dođu do neke vrednosti oko koje osciluju. Ako se krene sa konfiguracijom rešetke tada bi trebalo da rastu pre nego što sistem uđe u stacionarno stanje. Takođe, ukoliko se pođe od rešetke, može se odrediti trenutak u kome sistem napušta kristalnu strukturu, to je onaj trenutak kada parametar translatornog

uređenja postane blizak 0. Problem koji se može javiti tokom simulacije je da sistem uđe u dvofazni režim između gasa i tečnosti [11]. Tada je potrebno mnogo vremena da se uđe u ravnotežno stanje. Ulazak sistema u dvofazni režim se manifestuje pojavom pika u funkciji raspodele parova. Može se pokazati da pri većim temperaturama sistem brže uđe u TDR nego pri manjim. Zato, ako želimo da što pre dobijemo sistem u TDR-u, možemo podeliti simulaciju na dva dela. Najpre vršimo simulaciju pri velikoj temperaturi, a potom smanjimo temperaturu na željenu vrednost. Ne može se unapred znati koliko je potrebno Monte Karlo koraka da bi sistem ušao u ravnotežu. Obično je potrebno do 10 000 000 koraka, pri čemu ako simuliramo sistem od N čestica, jedan Monte Karlo korak predstavlja N pomeranja. Na slici 9 prikazan je potencijal sistema u zavisnosti od broja Monte Karlo koraka. Nakon otprilike 10 miliona koraka sistem je ušao u ravnotežu.



Slika 9: Ilustracija ulaska sistema u ravnotežno stanje. Energija je data u bezdimenzionim Lenard - Džons jedinicama

3.9 Simulacije molekula

Najpre ćemo podeliti molekule na krute i promenljive. Oblik krutih molekula se ne menja i tokom simulacije molekul se kreće kao celina. Primeri takvih molekula su derivati benzena. Sa druge strane, promenljivi molekuli menjaju svoj oblik, dužinu veza, međusobni raspored između atoma ili grupa atoma. Takvi molekuli su polimeri.

Ponašanje krutih molekula opisano je centrom mase. Translaciju molekula vršimo tako što pomeramo centar mase duž sve tri koordinantne ose na sledeći način:

$$r_x^{'} = r_x + (2X - 1)\delta r_{max}, \tag{31}$$

$$r_y^{'} = r_y + (2X - 1)\delta r_{max}, \tag{32}$$

$$r_z^{'} = r_z + (2X - 1)\delta r_{max}, \tag{33}$$

gde je X slučajno generisani broj između 0 i 1, a δr_{max} maksimalni pomeraj duž osa. Parametar δr_{max} se menja, tj množi sa 1.05 ili 0.95 (ne obavezno tim vrednostima), kako bi broj uspešnih Monte Karlo koraka u odnosu na ukupan broj koraka stalno bio oko 0.5.

Postoji više načina za rotiranje molekula. Prvi je pridruživanje seta uglova centru mase. Najčešće se koriste već pomenuti Ojlerovi uglovi ϕ, θ i ψ. Ukoliko želimo da znamo poziciju svakog atoma ili grupe atoma u odnosu na centar mase, vektor položaja atoma $\overrightarrow{r_i}$ množimo sa sledećom matricom, takozvanom matricom rotacije:

$$\begin{pmatrix} \cos\phi\cos\psi - \sin\phi\cos\theta\sin\phi & \sin\phi\cos\psi + \cos\phi\cos\theta\sin\psi & \sin\theta\sin\psi \\ -\cos\phi\sin\psi - \sin\phi\cos\theta\cos\phi & \sin\phi\sin\psi + \cos\phi\cos\theta\cos\psi & \sin\theta\cos\psi \\ \sin\phi\sin\theta & -\cos\phi\sin\theta & \cos\theta \end{pmatrix}. \quad (34)$$

Pomeranje uglova vršimo na isti način kao i pomearnje duž koordinatnih osa:

$$\phi' = \phi + (2X-1)\delta\phi_{max}, \quad (35)$$

$$\theta' = \theta + (2X-1)\delta\theta_{max}, \quad (36)$$

$$\psi' = \psi + (2X-1)\delta\psi_{max}. \quad (37)$$

Verovatnoća prihvatanja pomeraja čestice sada iznosi:

$$\exp(-\beta(H(f) - H(i))\frac{\sin\theta_f}{\sin\theta_i}) \quad (38)$$

Ukoliko umesto θ pomeramo $\cos(\theta)$ tada je uslov prihvatanja isti kao i kada ne vršimo rotaciju molekula, definisan jednačinom (7). Uglovi ϕ i ψ uzimaju vrednosti iz intervala $[-\pi, \pi]$, dok ugao θ uzima vrednosti iz intervala $[0, \pi]$. Postoji nekoliko varijanti Ojlerovih uglova, u zavisnosti od toga kako se definišu uglovi i tada se matrica rotacije neznatno razlikuje od matrice (34).

Drugi način rotacije krutih molekula je rotacija oko fiksnih osa koordinatnog sistema. Prvo se na slučajan način izabere osa oko koje se vrši rotacija, a potom se molekul zarotira oko nje za neki slučajan ugao γ. Matrice rotacije oko koordinatnih osa x, y i z su redom:

$$A_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{pmatrix} \quad (39)$$

$$A_y = \begin{pmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{pmatrix} \quad (40)$$

$$A_z = \begin{pmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (41)$$

Matrica rotacije oko sve tri ose jednaka je proizvodu matrica A_x, A_y i A_z.

Simulacije promenljivih molekula (polimera) su potpuno drugačije. Najpre se formira inicijalna konfiguracija na dvodimenzionalnoj rešetki tako da molekul ne seče sam sebe. Tokom simulacije formiraju se parovi susednih atoma ili grupa atoma, koji se pomeraju jedan u odnosu na drugi promenom dužine veze između njih. Takođe, formiraju se tripleti atoma, koji se pomeraju tako što se menja ugao između veza. Konačno formiraju se kvarteti atoma, fiksira se srednja veza i vrši se rotacija spoljašnjih atoma oko te veze. Za sve parove, triplete i kvartete se računa potencijal interakcije po posebnim formulama, dok se interakcije između atoma ili grupa atoma koji ne pripadaju istom paru, tripletu ili kvartetu tretiraju najčešće potencijalom Lenard Džonsa.

4 Teorija funkcionala gustine

Teorija funkcionala gustine (Density functional theory, skraćeno DFT) predstavlja moćan alat za rešavanje višečestičnih problema u kvantnoj mehanici. Omogućuje da se komplikovana N-elektronska talasna funkcija i njoj pridružena Šredingerova jednačina zamene mnogo jednostavnijom elektronskom gustinom $\rho(r)$. Začetnici ove teorije su Tomas i Fermi[16], čiji će model biti prikazan u nastavku.

4.1 Tomas-Fermijev model

Podelimo prostor na ćelije oblika kocke dužine l i zapremine $\Delta V = l^3$. Neka se u svakoj ćeliji nalazi fiksni broj elektrona ΔN (koji može biti različit za različite ćelije) i neka se elektroni ponašaju kao fermioni na temperarturi 0K. Energija čestice u beskonačnoj kvantnoj jami data je formulom:

$$\varepsilon(n_x, n_y, n_z) = \frac{h^2}{8ml^2}(n_x^2 + n_y^2 + n_z^2) = \frac{h^2}{8ml^2}R^2, \tag{42}$$

gde su n_x, n_y i n_z kvantni brojevi. Za veće kvantne brojeve, broj različitih nivoa energije manje od ε može se aproksimirati zapreminom oktanta sfere radijusa R. Taj broj iznosi:

$$\Phi(\epsilon) = \frac{1}{8}(\frac{4\pi R^3}{3}) = \frac{\pi}{6}(\frac{8ml^2\varepsilon}{h^2})^{3/2}. \tag{43}$$

Broj nivoa u intervalu energija ε i $\varepsilon + \delta\varepsilon$ iznosi:

$$g(\varepsilon)\Delta\varepsilon = \frac{\pi}{4}(\frac{8ml^2}{h^2})^{3/2}\varepsilon^{1/2}\delta\varepsilon + O((\delta\varepsilon)^2), \tag{44}$$

gde je funkcija $g(\varepsilon)$ gustinja stanja na energiji ε. Da bi se izračunala totalna enegija elektrona u ćeliji potrebno je uvesti verovatnoću da neko stanje bude okupirano. Ova verovatnoća je data Fermijevom raspodelom:

$$f(\varepsilon) = \frac{1}{1 + e^{\beta(\varepsilon-\mu)}}, \tag{45}$$

gde je $\beta = \frac{1}{k_B T}$, a μ hemijski potencijal. Na tempereturi $0K$ Fermijeva raspodela postaje step funkcija:

$$f(\varepsilon) = \begin{cases} 1, & \varepsilon < \varepsilon_F \\ 0, & \varepsilon > \varepsilon_F \end{cases}, \tag{46}$$

gde je ε_F Fermijeva energija. Ukupna energija elektrona u ćeliji dobija se sumiranjem po energijskim nivoima:

$$\Delta E = 2\int \varepsilon f(\varepsilon) g(\varepsilon) d\epsilon = 4\pi (\frac{2m}{h^2})^{3/2} l^3 \int_0^{\varepsilon_F} \varepsilon^{3/2} d\varepsilon = \frac{8\pi}{5} (\frac{2m}{h^2})^{3/2} l^3 \varepsilon_F^{5/2}. \tag{47}$$

Faktor 2 se uzima zbog spinske degenracije. Broj elektrona u jednoj ćeliji je dat formulom:

$$\Delta N = 2\int f(\varepsilon) g(\varepsilon) d\epsilon = \frac{8\pi}{3} (\frac{2m}{h^2})^{3/2} l^3 \varepsilon_F^{3/2}. \tag{48}$$

Eliminacijom ε_F iz jednačine (47) i ubacivanjem u (48) dobijamo:

$$\Delta E = \frac{3}{5} \Delta N \varepsilon_F = \frac{3h^2}{10m} (\frac{3}{8\pi})^{2/3} l^3 (\frac{\Delta N}{l^3})^{5/3}. \tag{49}$$

Jednačina (49) predstavlja vezu između energije i gustine elektrona, jer je $\rho = \frac{\Delta N}{l^3}$. Sumiranjem po svim ćelijama dobija se izraz:

$$T_{TF}[\rho] = C_F \int (\rho)^{5/3} \overrightarrow{r}\, d\overrightarrow{r}, \quad C_F = \frac{3}{10} (3\pi^2)^{2/3} = 2.871. \tag{50}$$

Ukoliko se pored kinetičke energije uračuna elektron - jezgro i elektron - elektron interakcija dobija se konačan izraz za energiju atoma u funkciji gustine elektrona:

$$E_{TF}[\rho] = C_F \int (\rho)^{5/3} (\overrightarrow{r}) d\overrightarrow{r} - Z \int \frac{\rho(\overrightarrow{r})}{r} d\overrightarrow{r} + \frac{1}{2} \int\int \frac{\rho(\overrightarrow{r_1}) \rho(\overrightarrow{r_2})}{|\overrightarrow{r_1} - \overrightarrow{r_2}|} d\overrightarrow{r_1} d\overrightarrow{r_2}. \tag{51}$$

Izraz (51) naziva se Tomas - Fermijev funkcional energije za atome. Pretpostavimo sada da osnovno stanje gustine energije minimizuje funkcional energije uz uslov:

$$N = \int \rho(\overrightarrow{r}) d\overrightarrow{r}, \tag{52}$$

gde je N ukupna broj elektrona u atomu. Osnovno stanje gustine energije mora da zadovolji varijacioni princip:

$$\delta E_{TF}[\rho] - \mu_{TF} (\int \rho(\overrightarrow{r}) d\overrightarrow{r} - N) = 0, \tag{53}$$

koji vodi do Ojler - Lagranžove jednačine:

$$\mu_{TF} = \frac{\delta E_{TF}}{\delta \rho(\overrightarrow{r})} = \frac{5}{3} C_F \rho^{2/3} (\overrightarrow{r}) - \phi(\overrightarrow{r}), \tag{54}$$

gde je $\phi(\overrightarrow{r})$ elektrostatički potencijal koji potiče od jezgara i elektrona:

$$\phi(\overrightarrow{r}) = \frac{Z}{|\overrightarrow{r}|} - \int \frac{\rho(\overrightarrow{r_2})}{|\overrightarrow{r} - \overrightarrow{r_2}|} \overrightarrow{dr_2}. \tag{55}$$

Jednačine (51), (54) i (55) se samo - konzistentno rešavaju. Ovim je završen opis Tomas -Fermi modela za atome. Ovaj model je dugo bio u zapećku zbog slabe tačnosti u odnosu na druge metode. Situacija se promenila kada su Hohenberg i Kon objavili rad u kome su pokazali da se za osnovna stanja Tomas-Fermijev model može smatrati aproksimacijom tačne teorije, teorije funkcionala gustine.

4.2 Hohenberg - Konove teoreme

Za N-elektronski sistem eksterni potencijal $v(\overrightarrow{r})$ određuje hamiltonijan sistema dok je osnovno stanje određeno sa N i $v(\overrightarrow{r})$. Prva Hohenberg - Konova teorema opravdava korišćenje gustine elektrona $\rho(\overrightarrow{r})$ kao osnovne promenljive. Ona glasi: *Eksterni potencijal* $v(\overrightarrow{r})$ *je određen gustinom elektrona* $\rho(\overrightarrow{r})$ *do na trivijalnu aditivnu konstantu.* Kako ρ određuje broj elektrona, to određuje i osnovno stanje talasne funkcije, kao i sve ostale elektronske osobine sistema. Dokaz ove teoreme je veoma jednostavan i koristi samo princip minimalne energije osnovnog stanja. Pretpostavimo da dva različita eksterna potencijala v i v' imaju istu gustinu osnovnog stanja ρ. Ova dva potencijala određuju dva različita Hamiltonijana H i H' čije talasne funkcije Ψ i Ψ' mogu biti različite. Uzimimo Ψ' za probnu funkciju za H problem:

$$E_0 < \langle \Psi' | H | \Psi' \rangle = \langle \Psi' | H' | \Psi' \rangle + \langle \Psi' | H - H' | \Psi' \rangle = E_0' + \int \rho(\overrightarrow{r})[v(\overrightarrow{r}) - v'(\overrightarrow{r})] d\overrightarrow{r}, \tag{56}$$

gde su E_0 i E_0' energije osnovnog stanja za hamiltonijane H i H', respektivno. Takođe, uzmimo Ψ za probnu funkciju za H' problem:

$$E_0' < \langle \Psi | H' | \Psi \rangle = \langle \Psi | H | \Psi \rangle + \langle \Psi | H' - H | \Psi' \rangle = E_0 + \int \rho(\overrightarrow{r})[v'(\overrightarrow{r}) - v(\overrightarrow{r})] d\overrightarrow{r}. \tag{57}$$

Sabiranjem jednačina (56) i (57) dobijamo: $E_0 + E_0' < E_0' + E_0$, što je kontradikcija, dakle ne postoje dva različita eksterna potencijala koji imaju istu gustinu elektrona osnovnog stanja, čime je prva Hohenberg - Konova teorema dokazana. Gustina elektrona određuje sve osobine elektrona u osnovnom stanju, tako i kinetiču energiju $T[\rho]$ i potencijalne energije elektron - jezgro $V_{ne}[\rho]$ i elektron - elektron $V_{ee}[\rho]$ interkacije. Izraz za ukupnu energiju atoma se može napisati kao:

$$E[\rho] = T[\rho] + V_{ne}[\rho] + V_{ee}[\rho] = \int \rho(r) v(r) dr + F_{HK}[\rho], \tag{58}$$

gde je $F_{HK}[\rho] = T[\rho] + V_{ee}[\rho]$. $V_{ee}[\rho]$ se množe razdvojiti na dva člana:

$$V_{ee}[\rho] = J[\rho] + C[\rho], \tag{59}$$

gde $J[\rho]$ predstavlja klasičnu odbojnu interakciju, a $C[\rho]$ je takozvani "neklasični" član koji sadrži izmensko - korelacionu energiju. Druga Hohenberg - Konova teorema uvodi varijacioni princip za energiju. Ona glasi: *Za probnu gustinu elektrona* $\rho'(\overrightarrow{r})$*, za koju važi* $\rho'(r) \geq 0$ *i* $\int \rho'(\overrightarrow{r})d\overrightarrow{r} = N$*,* $E_0 \leq E[\rho']$*,* varijacioni princip zahteva sledeći uslov:

$$\delta E[\rho'] - \mu[\int \rho(\overrightarrow{r})d\overrightarrow{r} - N] = 0, \tag{60}$$

iz koga se izvodi Ojler - Lagranžova jednačina:

$$\mu = \frac{\delta E[\rho]}{\delta \rho} = v(\overrightarrow{r}) + \frac{\delta F_{HK}[\rho]}{\delta \rho}. \tag{61}$$

Izraz za $F_{HK}[\rho]$ nije moguće napisati u eksplicitnoj formi i zbog toga je potrebno primeniti aproksimacije. Jedna takva aproksimacija je aproskimacija lokalne gustine (local density aproximation, skraćeno LDA) o kojoj će biti više reči u odeljku 5.4.

4.3 Kon - Šamove jednačine

Tomas -Femijev model je pokazivao manjkavost u pogledu tačonsti. Da bi povećali tačnost elektronskog proračuna, Kon i Šam su predložili svoj model, koji se zasniva na uvođenju elektronskih orbitala. Na ovaj način kinetička energija može biti sračunata jednostavno i sa velikom tačnošću. Pođimo najpre od tačne formule za kinetičku energiju:

$$T = \sum_i^N n_i \langle \psi_i | -\frac{1}{2}\nabla^2 | \psi_i \rangle, \tag{62}$$

gde su ψ_i i n_i prirodne spinske orbitale i njeni okupacioni brojevi. Saglasno sa Paulijevim principom važi $0 \leq n_i \leq 1$. Takođe važi i sledeći izraz:

$$\rho(r) = \sum_i^N n_i \sum_s |\psi_i(\overrightarrow{r}, s)|^2. \tag{63}$$

Kon i Šam su predložili korišćenje jednostavnijih formula:

$$T_s[\rho] = \sum_i^N \langle \psi_i | -\frac{1}{2}\nabla^2 | \psi_i \rangle \tag{64}$$

i

$$\rho(r) = \sum_i^N \sum_s |\psi_i(\overrightarrow{r}, s)|^2. \tag{65}$$

Poslednje dve jednačine predstavljaju specijalan slučaj jednačina (62) i (63) za $n_i = 1$ za prvih N orbitala i $n_i = 0$ za ostale orbitale. Ovakava reprezentacija kinetičke energije i

gustine je tačna za slučaj N međusobno neinteragujućih elektrona. Kon i Šam su uveli neintegarugujući referentni sistem sa hamiltonijanom:

$$H_s = \sum_i^N (-\frac{1}{2}\nabla_i^2) + \sum_i^N v_s(r), \tag{66}$$

u kojem ne postoji elektron - elektron repulzivni član i čija je talasna funkcija osnovnog stanja:

$$\Psi_s = \frac{1}{\sqrt{N!}} \det[\psi_1 ... \psi_N], \tag{67}$$

gde su ψ_i N najnižih svojstvenih vrednosti jednoelektronskog hamiltonijana h_s:

$$h_s\psi_i = [-\frac{1}{2}\nabla^2 + v_s(r)]\psi_i = \varepsilon_i\psi_i. \tag{68}$$

Izraz za kinetičku energiju (64) nije jednak izrazu (50), već predstavlja njegovu aproksimaciju. Napišimo komponentu energije $F[\rho]$ kao:

$$F[\rho] = T_s[\rho] + J[\rho] + E_{xc}[\rho], \tag{69}$$

gde je $E_{xc}[\rho]$ izmensko-korelaciona energija data sa:

$$E_{xc}[\rho] = T[\rho] - T_s[\rho] + V_{ee}[\rho] - J[\rho]. \tag{70}$$

Ojler- Lagranžova jednačina sada glasi:

$$\mu = v_{eff}(\overrightarrow{r}) + \frac{\delta T_s}{\delta\rho(\overrightarrow{r})}, \tag{71}$$

gde je $v_{eff}(r)$ efektivni Kon - Šam potencijal definisan sa:

$$v_{eff}(\overrightarrow{r}) = v(\overrightarrow{r}) + \frac{\delta J[\rho]}{\delta\rho(\overrightarrow{r})} + \frac{\delta E_{xc}[\rho]}{\delta\rho(\overrightarrow{r})} = v(\overrightarrow{r}) + \int \frac{\rho(\overrightarrow{r}')}{|\overrightarrow{r} - \overrightarrow{r}'|} d\overrightarrow{r}' + v_{xc}(\overrightarrow{r}). \tag{72}$$

Jednačina (72) zajedno sa jednačinama (65) i (68) čini set Kon-Šamovih jednačina. Kao što se vidi iz njih, $v_{eff}(\overrightarrow{r})$ zavisi od $\rho(\overrightarrow{r})$ i obrnuto, što znači da je postupak rešavanja samo-saglasan. Polazi se od probne gustine, na osnovu koje se konstruiše efektivni potencijal. Potom se računaju svojstvene talasne funkcije i energije, na osnovu kojih se računa nova gustina. Postupak se ponavlja dok se ne zadovolji željena tačnost.

4.4 Aproksimacija lokalne gustine

Kon - Šamove jednačine ostavljaju član izmensko-korelacione energije eksplicitno nedefinisanim. Jedan od načina da se ovaj problem prevaziđe je da se primeni aproksimacija

lokalne gustine. U ovoj apriksomaciji koristi se raspodela uniformnog elektronskog gasa. Izraz za izmensko-korelacionu energiju postaje:

$$E_{XC}^{LDA}[\rho] = \int \rho(\overrightarrow{r})\varepsilon_{xc}(\rho)d\overrightarrow{r}, \tag{73}$$

gde je $\varepsilon_{xc}(\rho)$ izmensko-korelaciona energija jedne čestice u uniformnom elektronskom gasu na gustini ρ. Odgovarajući potencijal postaje:

$$v_{xc}^{LDA}(\overrightarrow{r}) = \frac{\delta E_{xc}^{LDA}[\rho]}{\delta\rho(\overrightarrow{r})} = \varepsilon_{xc}(\rho(\overrightarrow{r})) + \rho(\overrightarrow{r})\frac{\delta\varepsilon_{xc}(\rho)}{\delta\rho}. \tag{74}$$

Kon-Šamova orbitalna jednačina sada glasi:

$$[-\frac{1}{2}\nabla^2 + v(\overrightarrow{r}) + \int \frac{\rho(\overrightarrow{r}')}{|\overrightarrow{r} - \overrightarrow{r}'|}d\overrightarrow{r}' + v_{xc}^{LDA}(\overrightarrow{r})]\psi_i = \varepsilon_i\psi_i. \tag{75}$$

Izmensko-korelaciona energija $\epsilon_{xc}(\rho)$ može se razdvojiti na dva dela: izmenski i korelacioni. Izmenski deo se može izračunati pomoću formule:

$$\varepsilon_x(\rho) = -C_x\rho(\overrightarrow{r})^{1/3}, \quad C_x = \frac{3}{4}(\frac{3}{\pi})^{1/3}, \tag{76}$$

dok je vrednost korelacionog člana poznata zahvaljujući Monte Karlo proračunima Keperlija i Aldera.

4.5 Metod krpljenja naelektrisanja

Najvažniji korak u istraživanju organskih poluprovodnika jeste ispitivanje njihove elektronske strukture. Za sisteme koji sadrže veliki broj atoma i molekula, direktni proračun iz teorije funkcionala gustine često nije izvodljiv. Metod krpljenja naelektrisanja predstavlja modifikaciju proračuna na bazi teorije funkcionala gustine, tako što se umesto samo-saglasnim postupkom gustina nosilaca konstruiše na način koji će biti prikazan u nastavku. Ovaj metod je baziran na činjenici da raspodela naelektrisanja u sistemu sa kovalentnim vezama oko zadatog atoma zavisi od lokalnog okruženja tog atoma. Metod uvodi pojam motiva koji se pridružuje određenom atomu i predstavlja opis njegovog okruženja [17, 18]. Gustina naelektrisanja jednog motiva se odredjuje iz DFT proračuna malog sistema, na primer jednog molekula. Gustina naelektrisanja celog sistema se računa sumiranjem doprinosa svih motiva u sistemu. U organskim poluprovodnicima isti atomi mogu imati različita okruženja, samim tim i različite motive. Motivi se klasifikuju po imenima. Ime motiva sadrži tip atoma, a na to se dodaju tipovi susednih atoma, sve dok postoje motivi sa različitim okruženjem. Na primer, u molekulu naftalena postoji pet motiva: C2-C2C2H, C2-C3C2H, C3-C3C2C2, H-C2-C2C2, H-C2-C3C2 (CX predstavlja atom ugljenika koji je vezan za X ugljenikovih atoma). Doprinos motiva ukupnoj gustini naelektrisanja se računa pomoću formule:

$$m_A(\overrightarrow{r} - \overrightarrow{R_A}) = \frac{w_A(\overrightarrow{r} - \overrightarrow{R_A})}{\sum_B w_B(\overrightarrow{r} - \overrightarrow{R_B})}\rho(\overrightarrow{r}), \tag{77}$$

gde je $\rho(\overrightarrow{r})$ gustina naelektrisanja izvedena iz DFT proračuna malog sistema, R_A pozicija atoma A, a w_A takozvana težinska funkcija atoma, koja ima oblik eksponencijalno opadajuće funkcije. Konačno, ukupna gustina naelektrisanja sistema iznosi:

$$\rho(\overrightarrow{r}) = \sum_A m_A(\overrightarrow{r} - \overrightarrow{R_A}). \tag{78}$$

Sada je moguće, rešavanjem Poasonove jednačine, dobiti elektrostatički Hartrijev potencijal (treći član u jednačini (75)). Kada su poznati svi članovi u jednačini (75), moguće je rešiti i dobiti svojstvene vrednosti ε i svojstvene vektore ψ. Umesto klasičnog rešavanja Šredingerove jednačine koji daje sve energije i talasne funkcije, koristi se folded spectrum metod (FSM) [19] koji daje energije i talasane funkcije u okolini zadate energije. Ovaj metod se bazira na činjenici da ako su (ε, ψ) rešenja problema $\hat{H}\psi = \varepsilon\psi$, onda je i rešenje problema $(\hat{H} - \varepsilon_{ref})^2\psi = (\varepsilon - \varepsilon_{ref})^2\psi$, gde je ε_{ref} referentna energija. Pri tome važi da je N-to stanje (N je broj okupiranih stanja) računato od dna energijskog spektra za H problem, najniže stanje za $(H - \varepsilon_{ref})^2$ problem, ukoliko se referentna energija nalazi u energijskom procepu i blizu vrha valentne zone (ili dna provodne zone). Rezultati dobijeni metodom krpljenja naelektrisanja se u velikoj meri slažu sa onima dobijenim DFT proračunom.

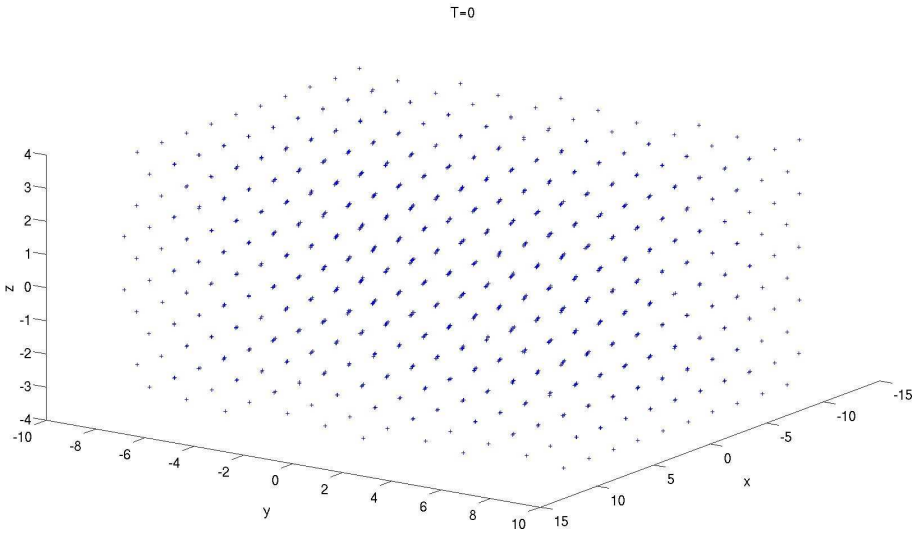
5 Atomska struktura granice između kristalnih domena u naftalenu

U ovom odeljku biće prikazani rezultati simulacije atomske strukture naftalena, najpre balk monokristala, a potom i granice između dva monokristala.

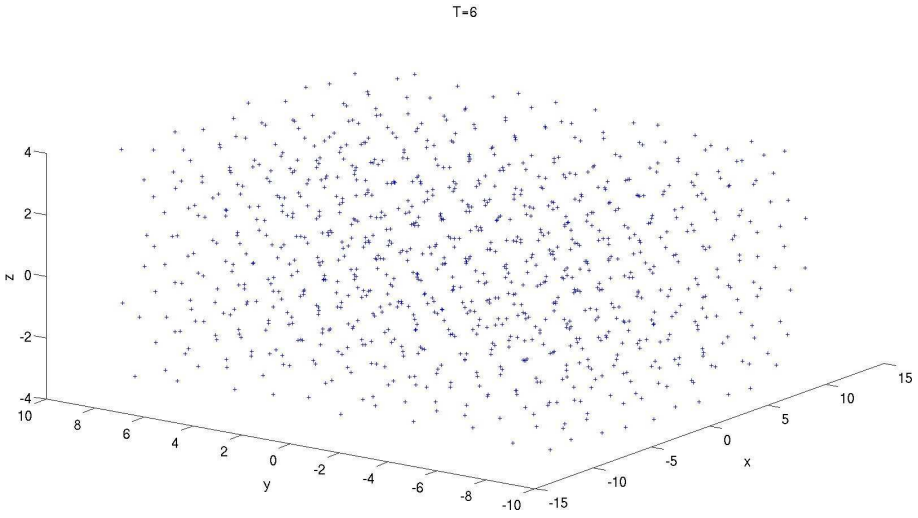
5.1 Simulacija monokristala naftalena

Implementirani Monte Karlo kod testiran je na simulacijama monokristala naftalena. Simulacije su izvedene pri konstantnom broju molekula (1000), konstantnoj zapremini na različitim temperaturama: temperaturi apsolutne nule, sobnoj temperaturi ($300K$, odnosno 6 u Lenard - Džons jedinicama) i temperaturi topljenja ($350K$, odnosno 7 u Lenard - Džons jedinicama). Početna konfiguracija u simulacijama bila je idealna kristalna rešetka naftalena. Na slikama 10, 11 i 12 prikazani su rezulati simulacija kroz prikaz strukture monokristala naftalena projektovanu na ac ravan. Centri mase molekula označeni su krstićima.

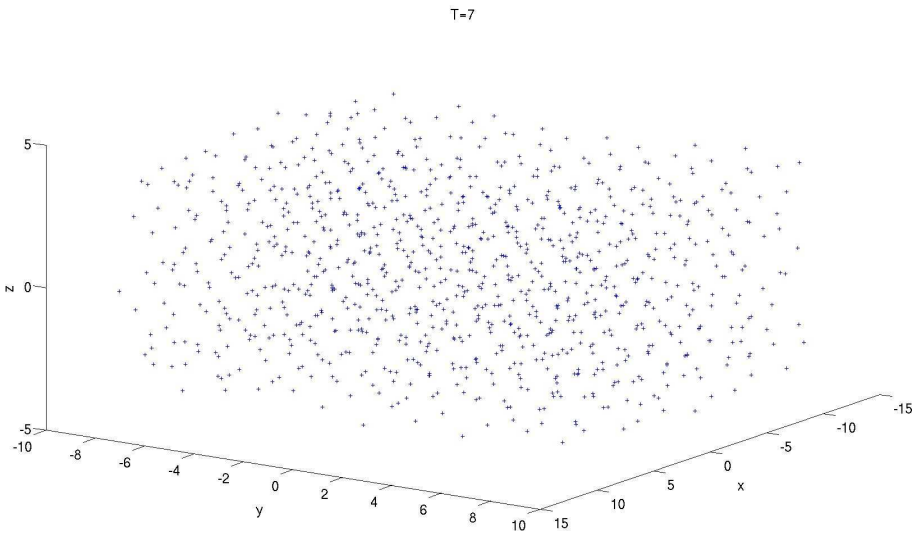
U tabeli 5 prikazani su rezultati proračuna energije, pritiska i parametra uređenja sistema za tri referentne temperature. Možemo zaključiti da na temperaturi apsolutne nule naftalen ima savršenu kristalnu strukturu. Na sobnoj temperaturi kristalna struktura je očuvana, ali narušena u velikoj meri. Na temperarturi topljenja kristalna struktura je potpuno uništena. Međutim, parametar uređenja je visok i na sobnoj temperaturi i na temperaturi topljenja. Dobijeni rezultati su u skladu sa teorijskim očekivanjima, pa



Slika 10: Atomska struktura monokristala naftalena na temperaturi apsolutne nule



Slika 11: Atomska struktura monokristala naftalena na sobnoj temperaturi



Slika 12: Atomska struktura monokristala naftalena na temperaturi topljenja

se implementirani kod može koristiti za atomsku strukturu granice dva monokristala u polikristalu naftalena.

Temperatura	Energija	Pritisak	Parametar uređenja
0	-301.774	0.022	0.98
6	-193.393	0.713	0.77
7	-170.158	0.835	0.73

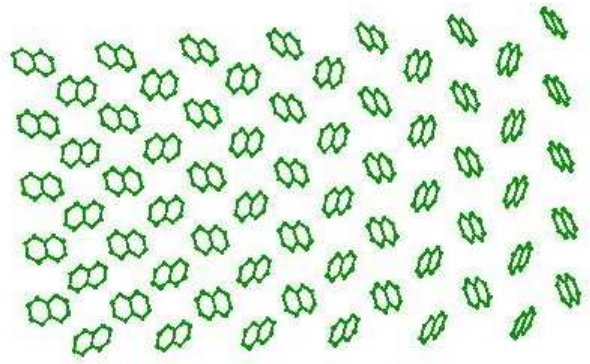
Tabela 5: Parametri monokrsitala naftalena za tri temperature. Sve veličine su date u bezdimenzionim Lenard - Džons jedicama

5.2 Simulacija granice dva monokristala naftalena

U ovom poglavlju biće prikazani rezultati simulacije granice dva monokristala naftalena. Dva monokristala su međusobno zarotirani za određeni ugao. Kontaktna površina je normalna na *ab* ravan oba kristala. Na slikama 13 i 14 prikazana je samo jedna *ab* ravan radi preglednosti.

Početna struktura sadrži 1000 molekula, odnosno 18000 atoma. Generiše se prostim spajanjem dva monokristala. Potom se pristupa relaksaciji strukture pomoću Monte Karlo simulacija. Najpre se sistem relaksira na sobnoj temperaturi. Potom se temperaturi lagano spušta sve do temperature apsolutne nule. Na ovaj način se poništavaju efekti dinamičkog neuređenja i ističu se efekti postojanja kontaktne površine.

Na slici 13 data je atomska struktura granice dva ista monokristala, dok su na slici 14 prikazane atomske strukture granica dva monokristala zarotirana za ugao φ. Sa slike 14



Slika 13: Atomska struktura granice između dva ista monokristala

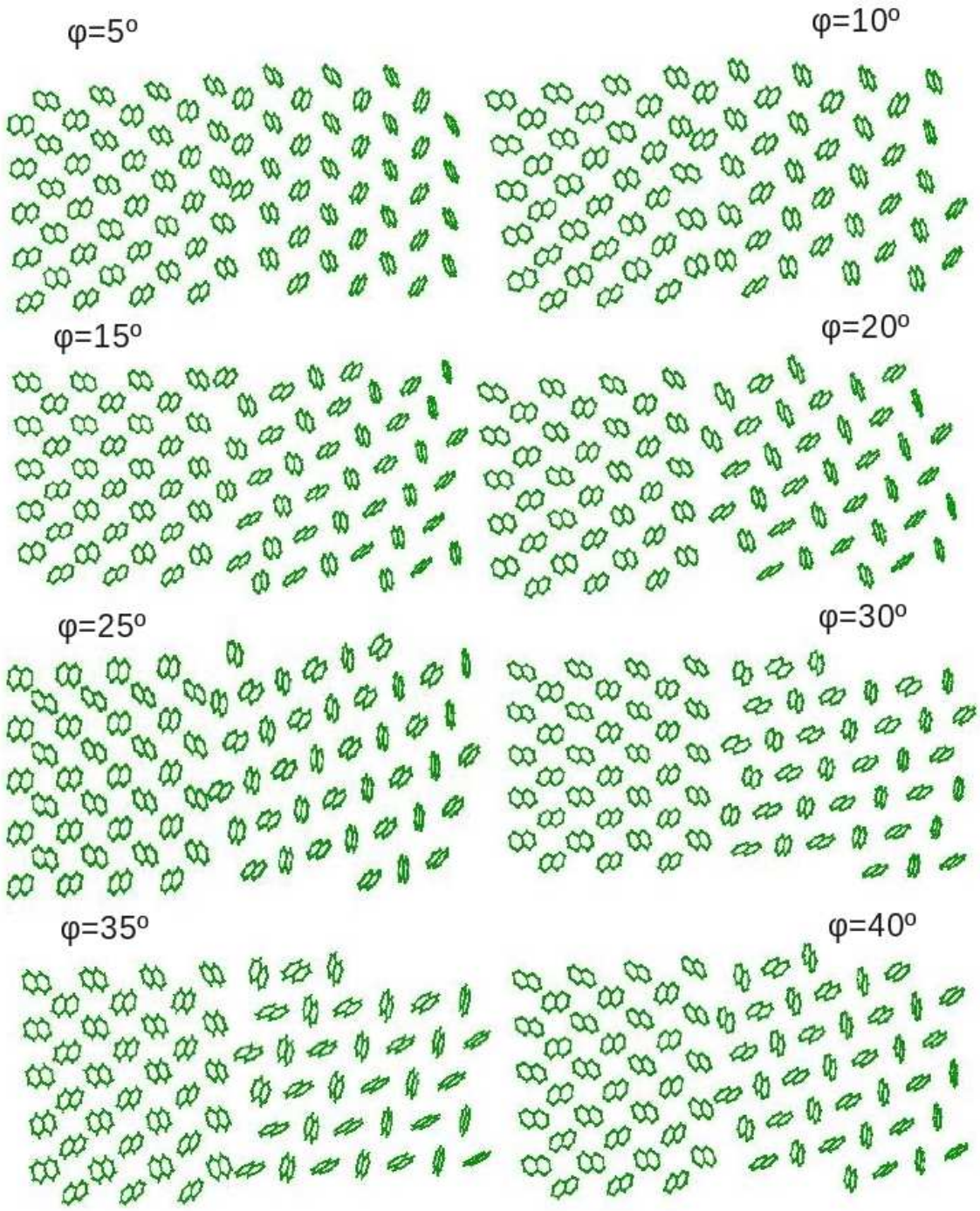
se vidi da uticaj granice između dva monokristala osećaju samo oni molekuli u neposrednoj okolini granice. Molekuli koji se nalaze u samoj okolini granice pripadaju kristalnoj rešetki jednog ili drugog kristala. Ipak, primećuje se da su molekuli najbliži granici blago zarotirani u odnosu na svoj položaj u monokristalu. Kontaktna površina ima najveći uticaj na energiju sistema, koja monotono raste sa porastom ugla između dva monokristala, što je prikazano na slici 15.

6 Elektronska struktura granice između kristalnih domena u naftalenu

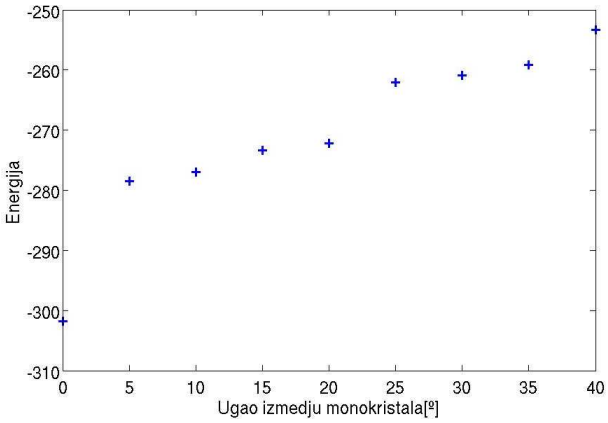
Pre prikaza rezultata proračuna elektronske strukture granice između kristalnih domena u naftalenu, biće opisan algoritam kojim se vrši proračun. Alogitam je šematski prikazan na slici 16. Ulazni podaci su atomska struktura, odnosno pozicije svih atoma u okolini granice dva monokristala i gustina naelektrisanja motiva koja se dobija iz DFT proračuna jednog molekula naftalena, kao što je to opisano u poglavlju 4.5. Na osnovu tih ulaznih podataka konstruiše se gustina naelektrisanja (elektrona). Kada je gustina naelektrisanja poznata moguće je sračunati potencijal, koji se ubacije u Šredingerovu jednačinu. Rezultat rešavanja Šredingerove jednačine su talasne funkcije i odgovarajući energijski nivoi. Kako nije moguće izračunati sva stanja u razumnom vremenskom roku, potrebno je zadati referentnu energiju oko koje traži zadati broj stanja. U ovom slučaju najzanimljivija su stanja u energijskom procepu bliže valentnoj zoni. Da bi se odredio položaj referentne energije, potrebno je okvirno znati kako izgleda zonska struktura. To je moguće videti iz gustine stanja.

6.1 Elektronska struktura monokristala naftalena

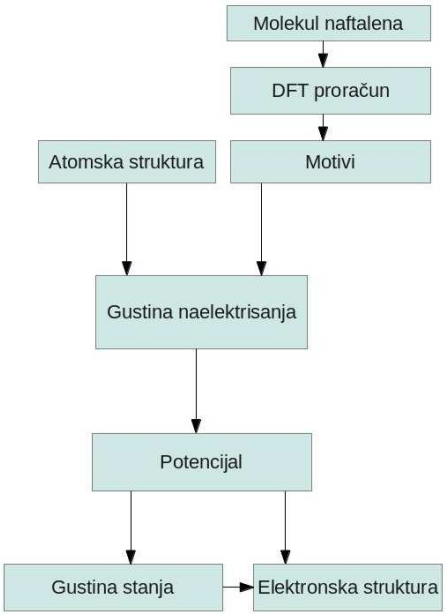
Gustina stanja monokristala je prikazana na slici 17. Sa slike se vidi da nema stanja u energijskom procepu. Za referentnu energiju uzeto je $0.5eV$. Energije najviših popunjenih



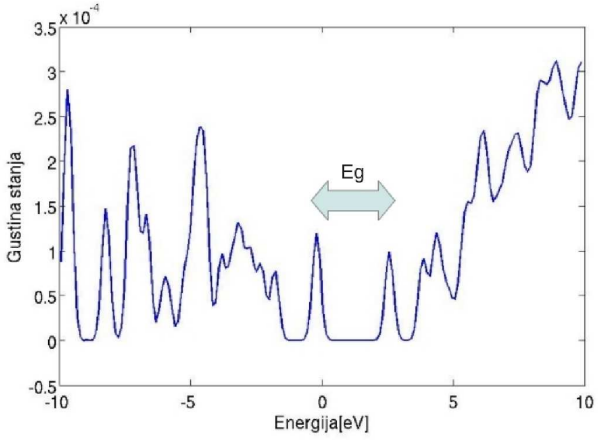
Slika 14: Atomske strukture granice između dva monokristala zarotirana međusobno za ugao φ



Slika 15: Zavisnost energije od ugla između monokristala. Energija je bezdimenziona, skalirana ε parametrom za CH grupu.



Slika 16: Algoritam proračuna elektronske strukture

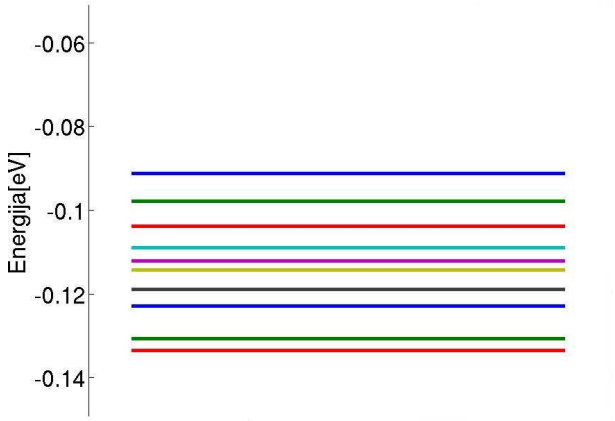


Slika 17: Gustina stanja u monokristalu naftalena

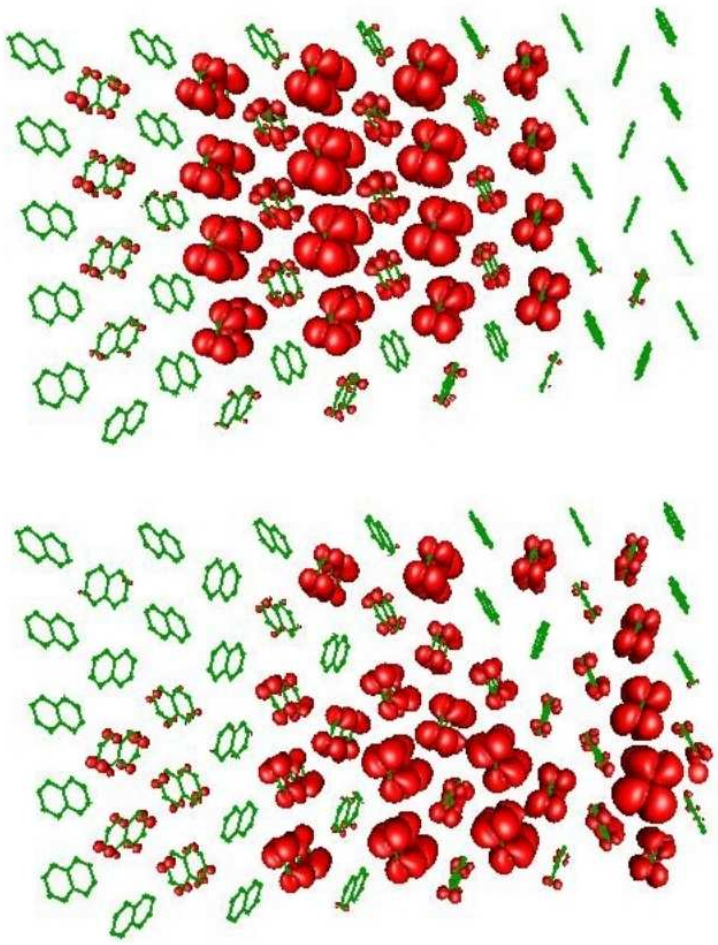
stanja u valentnoj zoni su prikazane na slici 18, dok su talasnje funkcije prvog i desetog stanja prikazane na slici 19. Oba stanja pripadaju valentoj zoni i delokalizovana su.

6.2 Elektronska struktura granice dva monokristala naftalena

U ovom odeljku će biti prikazani rezultati proračuna elektronske strukture granice dva monokristala koji su međusobno zarotirani za ugao od 5 stepeni. Gustina stanja polikristala je prikazana na slici 20. Sa slike se vidi da energijski procep u okolini valentne zone nije potpuno ravan, što navodi na zaključak da postoje stanja u energijskom procepu. Za referentnu energiju uzeto je $2eV$. Na slici 21 prikazane su energije stanja najbližih referentnoj energiji. Izvdajaju se 3 stanja koja su u energijskom procepu, dok preostala dva

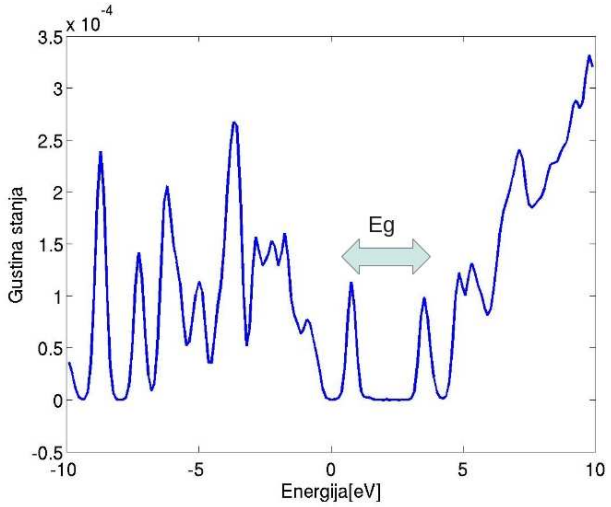


Slika 18: Energije najviših stanja u valentnoj zoni

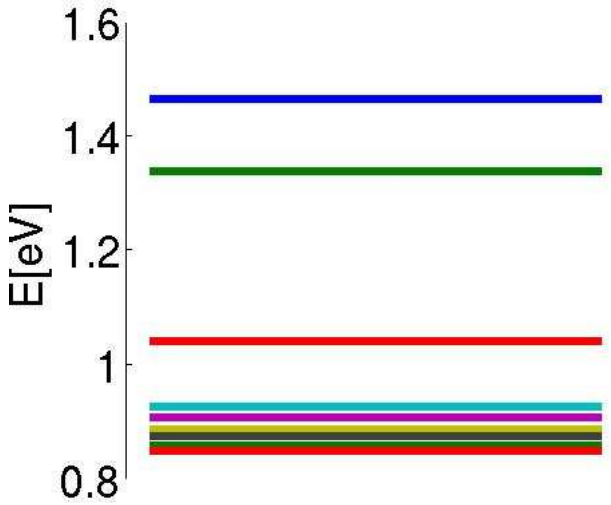


Slika 19: Talasna funkcije prvog (gore) i desetog (dole) popunjenog stanja u valentoj zoni. Verovatnoća nalaženja šupljine unutar površine obeležene crvenom bojom iznosi 90%.

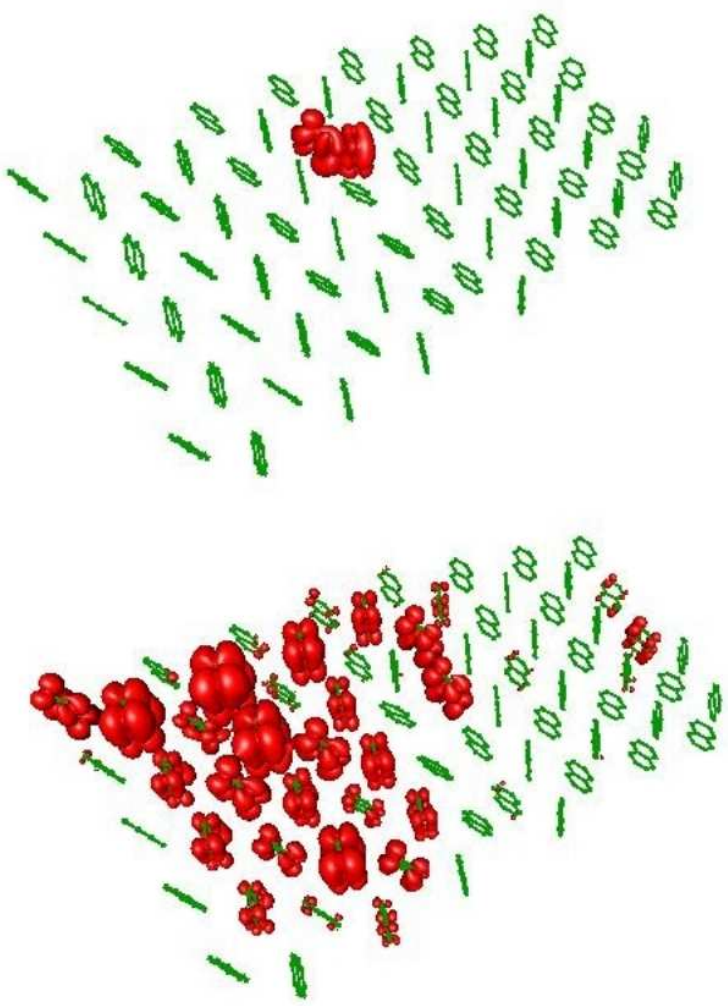
pripadaju valentnoj zoni. Talasne funkcije prvog i desetog najvišeg popunjenog stanja su prikazane na slici 22. Sa ove slike vidimo da je prvo stanje lokalizovano na molekulima na samoj granici monokristala, i to tamo gde je rastojanje između tih molekula manje od rastojanja susednih molekula u kristalu naftalena. Dakle, kontaktna površine dva monokristala dovodi do pojave lokalizovanih stanja u energijskom procepu.



Slika 20: Gustina stanja u okolini granice dva monokristala



Slika 21: Energije stanja u vrhu valentne zone



Slika 22: Talasna funkcije prvog (gore) i desetog (dole) popunjenog stanja u valentoj zoni. Verovatnoća nalaženja šupljine unutar površine obeležene crvenom bojom iznosi 90%.

7 Zaključak

U okviru ovog odeljka biće dat kratak pregled onoga što je urađeno prilikom izrade ovog rada, kao i neki zaključci koji se mogu izvući iz prezentovanih rezultata. Za dobijanje atomske strukture naftalena implementiran je Monte Karlo kod. Najpre su urađene simulacije monokristala naftalena. Ispitana je struktura naftalena na tri različite temperature. Dobijeni rezultati su u skladu sa očekivanim, što je kvalifikovalo kod za ispitivanje atomske strukture naftalena u okolini granice dva monokristala različite orijentacije kristalne rešetke. Rezultati za pokazali da kontaktna površina utiče na atomsku strukturu dva monokristala samo u okolini površine. Energija strukture koja se sastoji od dva monokristala značajno raste sa povećanjem ugla između njih. Ipak, kontaktna površina ima najveći uticaj na elektronsku strukturu. Elektronska struktura je računata pomoću metoda krpljenja naelektrisanja, koji daje gustinu nealektrisanja. Ta gustina se potom ubacuje u Kon - Šamove jednačine, čijim se rešavanjem dobijaju talasne funkcije i energije stanja sa vrha valentne zone i u energijskom procepu. Dato je poređenje elektronske strukture monokristala naftalena i kontaktne površine dva monokristala međusobno zarotiranih za ugao od 5 stepeni. U slučaju monokristala, ne postoje stanja u energijskom procepu, dok su stanja u valentnoj zoni delokalizovana, što je u skladu sa teorijom. U slučaju granice dva monokristala, postoje stanja u energijskom procepu i lokalizovana su na molekulima na samoj granici koji su blizu jedni drugima. Ova stanja predstavljaju zamke za nosioce i negativno utiču na transport nosilaca u naftalenu.

Pokretljivost nosilaca u materijalu zavisi od koncentracije zamki, koja sa druge strane zavisi od odnosa površina/zapremina kristalnih domena i broja zamki po jedinici kontaktne površine. U napravama koje rade pri maloj koncentraciji nosilaca, kao što su LED i solarne ćelije, može se dogoditi da nosioci leže u zamkama u slučaju visoke koncentracije zamki. U ovom slučaju, postojanje zamki će značajno smanjiti pokretljiovst nosilaca i ukupne performanse naprave. Takođe, zamke će dovesti do širenja emisionog ili apsorcionog spektra materijala, što je u nekim slučajevima nepoželjno.

Literatura

[1] D. L. Cheung, A. Triosi, Phys. Chem. Chem. Phys, 2008, **10**, 5941-5952

[2] M. Berggren, D. Nilsson, N. D. Robinson, Nature Mater, 2007, **6**, 3-5

[3] S. R. Forrest, Nature, 2004. **428**, 911-918

[4] R. H. Friend, R. W. Gymer, A. B. Holmes, J. H. Burroughes, R. N. Marks, C. Taliani, D. D. C. Bradley, D. A. D. Santos, J. L. Bredas, M. Logdlund, W. R. Salaneck, 1999, **397**, 121

[5] J. H. Burroughes, D. D. C. Bradley, A. R. Brown, R. N. Marks, K. Mackay, R. H. Friend, P. L. Burns, A. B. Holmes, Nature, 1990, **347**, 539-541

[6] V. L. Colvin, M. C. Schlamp, A. P. Alivisatos, Nature, 1994, **370**, 354

[7] W. L. Kalb, S. Haas, C. Krellner, T. Mathis, B. Batlogg, Phys. Rew. B, 2010, **81**, 155315

[8] G. Li, V. Shrotriya, J. S. Huang, Y. Yao, T. Moriarty, K. Emery, Y. Yang, Nature Mater, 2005, **4**, 864

[9] A .Dodabalapur, L. Torsi, H. E. Katz, Science, 1995, **268**, 270

[10] M. Grundmann, *The Physics of Semiconductors: An Introduction Including Nanophysics and Applications*, Springer, 2010.

[11] M.P. Allen, D. J. Tildesey, *Computer Simulations of Liquids*, Oxford University Press, 1999.

[12] D. Frenkel, B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*, Academic Press, 2002.

[13] C. D. Wick, M. G. Martin, J. I. Siepmann, J. Phys. Chem. B, 2000, **104**, 8008-8016

[14] M. G. Martin, J. I. Siepmann J. Phys. Chem. B, 1998, **102**, 2569-2577

[15] H. Steuer, *Thermodynamical Properties of a Model Liquid Crystal*, TU Berlin, 2004.

[16] R. G. Parr, W. Yang, *Density - Functional Theory of Atoms and Molecules, Oxford University Press, 1989*

[17] N. Vukmirović, L. W. Wang, J. Chem. Phys, 2008, **128**, 121102

[18] N. Vukmirović, L. W. Wang, J. Phys. Chem. B, 2009, **113**, 409-415

[19] A. Canning, L. W. Wang, A. Williamson, A. Zunger, J. Chem. Phys. 2000, **160**, 29-41

8 Dodatak A: Program za Monte Karlo simulacije

U nastavku je dat kod programa Monte Karlo simulacije naftalena pisan u programskoj jeziku C.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define NR_END 0
#define FREE_ARG char*
#define MAXNPC  50
#define MAXNPN 200
#define FREQ 0.05
#define R0 0.378378
#define R1 0.655369
#define pi 3.1415926535

#define sina1 0.912044581
#define cosa1 0.410091065
#define sinc 0
#define cosc 1
#define sind 0
#define cosd 1

#define NRANSI
#include "nrutil.h"
#define ROTATE(a,i,j,k,l) g=a[i][j];h=a[k][l];a[i][j]=g-s*(h+g*tau);\
  a[k][l]=h+s*(g-h*tau);


int N, NN, Mcx, Mcy, Mcz, Nc, Mnx, Mny, Mnz, Nn, N0, N1, Nstep, Nrefr;
double *rx, *ry, *rz, *fi, *teta, *omega;
double **rmx, **rmy, **rmz;
double *rxnew, *rynew, *rznew, *pom1, *pom2, *pom3, *pom4, *pom5, *pom6;
FILE *matlab;

int *ivector(long, long);
double *dvector(long, long);
int **imatrix(long, long, long, long);
double **dmatrix(long, long, long, long);

void free_ivector(int *, long, long);
void free_dvector(double *, long, long);
void free_imatrix(int **, long, long, long, long);
void nrerror(char *);
void free_dmatrix(double **, long, long, long, long);


void mapping(int **);
int icell(int, int, int);
int ineig(int, int, int);
int box_rcut(double, double);
void make_list(double, double, double, double, double, double, int **, int *, int *, ↩
    int **, int *, int **, int *);
void potential(double, double, double, double, double, double *, double *);
void total_energy(double, double, double, double, double, int *, int **, double *, ↩
    double *);
void energy(int, double, double, double, double, double, double, double, double, double↩
    , double, double, int *, int **, double *, double *, double *);
double ran2(long *);
void jacobi(double**, int , double ,double **, int *);
double nematic();
```

```
void molecule(int, double **, double **, double **);

int main()
{
  int i, step, acm, sp, sb, **map, *icc, *n0, **list0, *n1, **list1, freq, NPT, m, ↩
      first, k;
  long seed;
  double boxx, boxy, boxz ,boxinvx, boxinvy, boxinvz, boxratx, boxraty, boxratz, ↩
      boxnewx, boxnewy, boxnewz, cellx, celly, cellz, vol, volnew, rmin, rcut, msize, A↩
      , B, C,
  dens, temp, beta, pressure, pres, v, vn, vold, vnew, w, wold, wnew,
        acv, acvsq, avv, acp, acpsq, avp, acd, acdsq, avd, acvol, acvolsq, avvol, flv, ↩
            flp, fld, flvol,rxi,ryi,rzi,
        rxiold, ryiold, rziold, rxinew, ryinew, rzinew, dboxmaxx,dboxmaxy, dboxmaxz, ↩
            drmax, dfmax, deltv, deltvb, deltw, delthb, finew, tetanew, omeganew, fiold↩
            , tetaold, omegaold, pom, order,vmaxold, vmaxnew, deltvmax, deltvmaxb;
  FILE *f_input, *f_output, *input, *output, *f_output1, *f_output2;
        char input_name[50], output_name[50];

  f_input = fopen("input.txt","r");
  fscanf(f_input,"\n");
        fscanf(f_input, "input %s\n",input_name);
  fscanf(f_input, "output  %s\n", output_name);
  fscanf(f_input, "N          %d \n", &N);
  fscanf(f_input, "temp       %lf\n", &temp);
  fscanf(f_input, "dens       %lf\n", &dens);
  fscanf(f_input, "pres       %lf\n", &pressure);
  fscanf(f_input, "rmin       %lf\n", &rmin);
  fscanf(f_input, "rcut       %lf\n", &rcut);
  fscanf(f_input, "Nstep      %d \n", &Nstep);
  fscanf(f_input, "Nrefr      %d \n", &Nrefr);
  fscanf(f_input, "NPT        %d \n", &NPT);
  fscanf(f_input, "msize      %lf\n", &msize);
  fclose(f_input);

        printf("input name %s\n", input_name);
  printf("output name %s\n", output_name);
  printf("N          %d\n", N);
  printf("temp       %f\n", temp);
  printf("dens       %f\n", dens);
  printf("pres       %f\n", pressure);
  printf("rmin       %f\n", rmin);
  printf("rcut       %f\n", rcut);
  printf("Nstep      %d\n", Nstep);
  printf("Nrefr      %d\n", Nrefr);
  printf("NPT        %d\n", NPT);
        printf("msize      %lf\n",msize);
  printf("\n");

  /* random generator */
  seed = -124656452;

  int nx=5;
  int ny=10;
  int nz=5;

        N = nz * nx * ny;
  NN = 4 * N ;

  /* allocate memory for the positions and angles of the molecules */
  rx = dvector(0, NN-1);
  ry = dvector(0, NN-1);
  rz = dvector(0, NN-1);
  fi = dvector(0, NN-1);
```

```
    teta = dvector(0, NN-1);
  omega = dvector(0, NN-1);
    pom1 = dvector(0, NN-1);
    pom2 = dvector(0, NN-1);
    pom3 = dvector(0, NN-1);
    pom4 = dvector(0, NN-1);
    pom5 = dvector(0, NN-1);
    pom6 = dvector(0, NN-1);
rmx=dmatrix(0, NN-1, 0 ,9);
      rmy=dmatrix(0, NN-1, 0 ,9);
      rmz=dmatrix(0, NN-1, 0 ,9);
if (NPT)
{
  rxnew = dvector(0,  NN-1);
  rynew = dvector(0,  NN-1);
  rznew = dvector(0,  NN-1);
}

      /* allocate memory for indices of the cells */
icc = ivector(0, NN-1);

/* calculate initial box length */
A = 2.188648498;
B = 1.608919284;
C = 2.115494143;
boxx = 2 * A * nx;
boxy = C * ny * sina1;
boxz = B * nz;
      boxinvx = 1 / boxx;
boxinvy = 1 / boxy;
boxinvz = 1 / boxz;


/* calculate volume */
vol = boxx * boxy * boxz;

/* reduced maximum displacement for the box length*/
dboxmaxx = boxx / 100;
dboxmaxy = boxy / 100;
dboxmaxz = boxz / 100;

/* reduced maximum displacement for the molecule position and angles*/
drmax = 0.005;
      dfmax = 0.005;

/* convert input data to progr m units */
beta = 1. / temp;

/*defining starting configuration*/
for(i=0; i<N; i++)
    {
  rz[i]=((i /nx) % nz) * B  + B/4 - boxz/2;
  rx[i]=(i % nx) * A  + (i /(nz * nx)) * C * cosa1 - boxx/2;
  ry[i]=(i /(nz * nx)) * C * sina1 - boxy/2 + C * sina1/2;
  fi[i]= -1.0985;
  teta[i]= -0.7431;
  omega[i]= 0.9432;
              pom1[i]=cos(fi[i]);
              pom2[i]=sin(fi[i]);
              pom3[i]=cos(teta[i]);
        pom4[i]=sin(teta[i]);
        pom5[i]=cos(omega[i]);
        pom6[i]=sin(omega[i]);
        if (rx[i] > 0) {rx[i] = rx[i] - boxx/2;}
```

```
  molecule(i, rmx, rmy, rmz);
              }

      for(i=N; i<(2 * N); i++)
    {
  rz[i]=(((i - N)/nx) % nz) * B  + 3 * B/4 - boxz/2;
  rx[i]=((i -  N) % nx) * A + ((i - N)/(nz * nx)) * C * cosa1 + A/2  - boxx/2;
  ry[i]=((i - N)/(nz * nx)) * C * sina1 - boxy/2  + C * sina1/2;
        fi[i]=2.1051;
  teta[i]=0.6945;
  omega[i]=-0.9720;
              pom1[i]=cos(fi[i]);
              pom2[i]=sin(fi[i]);
              pom3[i]=cos(teta[i]);
        pom4[i]=sin(teta[i]);
        pom5[i]=cos(omega[i]);
        pom6[i]=sin(omega[i]);
  if (rx[i] > 0) {rx[i] = rx[i] - boxx/2;}

  molecule(i, rmx, rmy, rmz);
              }


for(i=2 * N; i< (3 * N); i++)
    {
  rz[i]=(((i - 2 * N)/nx) % nz) * B  * cosc + B/4 + ((i - 2 * N) % nx) * A * sinc - ↩
      boxz/2;
  rx[i]=((i - 2 * N) % nx) * A * cosc + ((i - 2 * N)/(nz * nx)) * C * cosa1 - (((i - ↩
      2 * N)/nx) % nz) * B  * sinc;
  ry[i]=((i - 2 * N)/(nz * nx)) * C * sina1 - boxy/2 + C * sina1/2;
  fi[i]= -1.0985;
  teta[i]= -0.7431;
  omega[i]= 0.9432;
  pom1[i]=cos(fi[i]);
        pom2[i]=sin(fi[i]);
        pom3[i]=cos(teta[i]);
        pom4[i]=sin(teta[i]);
        pom5[i]=cos(omega[i]);
        pom6[i]=sin(omega[i]);
  if (rx[i] > (boxx/2 * cosc)) {rx[i] = rx[i] - boxx/2 * cosc; rz[i] = rz[i] - boxx/2↩
       * sinc;}
              if (rz[i] > boxz) {rz[i] = rz[i] - boxz * cosc; rx[i] = rx[i] + boxz * ↩
                  sinc;}
              if (rx[i] < 0) {rx[i] = rx[i] + boxx/2 * cosc; rz[i] = rz[i] + boxx/2 *↩
                   sinc;}
    if (rz[i] > boxz) {rz[i] = rz[i] - boxz * cosc; rx[i] = rx[i] + boxz * sinc;}
              if (rx[i] > (boxx/2 * cosc)) {rx[i] = rx[i] - boxx/2 * cosc; rz[i] = rz↩
                  [i] - boxx/2 * sinc;}

  molecule(i, rmx, rmy, rmz);
              }

     for(i= 3 * N; i<(4 * N); i++)
    {
  rz[i]=(((i - 3 * N)/nx) % nz) * B  * cosc + B/4 + B/2 * cosc + ((i - 3 * N) % nx) *↩
       A * sinc + A/2 *  sinc - boxz/2;
  rx[i]=((i - 3 * N) % nx) * A * cosc + ((i - 3 * N)/(nz * nx)) * C * cosa1 - (((i - ↩
      3 * N)/nx) % nz) * B  * sinc + A/2 * cosc - B/2 * sinc;
  ry[i]=((i - 3 * N)/(nz * nx)) * C * sina1 - boxy/2 +  C * sina1/2;
  fi[i]=2.1051;
  teta[i]=0.6945;
  omega[i]=-0.9720;
  pom1[i]=cos(fi[i]);
```

```
          pom2[i]=sin(fi[i]);
          pom3[i]=cos(teta[i]);
          pom4[i]=sin(teta[i]);
          pom5[i]=cos(omega[i]);
          pom6[i]=sin(omega[i]);
   if (rx[i] > (boxx/2 * cosc)) {rx[i] = rx[i] - boxx/2 * cosc; rz[i] = rz[i] - boxx/2↩
        * sinc;}
                if (rz[i] > boxz) {rz[i] = rz[i] - boxz * cosc; rx[i] = rx[i] + boxz * ↩
                    sinc;}
                if (rx[i] < 0) {rx[i] = rx[i] + boxx/2 * cosc; rz[i] = rz[i] + boxx/2 *↩
                    sinc;}
   if (rz[i] > boxz) {rz[i] = rz[i] - boxz * cosc; rx[i] = rx[i] + boxz * sinc;}
                if (rx[i] > (boxx/2 * cosc)) {rx[i] = rx[i] - boxx/2 * cosc; rz[i] = rz↩
                    [i] - boxx/2 * sinc;}

   molecule(i, rmx, rmy, rmz);
                }


      /* create initial configuration */
     if ((input=fopen(input_name, "r"))==NULL) first=1;
     else first=0;

   if (first==0) {
        input=fopen(input_name, "r");
                    fread(rx, sizeof(double),NN, input);
                    fread(ry, sizeof(double),NN, input);
                    fread(rz, sizeof(double),NN, input);
                    fread(fi, sizeof(double),NN, input);
                    fread(teta, sizeof(double),NN, input);
                    fread(omega, sizeof(double),NN, input);
                    for (i = 0; i < NN; i++)
                    {
     icc[i] = 0;
                 pom1[i]=cos(fi[i]);
           pom2[i]=sin(fi[i]);
           pom3[i]=cos(teta[i]);
           pom4[i]=sin(teta[i]);
           pom5[i]=cos(omega[i]);
           pom6[i]=sin(omega[i]);
     molecule(i, rmx, rmy, rmz);
                    }
               fclose(input);
            }

  /* divide box on cells */
  Mcx = box_rcut(boxx, rcut+msize);
  Mcy = box_rcut(boxy, rcut+msize);
  Mcz = box_rcut(boxz, rcut+msize);
  Mnx = (Mcx > 3) ? 3 : 1;
  Mny = (Mcy > 3) ? 3 : 1;
  Mnz = (Mcz > 3) ? 3 : 1;
  Nc = Mcx * Mcy * Mcz;
  Nn = Mnx * Mny * Mnz;

  /* calculate initial cell length */
  cellx = boxx / Mcx;
  celly = boxy / Mcy;
  cellz = boxz / Mcz;

  /* assumed maximum number of the atoms in the cell (N0) and the cell neighborhood (N1↩
      ) */
  N0 = (int)(MAXNPC *  NN/ Nc);
  N1 = (int)(MAXNPN *  NN/ Nc);
```

```
/* allocate memory */
map = imatrix(0, Nc - 1, 0, Nn - 1);
n0 = ivector(0, Nc - 1);
list0 = imatrix(0, Nc - 1, 0, N0 - 1);
n1 = ivector(0, NN-1);
list1 = imatrix(0, NN-1, 0, N1 - 1);

/* allocate memory */
mapping(map);
make_list(boxx, boxy, boxz, cellx, celly, cellz, map, icc, n0, list0, n1, list1, &↩
    freq);

/* calculate initial energy */
total_energy(rmin, rcut, boxx, boxy, boxz, n1, list1, &v, &w);
vn = v / NN;
pres = dens * temp + w / vol / NN;

printf("initial volume:           %22.10f\n", vol);
printf("initial potential energy: %22.10f\n", vn);
printf("initial pressure:         %22.10f\n", pres);
printf("\n");
printf("% 7s %7s %22s %22s %22s %22s\n", "step", "sp", "vn", "pres", "dens", "order")↩
    ;


/*****************************************/
/* loops over all cycles and all molecules */
/*****************************************/
sp = 0;
acm = 0;
acv = acvsq = acp = acpsq = acd = acdsq = acvol = acvolsq = 0;

/*Monte Carlo loop*/
for (step = 1; step <= Nstep; step++)
{

   if (step % Nrefr == 0)
    if (Mcx == box_rcut(boxx, rcut) && (Mcx>1) && Mcy == box_rcut(boxy, rcut) && (Mcy↩
        >1) && Mcz == box_rcut(boxz, rcut) && (Mcz>1))
    {
      /* make list */
      make_list(boxx, boxy, boxz, cellx, celly, cellz, map, icc, n0, list0, n1, list1↩
          , &freq);
      if ((double)freq / Nrefr / NN> FREQ)
        nrerror("You have to decrease Nrefr and/or boxrcut in function box_rcut!\n");
    }
    else
    {
          /* memory free */
      free_imatrix(map, 0, Nc - 1, 0, Nn - 1);
      free_ivector(n0, 0, Nc - 1);
      free_imatrix(list0, 0, Nc - 1, 0, N0 - 1);
      free_ivector(n1, 0, NN-1);
      free_imatrix(list1, 0, NN-1, 0, N1 - 1);

      Mcx = box_rcut(boxx, rcut+msize);
      Mcy = box_rcut(boxy, rcut+msize);
      Mcz = box_rcut(boxz, rcut+msize);
      Mnx = (Mcx > 3) ? 3 : 1;
      Mny = (Mcy > 3) ? 3 : 1;
      Mnz = (Mcz > 3) ? 3 : 1;
      Nc = Mcx * Mcy * Mcz;
      Nn = Mnx * Mny * Mnz;
```

```
  cellx = boxx / Mcx;
  celly = boxy / Mcy;
  cellz = boxz / Mcz;

  N0 = (int)(MAXNPC * NN / Nc);
  N1 = (int)(MAXNPN * Nn * NN/ Nc);

  map = imatrix(0, Nc - 1, 0, Nn - 1);
  n0 = ivector(0, Nc - 1);
  list0 = imatrix(0, Nc - 1, 0, N0 - 1);
  n1 = ivector(0,NN-1 );
  list1 = imatrix(0, NN-1, 0, N1 - 1);

  mapping(map);
  make_list(boxx, boxy, boxz, cellx, celly, cellz, map, icc, n0, list0, n1, list1↩
      , &freq);
}

/*randomly pick-up one molecule*/
i=round((NN-1) * ran2(&seed));
rxiold = rx[i];
ryiold = ry[i];
rziold = rz[i];
fiold=fi[i];
tetaold=teta[i];
omegaold=omega[i];

/* calculate the energy of i in the old configuration */

          energy(i, rxiold, ryiold, rziold, fiold, tetaold, omegaold, rmin, rcut,↩
               boxx, boxy, boxz, n1, list1, &vold, &wold, &vmaxold);

/* move i and pickup the central image */
rxinew = rxiold + (2 * ran2(&seed) - 1) * drmax;
ryinew = ryiold + (2 * ran2(&seed) - 1) * drmax;
rzinew = rziold + (2 * ran2(&seed) - 1) * drmax;
rxinew = rxinew - round(rxinew * boxinvx) * boxx;
ryinew = ryinew - round(ryinew * boxinvy) * boxy;
rzinew = rzinew - round(rzinew * boxinvz) * boxz;
          finew = fiold + (2 * ran2(&seed) - 1) * dfmax;
tetanew= tetaold + (2. * ran2(&seed) - 1.) * dfmax;
          omeganew= omegaold + (2. * ran2(&seed) - 1.) * dfmax;
          pom1[i]=cos(finew);
pom2[i]=sin(finew);
pom3[i]=cos(tetanew);
pom4[i]=sin(tetanew);
pom5[i]=cos(omeganew);
pom6[i]=sin(omeganew);
molecule(i, rmx, rmy, rmz);

/* calculate the energy of i in the new configuration */
       energy(i, rxinew, ryinew, rzinew, finew, tetanew, omeganew, rmin, rcut, ↩
           boxx, boxy, boxz, n1, list1, &vnew, &wnew, &vmaxnew);

deltv = vnew - vold;
deltvmax=vmaxnew - vmaxold;
deltw = wnew - wold;
deltvb = beta * deltv;
deltvmaxb = beta * deltvmax;
                  double ggggg = ran2(&seed);

/*Markov chain*/
if ((deltvb < 10) && ((deltvb <= 0) || (ggggg < exp(-deltvb))))
```

```
      {
        v += deltv;
        w += deltw;
        rx[i] = rxinew;
        ry[i] = ryinew;
        rz[i] = rzinew;
        fi[i] = finew;
        teta[i] = tetanew;
        omega[i] = omeganew;
        sb=1;
        sp++;
      }
      else {
                    pom1[i]=cos(fiold);
        pom2[i]=sin(fiold);
        pom3[i]=cos(tetaold);
        pom4[i]=sin(tetaold);
        pom5[i]=cos(omegaold);
        pom6[i]=sin(omegaold);
        molecule(i, rmx, rmy, rmz);
              sb=0;
                }

    if ((double)sp / (double)step > 0.9)
      drmax *= 1.05;
    else
      drmax *= 0.95;
                if (drmax<0.001) drmax=0.001;
                if (drmax>0.01)   drmax=0.01;

    if ((double)sp / (double)step > 0.9)
      dfmax *= 1.05;
    else
      dfmax *= 0.95;
                if (dfmax<0.001) dfmax=0.001;
                if (dfmax>0.01)   dfmax=0.01;



                if (step%Nrefr==0)
    {
      total_energy(rmin, rcut, boxx, boxy, boxz, n1, list1, &v, &w);
      vn = v / NN;
      pres = dens * temp + w / vol / NN;

      /* increment accumulators */
      if (step>(Nstep/3)) {
      acm++;
      acv += vn;
      acp += pres;
      acvsq += vn * vn;
      acpsq += pres * pres; }


    order=nematic();
    printf("%7d %7d   %22.10f %22.10f %22.10f %22.10f\n", step, sp, vn, pres, dens, ↩
        order);



    f_output1=fopen("output1.txt", "w");
    for (i = 0; i < NN; i++)
      {
      energy(i, rx[i], ry[i], rz[i], fi[i], teta[i], omega[i], rmin, rcut, boxx, boxy, ↩
```

```
          boxz, n1, list1, &v, &w, &vmaxnew);
                  fprintf(f_output1, "%f %f %f %f %f %f %f\n", rx[i], ry[i], rz[i], fi[↩
                      i], teta[i], omega[i], v);
    }
  fclose(f_output1);

}


  /* NPT simulation */
  if (NPT)
  {
    boxnewx = boxx + (2 * ran2(&seed) - 1) * dboxmaxx;
    boxnewy = boxy + (2 * ran2(&seed) - 1) * dboxmaxy;
    boxnewz = boxz + (2 * ran2(&seed) - 1) * dboxmaxz;

    volnew = boxnewx * boxnewy * boxnewz;
    dens = NN / volnew;
    boxratx= boxnewx/ boxx;
    boxraty= boxnewy/ boxy;
    boxratz= boxnewz/ boxz;

    /* calculate new coordinates */
    for (i = 0; i < NN; i++)
    {
      rx[i] = rx[i] * boxratx;
      ry[i] = ry[i] * boxraty;
      rz[i] = rz[i] * boxratz;
      molecule(i, rmx, rmy, rmz);
    }

    /* calculate new energy */
    total_energy(rmin, rcut, boxnewx, boxnewy, boxnewz, n1, list1, &vnew, &wnew);
    pres = dens * temp + wnew / volnew;
    delthb = beta * (vnew - v + pressure * (volnew - vol)) - NN * log(volnew / vol);

    /* check for acceptance */
    if ((delthb < 10) && ((delthb <= 0) || (ran2(&seed) < exp(-delthb))))
    {
      for (i = 0; i < NN; i++)
      {
        rx[i] = rxnew[i];
        ry[i] = rynew[i];
        rz[i] = rznew[i];
        molecule(i, rmx, rmy, rmz);
      }
      v = vnew;
      w = wnew;
      boxx = boxnewx;
      boxy = boxnewy;
      boxz = boxnewz;
      boxinvx = 1 / boxx;
      boxinvy = 1 / boxy;
      boxinvz = 1 / boxz;
      sb++;
    }

    else    { rx[i] = rx[i] / boxratx;
        ry[i] = ry[i] / boxraty;
        rz[i] = rz[i] / boxratz;
        molecule(i, rmx, rmy, rmz);
      }

    vol = boxx * boxy * boxz;
```

```
    dens = NN / vol;

    vn = v / NN;
    pres = dens * temp + w / vol;

    if (step>(Nstep/3)) {
      acm++;
      acv += vn;
      acp += pres;
      acd += dens;
      acvol += vol;
      acvsq += vn * vn;
      acpsq += pres * pres;
      acdsq += dens * dens;
      acvolsq += vol * vol;
            }

    if ((double)sp / step > 0.5)
      dboxmaxx *= 1.05;
    else
      dboxmaxx *= 0.95;

  }


}

                    output=fopen(output_name, "w");
                    fwrite(rx, sizeof(double),NN, output);
                    fwrite(ry, sizeof(double),NN, output);
                    fwrite(rz, sizeof(double),NN, output);
                    fwrite(fi, sizeof(double),NN, output);
                    fwrite(teta, sizeof(double),NN, output);
                    fwrite(omega, sizeof(double),NN, output);
                  fclose(output);


avv = acv / acm;
acvsq = acvsq / acm - avv * avv;
flv = (acvsq > 0) ? acvsq : 0.;

avp = acp / acm;
acpsq= acpsq / acm - avp * avp;
flp = (acpsq > 0) ? acpsq : 0.;

if (NPT)
{
  avd = acd / (acm / 2);
  acdsq = acdsq / (acm / 2) - avd * avd;
  fld = (acdsq > 0) ? acdsq : 0.;

  avvol = acvol / (acm / 2);
  acvolsq = acvolsq / (acm / 2) - avvol * avvol;
  flvol = (acvolsq > 0) ? acvolsq : 0.;

}
printf("potential: %22.10f +- %22.10f\n", avv, flv);
printf("preasure:  %22.10f +- %22.10f\n", avp, flp);
if (NPT)
{
  printf("density:   %22.10f +- %22.10f\n", avd, fld);
  printf("volume:    %22.10f +- %22.10f\n", avvol, flvol);

}
```

```
  /* memory free */
  free_dvector(rx, 0, NN-1);
  free_dvector(ry, 0, NN-1);
  free_dvector(rz, 0, NN-1);
  free_dvector(fi, 0, NN-1);
  free_dvector(teta, 0, NN-1);
  free_dvector(omega, 0, NN-1);
  free_dvector(pom1, 0, NN-1);
  free_dvector(pom2, 0, NN-1);
  free_dvector(pom3, 0, NN-1);
  free_dvector(pom4, 0, NN-1);
  free_dvector(pom5, 0, NN-1);
  free_dvector(pom6, 0, NN-1);
  if (NPT)
  {
    free_dvector(rxnew, 0, NN-1);
    free_dvector(rynew, 0, NN-1);
    free_dvector(rznew, 0, NN-1);
  }
  free_ivector(icc, 0, NN-1);
  free_imatrix(map, 0, Nc - 1, 0, Nn - 1);
  free_ivector(n0, 0, Nc - 1);
  free_imatrix(list0, 0, Nc - 1, 0, N0 - 1);
  free_ivector(n1, 0, NN-1);
  free_imatrix(list1, 0, NN-1, 0, N1 - 1);
  free_dmatrix(rmx, 0, NN-1, 0, 9);
  free_dmatrix(rmy, 0, NN-1, 0, 9);
  free_dmatrix(rmz, 0, NN-1, 0, 9);
  return 0;
}

/* allocate an int vector with subscript range v[nl..nh] */
int *ivector(long nl, long nh)
{
  int *v;
  v = (int *)malloc((size_t)((nh - nl + 1 + NR_END) * sizeof(int)));
  if (!v)
    nrerror("allocation failure in ivector()");
  return v - nl + NR_END;
}

/* allocate a double vector with subscript range v[nl..nh] */
double *dvector(long nl, long nh)
{
  double *v;
  v = (double *)malloc((size_t)((nh - nl + 1 + NR_END) * sizeof(double)));
  if (!v)
    nrerror("allocation failure in dvector()");
  return v - nl + NR_END;
}

/* allocate an int matrix with subscript range m[nrl..nrh][ncl..nch] */
int **imatrix(long nrl, long nrh, long ncl, long nch)
{
  long i, nrow = nrh - nrl + 1, ncol = nch - ncl + 1;
  int **m;
  /* allocate pointers to rows */
  m = (int **) malloc((size_t)((nrow + NR_END) * sizeof(int *)));
  if (!m)
    nrerror("allocation failure 1 in matrix()");
  m += NR_END;
  m -= nrl;
  /* allocate rows and set pointers to them */
```

```
  m[nrl] = (int *) malloc((size_t)((nrow * ncol + NR_END) * sizeof(int)));
  if (!m[nrl])
    nrerror("allocation failure 2 in matrix()");
  m[nrl] += NR_END;
  m[nrl] -= ncl;
  for (i = nrl + 1; i <= nrh; i++)
    m[i] = m[i - 1] + ncol;
  /* return pointer to array of pointers to rows */
  return m;
}
double  **dmatrix(long nrl, long nrh, long ncl, long nch)
{
        long i, nrow = nrh - nrl + 1, ncol = nch - ncl + 1;
        double **m;
        /* allocate pointers to rows */
        m = (double **) malloc((size_t)((nrow + NR_END) * sizeof(double *)));
        if (!m)
                nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;
        /* allocate rows and set pointers to them */
        m[nrl] = (double *) malloc((size_t)((nrow * ncol + NR_END) * sizeof(double)));
        if (!m[nrl])
                nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;
        for (i = nrl + 1; i <= nrh; i++)
                m[i] = m[i - 1] + ncol;
        /* return pointer to array of pointers to rows */
        return m;
}

/* free an int vector allocated with ivector() */
void free_ivector(int *v, long nl, long nh)
{
  free((FREE_ARG)(v + nl - NR_END));
}

/* free a double vector allocated with dvector() */
void free_dvector(double *v, long nl, long nh)
{
  free((FREE_ARG)(v + nl - NR_END));
}

/* free an int matrix allocated by imatrix() */
void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch)
{
  free((FREE_ARG)(m[nrl] + ncl - NR_END));
  free((FREE_ARG)(m + nrl - NR_END));
}

void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch)
{
        free((FREE_ARG)(m[nrl] + ncl - NR_END));
        free((FREE_ARG)(m + nrl - NR_END));
}


/* standard error handler */
void nrerror(char *error_text)
{
  fprintf(stderr,"%s\n",error_text);
  exit(0);
}
```

```
void mapping(int **map)
{
  int ic, ix, iy, iz, ixn, iyn, izn;
  for (ix = 0; ix < Mcx; ix++)
    for (iy = 0; iy < Mcy; iy++)
      for (iz = 0; iz  < Mcz; iz++)
      {
        ic = icell(ix, iy, iz);
        for (ixn = 0; ixn < Mnx; ixn++)
          for (iyn = 0; iyn < Mny; iyn++)
            for (izn = 0; izn < Mnz; izn++)
              map[ic][ineig(ixn, iyn, izn)] = icell(ix + ixn - 1, iy + iyn -  1, iz + ↩
                  izn - 1);
      }
}

int icell(int ix, int iy, int iz)
{
  return ((ix + Mcx) % Mcx) +
       ((iy + Mcy) % Mcy) * Mcx +
       ((iz + Mcz) % Mcz) * Mcy * Mcx;
}

int ineig(int ixn, int iyn, int izn)
{
  return ixn +
       iyn * Mnx +
       izn * Mnx * Mny;
}

int box_rcut(double box, double rcut)
{
  int boxrcut;

  boxrcut = (int)(box / rcut) - 0;

  return (boxrcut > 3) ? boxrcut : 1;
}

void make_list(double boxx, double boxy, double boxz, double cellx, double celly, ↩
    double cellz, int **map, int *icc, int *n0, int **list0, int *n1, int **list1, int ↩
    *freq)
{
  int i, j, ic, in, ix, iy, iz;
  double cellinvx,  cellinvy,  cellinvz;

  cellinvx = 1 / cellx;
  cellinvy = 1 / celly;
  cellinvz = 1 / cellz;

  for (ic = 0; ic < Nc; ic++)
    n0[ic] = 0;

  *freq = 0;
  for (i = 0; i < NN; i++)
  {
    ix = (int)((rx[i] + 0.5 * boxx) * cellinvx);
    iy = (int)((ry[i] + 0.5 * boxy) * cellinvy);
    iz = (int)((rz[i] + 0.5 * boxz) * cellinvz);

    ic = icell(ix, iy, iz);

    if (ic != icc[i])
```

```
    {
      icc[i] = ic;
      (*freq)++;
    }

    list0[ic][n0[ic]++] = i;

    if (n0[ic] > N0)
      nrerror("You have to increase MAXNPC!");
  }

  for (i = 0; i < NN; i++)
  {
    ic = icc[i];

    n1[i] = 0;
    for (in = 0; in < Nn; in++)
      for (j = 0; j < n0[map[ic][in]]; j++)
        if (list0[map[ic][in]][j] != i)
          list1[i][n1[i]++] = list0[map[ic][in]][j];

    if (n1[i] > N1)
      nrerror("You have to increase MAXNPN!");
  }

      /*  printf("Update frequency %d\n",*freq); */
}

void potential(double rxij, double ryij, double rzij, double rmin, double rcut, double ↩
    *vij, double *wij)
{
  double rminsq, rcutsq, rijsq, vij2, vij6;

  rminsq = rmin * rmin;
  rcutsq = rcut * rcut;

  rijsq = rxij * rxij + ryij * ryij + rzij * rzij;

  if ((rminsq <= rijsq) && (rijsq < rcutsq))
  {
    vij2 = 1 / rijsq;
    vij6 = vij2 * vij2 * vij2;

    *vij = 4 * vij6 * (vij6 - 1);
    *wij = 16 * vij6 * (vij6 - 0.5);
  }
  else
    *vij = *wij = 0.;
}

void molecule(int i, double **rmx, double **rmy, double **rmz)
{
            rmx[i][0] = rx[i] - R0/2 * (pom1[i] * pom5[i]);
                  rmy[i][0] = ry[i] + R0/2 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  ↩
                      pom6[i]);
                  rmz[i][0] = rz[i] - R0/2 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * ↩
                      pom1[i]);

      rmx[i][1] = rx[i] + R0/2 * (pom1[i] * pom5[i]);
            rmy[i][1] = ry[i] - R0/2 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  pom6[i↩
                ]);
      rmz[i][1] = rz[i] + R0/2 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * pom1[i]);

      rmx[i][2] = rx[i] - R0 * (pom1[i] * pom5[i]) + R1/2 * (pom2[i] * pom5[i]);
```

```
                rmy[i][2] = ry[i] + R0 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  ↩
                    pom6[i]) + R1/2 * (pom1[i] * pom3[i] + pom4[i] * pom2[i] * pom6[i↩
                    ]);
                rmz[i][2] = rz[i] - R0 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * ↩
                    pom1[i]) - R1/2 * (pom1[i] * pom4[i] - pom3[i]
* pom2[i] * pom6[i]);

      rmx[i][3] = rx[i] + R0 * (pom1[i] * pom5[i]) + R1/2 * (pom2[i] * pom5[i]);
                rmy[i][3] = ry[i] - R0 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  ↩
                    pom6[i]) + R1/2 * (pom1[i] * pom3[i] + pom4[i] * pom2[i] * pom6[i↩
                    ]);
                rmz[i][3] = rz[i] + R0 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * ↩
                    pom1[i]) - R1/2 * (pom1[i] * pom4[i] - pom3[i]
* pom2[i] * pom6[i]);

      rmx[i][4] = rx[i] - R0/2 * (pom1[i] * pom5[i]) + R1 * (pom2[i] * pom5[i]);
            rmy[i][4] = ry[i] + R0/2 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  pom6[i↩
                ]) + R1 * (pom1[i] * pom3[i] + pom4[i] * pom2[i] * pom6[i]);
            rmz[i][4] = rz[i] - R0/2 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * pom1[i↩
                ]) - R1 * (pom1[i] * pom4[i] - pom3[i]
* pom2[i] * pom6[i]);

      rmx[i][5] = rx[i] + R0/2 * (pom1[i] * pom5[i]) + R1 * (pom2[i] * pom5[i]);
            rmy[i][5] = ry[i] - R0/2 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  pom6[i↩
                ]) + R1 * (pom1[i] * pom3[i] + pom4[i] * pom2[i] * pom6[i]);
            rmz[i][5] = rz[i] + R0/2 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * pom1[i↩
                ]) - R1 * (pom1[i] * pom4[i] - pom3[i]
* pom2[i] * pom6[i]);

      rmx[i][6] = rx[i] - R0 * (pom1[i] * pom5[i]) - R1/2 * (pom2[i] * pom5[i]);
            rmy[i][6] = ry[i] + R0 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  pom6[i])↩
                 - R1/2 * (pom1[i] * pom3[i] + pom4[i] * pom2[i] * pom6[i]);
            rmz[i][6] = rz[i] - R0 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * pom1[i]) ↩
                + R1/2 * (pom1[i] * pom4[i] - pom3[i]
* pom2[i] * pom6[i]);

      rmx[i][7] = rx[i] + R0 * (pom1[i] * pom5[i]) - R1/2 * (pom2[i] * pom5[i]);
                rmy[i][7] = ry[i] - R0 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  ↩
                    pom6[i]) - R1/2 * (pom1[i] * pom3[i] + pom4[i] * pom2[i] * pom6[i↩
                    ]);
                rmz[i][7] = rz[i] + R0 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * ↩
                    pom1[i]) + R1/2 * (pom1[i] * pom4[i] - pom3[i]
* pom2[i] * pom6[i]);

      rmx[i][8] = rx[i] - R0/2 * (pom1[i] * pom5[i]) - R1 * (pom2[i] * pom5[i]);
            rmy[i][8] = ry[i] + R0/2 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  pom6[i↩
                ]) - R1 * (pom1[i] * pom3[i] + pom4[i] * pom2[i] * pom6[i]);
            rmz[i][8] = rz[i] - R0/2 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * pom1[i↩
                ]) + R1 * (pom1[i] * pom4[i] - pom3[i]
* pom2[i] * pom6[i]);

      rmx[i][9] = rx[i] + R0/2 * (pom1[i] * pom5[i]) - R1 * (pom2[i] * pom5[i]);
                rmy[i][9] = ry[i] - R0/2 * (pom2[i] * pom3[i] - pom1[i] * pom4[i] *  ↩
                    pom6[i]) - R1 * (pom1[i] * pom3[i] + pom4[i] * pom2[i] * pom6[i])↩
                    ;
            rmz[i][9] = rz[i] + R0/2 * (pom2[i] * pom4[i] + pom3[i] * pom6[i] * pom1[i↩
                ]) + R1 * (pom1[i] * pom4[i] - pom3[i]
* pom2[i] * pom6[i]);

}

void total_energy(double rmin, double rcut, double boxx, double boxy, double boxz, int ↩
    *n1, int **list1, double *v, double *w)
{
```

```
  int i, j, p, r;
  double boxinvx, boxinvy, boxinvz, rxi, ryi, rzi, rxij, ryij, rzij, vij, wij;

  boxinvx = 1 / boxx;
  boxinvy = 1 / boxy;
  boxinvz = 1 / boxz;

  *v = *w = 0.;

  /* loop over all the pairs of molecules */
  for (i = 0; i < NN; i++)
  {
    for (j = 0; j < n1[i]; j++)
           {
        int k = list1[i][j];
                          for (p=0; p < 10; p++)
                        for (r=0;r < 10; r++)
              {
          rxij=rmx[i][p]-rmx[k][r];
          ryij=rmy[i][p]-rmy[k][r];
          rzij=rmz[i][p]-rmz[k][r];

          rxij = rxij - round(rxij * boxinvx) * boxx;
          ryij = ryij - round(ryij * boxinvy) * boxy;
          rzij = rzij - round(rzij * boxinvz) * boxz;

          potential(rxij, ryij, rzij, rmin, rcut, &vij, &wij);

          if ((p <=1) && (r<=1)) {vij = 0.594 * vij;  wij = 0.594 * wij;}
               else if ((p<=1) || (r <=1)) {vij = 0.77 * vij;  wij = 0.77 * wij;}

          *v += vij;
          *w += wij;
          }

      }
  }
}


void energy(int i, double rxi, double ryi, double rzi, double fii, double tetai, double↩
     omegai, double rmin, double rcut, double boxx, double boxy, double boxz, int *n1, ↩
    int **list1, double *v, double *w, double *max)
{
  int j, p, r;
  double boxinvx, boxinvy, boxinvz, rxij, ryij, rzij, vij, wij, rxipom, ryipom, rzipom,↩
       p1, p2, p3, p4, p5, p6, vmax, wmax;

  boxinvx = 1 / boxx;
  boxinvy = 1 / boxy;
  boxinvz = 1 / boxz;

  int k;


  *v = *w = 0.;

    vmax=-100;
    wmax=-100;
    *max=-100;

  /* loop over all molecules except i */

    for (j = 0; j < n1[i]; j++)
```

```
          {
        k = list1[i][j];
                        for (p=0;p < 10; p++)
                      for (r=0;r < 10; r++)
              {
          rxij=rmx[i][p]-rmx[k][r];
          ryij=rmy[i][p]-rmy[k][r];
          rzij=rmz[i][p]-rmz[k][r];

          rxij = rxij - round(rxij * boxinvx) * boxx;
          ryij = ryij - round(ryij * boxinvy) * boxy;
          rzij = rzij - round(rzij * boxinvz) * boxz;

          potential(rxij, ryij, rzij, rmin, rcut, &vij, &wij);

          if ((p <=1) && (r<=1)) {vij = 0.594 * vij;  wij = 0.594 * wij;}
               else if ((p<=1) || (r <=1)) {vij = 0.77 * vij;  wij = 0.77 * wij;}


          *v += vij; if ((vij > vmax) && (abs(vij)> 10e-2)) vmax=vij;
          *w += wij;
          }

      }


*max=vmax;


}

double ran2(long *idum)
{
  long IM1=2147483563,IM2=2147483399,IMM1=(IM1-1),IA1=40014,IA2=40692,IQ1=53668,IQ2↩
      =52774,IR1=12211,IR2=3791,NTAB=32,NDIV=(1+IMM1/NTAB);
  double AM=(1.0/IM1),EPS=1.2e-7,RNMX=(1.0-EPS);
  int j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[32];
  double temp;

  if (*idum <= 0) {
    if (-(*idum) < 1) *idum=1;
    else *idum = -(*idum);
    idum2=(*idum);
    for (j=NTAB+7;j>=0;j--) {
      k=(*idum)/IQ1;
      *idum=IA1*(*idum-k*IQ1)-k*IR1;
      if (*idum < 0) *idum += IM1;
      if (j < NTAB) iv[j] = *idum;
    }
    iy=iv[0];
  }
  k=(*idum)/IQ1;
  *idum=IA1*(*idum-k*IQ1)-k*IR1;
  if (*idum < 0) *idum += IM1;
  k=idum2/IQ2;
  idum2=IA2*(idum2-k*IQ2)-k*IR2;
  if (idum2 < 0) idum2 += IM2;
  j=iy/NDIV;
  iy=iv[j]-idum2;
  iv[j] = *idum;
```

```
 if(iy < 1) iy += IMM1;
  if ((temp=AM*iy) > RNMX) return RNMX;
  else return temp;
 }

void jacobi(double **a, int n, double d[],double  **v, int *nrot)
{
  int j,iq,ip,i;
  double tresh,theta,tau,t,sm,s,h,g,c;
        double *b,*z;

  b=dvector(1,n);
  z=dvector(1,n);

  for (ip=1;ip<=n;ip++) {
    for (iq=1;iq<=n;iq++) v[ip][iq]=0.0;
    v[ip][ip]=1.0;
  }
  for (ip=1;ip<=n;ip++) {
    b[ip]=d[ip]=a[ip][ip];
    z[ip]=0.0;
  }
  *nrot=0;
  for (i=1;i<=50;i++) {
    sm=0.0;
    for (ip=1;ip<=n-1;ip++) {
      for (iq=ip+1;iq<=n;iq++)
        sm += fabs(a[ip][iq]);
    }
    if (sm == 0.0) {
      free_dvector(z,1,n);
      free_dvector(b,1,n);
      return;
    }
    if (i < 4)
      tresh=0.2*sm/(n*n);
    else
      tresh=0.0;
    for (ip=1;ip<=n-1;ip++) {
      for (iq=ip+1;iq<=n;iq++) {
        g=100.0*fabs(a[ip][iq]);
        if (i > 4 && (double)(fabs(d[ip])+g) == (double)fabs(d[ip])
          && (double)(fabs(d[iq])+g) == (double)fabs(d[iq]))
          a[ip][iq]=0.0;
        else if (fabs(a[ip][iq]) > tresh) {
          h=d[iq]-d[ip];
          if ((double)(fabs(h)+g) == (double)fabs(h))
            t=(a[ip][iq])/h;
          else {
            theta=0.5*h/(a[ip][iq]);
            t=1.0/(fabs(theta)+sqrt(1.0+theta*theta));
            if (theta < 0.0) t = -t;
          }
          c=1.0/sqrt(1+t*t);
          s=t*c;
          tau=s/(1.0+c);
          h=t*a[ip][iq];
          z[ip] -= h;
          z[iq] += h;
          d[ip] -= h;
          d[iq] += h;
          a[ip][iq]=0.0;
          for (j=1;j<=ip-1;j++) {
            ROTATE(a,j,ip,j,iq)
```

```
          }
          for (j=ip+1;j<=iq-1;j++) {
            ROTATE(a,ip,j,j,iq)
          }
          for (j=iq+1;j<=n;j++) {
            ROTATE(a,ip,j,iq,j)
          }
          for (j=1;j<=n;j++) {
            ROTATE(v,j,ip,j,iq)
          }
          ++(*nrot);
        }
      }
    }
    for (ip=1;ip<=n;ip++) {
      b[ip] += z[ip];
      d[ip]=b[ip];
      z[ip]=0.0;
    }
  }
  nrerror("Too many iterations in routine jacobi");
}

double nematic()
{
double **Q;
double **vec;
double *ev;
int *rot;
double par=1;
int i,j;

Q=dmatrix(0,3,0,3);
vec=dmatrix(0,3,0,3);
rot=ivector(0,3);
ev=dvector(0,3);

for(i=0;  i<3; i++)
    for(j=0;  j<3; j++)
        Q[i][j]=0;

for(i=0;  i<NN; i++)
{
    Q[0][0] += 1.5  * pom1[i]*pom4[i] * pom1[i]*pom4[i] - 0.5;
    Q[0][1] += 1.5  * pom1[i]*pom4[i] * pom2[i]*pom4[i];
    Q[0][2] += 1.5  * pom1[i]*pom4[i] * pom3[i];
    Q[1][0] += 1.5  * pom1[i]*pom4[i] * pom2[i]*pom4[i];
    Q[1][1] += 1.5  * pom2[i]*pom4[i] * pom2[i]*pom4[i] - 0.5;
    Q[1][2] += 1.5  * pom2[i]*pom4[i] * pom3[i];
    Q[2][0] += 1.5  * pom1[i]*pom4[i] * pom3[i];
    Q[2][1] += 1.5  * pom2[i]*pom4[i] * pom3[i];
    Q[2][2] += 1.5  * pom3[i]*pom3[i] - 0.5;
}
for(i=0;  i<3; i++)
    for(j=0;  j<3; j++)
        Q[i][j]=Q[i][j]/NN;

jacobi(Q, 3, ev, vec, rot);

par=ev[0];
if (ev[1]>par) par=ev[1];
if (ev[2]>par) par=ev[2];
free_dmatrix(Q,0,3,0,3);
free_dmatrix(vec,0,3,0,3);
```

```
free_ivector(rot,0,3);
free_dvector(ev,0,3);

return par;

}
```